

Interlisp-D fixed allocations: conversion to Intermezzo

(all numbers in octal)

Name	Old Size (pages)	Old Addr	New Size	New Addr	Old Real	New Real
<i>On page 26:</i>						
Interface Page	1	26,10000	1	6,0	3	3
Primary Page Map	2	26,0	10	6,1000	2153	3313
StatsSpace (unused)	2	26,120000	omit			
Interrupt Table (unused)	?	26,121200	omit			
MiscStats	2 (1 used)	26,122000	2	6,5000		
UFN Table	2	26,123000	2	6,6000		
DTD ****	20	26,124000	20 (140)	6,10000		
MDS Type Table	40	26,100000	1/2 seg	6,100000	1600	2400
FPTOVP	1/4 seg	26,40000	1 seg	4,0	501	2000
<i>Misc:</i>						
Secondary Page Map	1/4 seg	25,0	1 seg	5,0		
Stack	1 seg	27,0	1 seg	1,0	1000	1400
GC Hash table	1/2 seg	73,0	1/2 seg	20,0	1400	2600
GC Collision *	1 seg	74,0	1 seg	21,0		
GC Overflow **	1	73,100000	1	20,100000	1640	3000
GC Big Ref	1-?	73,100400	1-?	20,100400		
Display Bitmap	312	76,0	312	22,0	1641	3001
LockedPageTable	—	(26,20000)	20	6,70000		
Map (Dlion only)	100	—	400	—	400	400
IOPage (Dlion only)	1	0,177400	1	0,177400	500	1000
SmallPosP's	1 seg	16,0	1 seg	16,0		
SmallNegs	1 seg	17,0	1 seg	17,0		
Arrayspace Start		40,0		23,0		
<i>Atoms:</i> (if 64K atoms)						
Pname Pointers	1 seg	20,0	2 seg	10,0		
Definitions	1 seg	21,0	2 seg	12,0		
Topvals	1 seg	22,0	2 seg	14,0		
Property Lists	1 seg	23,0	2 seg	2,0		
Atom Hash Table	1/2 seg	24,0	1 seg	7,0		
Pname Chars ***	8 seg	30,0	6 seg	72,0		

* Collision table occupies 1 segment, all preallocated, for no particularly good reason. It wants to be big, because once it fills up, you have to disable gc. I have seen the table get as large as a quarter segment. Current algorithms prevent it from being larger than one segment, but it would be easy to make it 2 segments long.

** GC Overflow table is actually just a few words. Current microcode relies on it being in the same segment as GC Hash, but this is not very important.

*** Pname char space is currently far too large for 32K litatoms; it might be about right for 64K, but we plan to dispose of it when pnames are hunked (taken as allocblocks), leaving just enough to get thru MAKEINIT.

**** Want to allow a little extra space for DTD in case we expand number of datatypes. This layout allows us to expand from 256 datatypes (8 bits) to 1536 datatypes (11 bits) before bumping into the LockedPageTable.

Further notes, June 1986 (post-Koto):

Pname char space now gone—all pnames are allocated from hunks.

Atom Hash Table address range used also for cml Character type (an immediate). With packages, atom hash table will go away eventually.

problem type: Performance
Subject: Want faster GETPROP
subsystem: microcode

GETPROP could be open coded faster. It is time-critical for a number of user functions. GETPROP would be faster if PUTPROP put new properties on the front of symbols PList instead of the back.

subsystem: microcode

TYPENAME is too slow, and is used by TYPENAMEP. Want primitive in microcode which is JNTYPENAMEP [alpha, beta, offset], like DTEST except jumps instead of traps if type doesn't match.

subsystem: compiler

Some of the initial constants and global variables can now be expanded inline for faster execution. Do stats on system, and look at GLOBALVAR references of functions which show up in profile.

Date: 12 Feb. 1982 8:53 am PST (Friday)
From: Moran.PA
Subject: CLOSEF problem

CLOSEF doesn't seem to work on the Dolphin in functions that work perfectly well on Maxc. My style is the following:

(LAMBDA (F)...(OUTFILE F)...(PRINTOUT NIL...)(CLOSEF F)...)

When it gets to CLOSEF, it says that file val-of-F is not open. However, calling CLOSEALL in the break does close file val-of-F, which really was open and which did indeed get all the printout intended.

Tom

Date: 1 March 1982 7:20 pm PST (Monday)
From: Bobrow.PA

The compiler seems to me to be much too verbose. Cannot there be a flag so that

it only says things when either a) there is a free variable b) an undefined function called

Currently I scan my dribble file for these, which are most useful. But so little wheat and so much chaff.

Date: 19 March 1982 9:59 am PST (Friday)
From: VanLehn.PA
Subject: HELPFLG bug
To: Lispbug↑.pa
cc: VanLehn
Reply-To: VanLehn

LISPX seems to be rebinding HELPFLG so that SETQ's at top level have no effect. I can't force errors to break by setting it to BREAK!, which makes debugging under errorsets damn near impossible.

Kurt

Date: 19 Mar 1982 1006-PST
 From: Neil Goldman <GOLDMAN at USC-ISIF>
 Subject: CONSTANT
 To: MASINTER at PARC-MAXC

In the macro for CONSTANT, I propose that the evaluation of the form be done under ERRORSET protection, with an error causing the thing to be treated as a DEFERREDCONSTANT, just as is done when the value fails to pass the CONSTANTOK test.

neil

Subject: SEPRCASE/GETBRK bug
 To: LispCore↑

If a character X is defined as a macro in readtable Y, then (SYNTAXP X 'BREAK Y) is NIL, but X is not in (GETBRK Y). I don't know whether this is a bug or a feature, but in any case it means that SEPRCASE treats such a char as an alphabetic, and FINDCALLERS and friends will miss when the target atom is preceded by the macro character.

Date: 8 Apr 1982 1932-PST
 Subject: SPECVARS and the compiler.
 From: RBATES at USC-ISIB

The manual on page 18.21 states "Whenever bcompl or brecompile encounter a block declaration they rebind retns, specvars ... to their top level value". This is NOT true as far as SPECVARS are concerned. The reason is the function LOCALVARS in COMP seeing that LOCALVARS are T sets SPECVARS to SYSSPECVARS without checking what the current value of SPECVARS is. This bug has been around since Sept 1976! These two functions with the coms show off the bug:

```
(FOO (LAMBDA NIL (PROG (X) (FUM]
(FUM (LAMBDA NIL (NILL X)])
(SETQ BUGCOMS ' ((FNS FOO FUM)
                  (SPECVARS X)
                  (BLOCK (FOOBLOCK FOO FUM (ENTRIES FOO]
```

/Ray

Date: 19 Apr 1982 0956-PST
 Subject: DECLARE:
 From: RBATES at USC-ISIB

I noticed that DECLARE: don't get compiled away (the function DECLARE: always get called), but DECLARE does get compiled away. This problem has been around awhile. Also that the example on the end of page 23.16:

```
(FOR X IN Y (DECLARE: (LOCALVARS X)) -- )
```

doesn't work.

/Ray

Date: 19 MAY 1982 1535-PDT
 From: KAPLAN
 Subject: \TAKEINTERRUPT skeleton on AINTERRUPT

I defined a skeleton for the \TAKEINTERRUPT macro on AINTERRUPT.

It has dummy calls to 2 primitives, one for checking whether \INTERRUPTABLE is T everywhere above its lowest binding (which is predictably NIL by the client of \TAKEINTERRUPT). I can simulate this with a stack search using a constant stack pointer, but this probably should be done at a lower level.

The other is a function for calling INTERRUPTED (I called it \CALLINTERRUPT). I don't think this can be the same as \CAUSEINTERRUPT. Maybe it is sufficient simply to branch to the keyboard context here.

I noticed that there are 2 global variables used mark that an interrupt is pending, \InterruptChar (used in the keyboard handler) and \INTCHAR, used in INTERRUPTED. \CAUSEINTERRUPT clears \InterruptChar and sets \INTCHAR. I don't quite understand this--is it temporary cause we haven't committed to WIND?

--Ron

 Date: 28 MAY 1982 0003-PDT
 From: KAPLAN
 Subject: Bug in BCOMPL/BRECOPILE
 To: MASINTER

I noticed that BRECOMPILE and BCOMPL setup LOCALVARS slightly differently.

BCOMPL initializes it to SYSLOCALVARS. BRECOMPILE does that only if LOCALVARS is T. If it is not T, it sets it to (UNION SYSLOCALVARS LOCALVARS).

Do you understand this? Which is correct, or are both correct?

If the UNION makes sense, should it also happen for SPECVARS?

--Ron

 Date: 15 JUN 1982 2210-PDT
 From: MASINTER.PA
 Subject: INSPECT scrolling
 To: lispsupport

If you put up an inspect window, and then change radix (e.g., from RADIX(8) to RADIX(10)), you get inconsistent output when you scroll the window.

Larry

We need to have an updated suite of tests to give to the technicians for hardware checkout.

Date: 18 June 1982 9:18 am PDT (Friday)
 FROM: MANN
 Subject: Runmicrotest.cm/replacement

Can you send us a message about Runmicrotest.cm or a suitable replacement to use in checking out the Dorado's as we discussed the other day. We do need this to do a good verification that the machines we ship run all the emulators.

Interlisp-D I believe has never been verified to run the DISRAEL Y-FUN test of spaghetti stack operation.

Date: 23 Jun 1982 1332-EDT
 From: DISRAEL at BBNG
 To: masinter at PARC-MAXC

(1) YY2 is the Interlisp version of the standard fixed point

(recursion) operator in the lambda calculus. As written in lambda calculus form, it looks like this: LAMBDA F. (LAMBDA X. F (X X)) (LAMBDA X. F (X X)). You apply it to a functional - a function that returns a function as value - and then apply the result to, say a number. So if you give YY2 the factorial functional - everyone's favorite example : LAMBDA FUN. (LAMBDA N. IF N = 0, 1, ELSE TIMES N (FUN N-1)) - you get something which when applied to 3 yields 6. It does this bit of magic by unwinding an internal lambda expression 4 times (in this case). So it simulates recursion by as much iteration as you need. Note: the definition of FACTFUN is not syntactically recursive, and if one defines FACT as (YY2 'FACTFUN), then FACT is also not recursively defined.

One can actually think of YY2 (never mind why it's called that and not Y) as the limit (the least fixed point) of an infinite series, starting with Y-0 (which see) and getting on by applying Yn to G to get Yn+1. (Moreover Yn = (G 'Yn) - so every Y is a fixed point of G).

Z is another, slightly unstandard recursion operator - written in lambda calculus form as follows: LAMBDA F. ((LAMBDA X. F (LAMBDA Z. (X X) Z)) (LAMBDA X. F (LAMBDA Z. (X X) Z))). Again Z of FACTFUN is a non-recursively defined FACTORIAL applicable to numbers.

(3) As for COMBOY - it's the Y-type recursion operator written out purely in terms of the two (so-called) primitive combinators. K (LAMBDA X. (LAMBDA Y. X)) and S (LAMBDA X. (LAMBDA Y. (LAMBDA Z. ((X Z) (Y Z))))). But there is a bug in the code: as it stands, it is not applicable to numbers - only to functions; e.g. not to 3 but to LAMBDA.() 3 - and of course TIMES, SUB1, etc. barf at these.

(4) Speaking of combinators; BB is functional composition (in disguise), SKIAPPLY is function application and SKIAPPLY2 is "APPLY" - again in disguise. (It takes a function and an arg as arguments; SKIAPPLY takes a function and returns a function which is the argument function to SKIAPPLY primed for application. (Baroque, eh??). WW takes a function and produces a function which when applied to an argument, produces a version of a two-placed function whose two arguments are identified. (So SQUARE is WW applied to TIMES.)

(5) F and J are weird functionals of purely theoretical interest (unlike the others which are, as you'll surely allow, of immense practical import). J is a function provably equal to I (LAMBDA X. X); but which is, unlike I, provably non-normalizable. I, moreover, is provably the only fixed point of F. (I think J, like COMBOY, may require functions as arguments all the way down.)

To TEST:

(APPLY* (YY2 'FACTFUN) 3) will do nicely to compute (factorial 3). (The same goes for (APPLY* (Z 'FACTFUN) 3).) You can go (APPLY* (FACTFUN 'FACTORIAL) 3) - where FACTORIAL is the regular recursively defined factorial function. And, since YY2 (or Z) are fixed point operators (so F = YF) you can go (APPLY* (APPLY* 'FACTFUN (YY2 'FACTFUN)) 3). ETC...

Date: 27 June 1982 5:53 pm PDT (Sunday)
From: vanMelle.PA
Subject: incompatible changes

incompatible changes for whenever we feel like introducing an incompatible change:

Rearrange InterfacePage so that IFPFaultHi is even-aligned.

Rearrange DataTypeDescriptors so that DTDFREE, DTDCNT0 and DTDNEXTPAGE are all in the same quadword.

Make htfind xor the hiloc of the datum when computing the hash probe.

Date: 30 JUN 1982 0755-PDT
From: SPROULL
Subject: Dolphin experience

[This was a long report on Bob's experience with the Dolphin. I have excerpted the problems which I think are still relevant]

- - - - -
The bad news

My view is that Interlisp is sinking of its own weight, and the move onto a personal computer has shown the hulk in alarming vividness. This section presents a brief justification of this view and offers possibilities for remedies. The problem can be fixed, but it may be costly.

The problem, as I see it, is that Interlisp has never had a clean internal structure of the system (as separate from the language): features and packages have accreted, wired into the existing maze to create a tighter maze. It's now so bad that a good programmer who encounters an Interlisp system is at a loss for what to do for a good long time. Few if any of the "interfaces" in Interlisp correspond to things he recognizes from other environments. He has to seek out facilities one by one; his intuition for where to look is often wrong; and he remains worried about deep interactions among various parts of the system.

This problem has been made worse by the move onto a personal computer. I think to a great extent, new facilities are added to Interlisp using the same rather low standards of interface definition that have characterized Interlisp so far. For example, while I think the facilities provided by the Interlisp-D stuff for graphics are mostly OK, the interfaces can be substantially improved.

I feel the "system" part of Interlisp needs a thorough overhaul. I favor the "open system" approach in which a very few low-level primitives are built in, and the rest is done with packages that can be separated and that have well-defined interfaces.

I realize that an undertaking such as this is a big one. However, I believe that a great deal of the detailed functionality of Interlisp can be retained (even much of the code can be retained), but the interfaces need to be redesigned in light of the tremendous evolution of the system. It's remarkable that they've remained useful as long as they have, but they need overhaul. I see two hopeful signs:

1. To the extent possible, new work should be done in the form of LISPUSERS packages. I'm delighted, for example, to see the new editor done in this way. (But I suspect the interface between it and the rest of the system could be improved if the system were improved.) I think it's important that some (perhaps all) of these packages be distributed in source-file form so that users can actually understand what they do.
2. The new Common Lisp effort is an opportunity to redesign

many of these functions. Perhaps a long-range plan might be to put Common Lisp up on the Dolphin. Should the Xerox group be contributing to the design of Common Lisp, especially in those areas where it has the most expertise (e.g., user I/O)?

Another approach is to invest some effort in restructuring Interlisp. I think it might be worth a few days' effort to estimate the difficulty of this problem and the improvements that could be reasonably expected.

(long paragraph)

To summarize, I'd like a complete, consistent "graphics package" at the low level. Some of the pieces are there (bitblt, line, etc.), but I don't see the structure. It appears to be a complex set of stuff with complex inter-relations. There is no clear description of what a bitmap is, or what a display stream is, or what a window is. This needs to be cleaned up.

More to the present point, however, is that I find the manual almost completely lacking in discussion of concepts. To pick an example from the current manual, consider the file package. What is a "file" from the point of view of Lisp? What is its purpose? What does it contain? What is the distinction between what is remembered inside the Lisp virtual memory and what is retained on the disk? And so on and on . . . While the file package might have once started out so simple that none of these questions arose, it's long past that point now.

The new graphics stuff definitely needs a good deal such concept documentation. In the manual, old and new, there's too much emphasis on functions and not enough on structure and concepts.

The best manual would result from a system restructuring. The Interlisp language should form a distinct part of the manual -- some sections of the current manual are salvageable in this respect. If the interface between the language and the packages were cleaned up, this section would describe the interface. I, for one, would benefit enormously from a self-contained section that describes the language without reference to any of the system stuff.

Date: 8 July 1982 8:25 am PDT (Thursday)
From: VanLehn.PA
Subject: main data space overflow

....

I've tried to find the storage leaks using COUNTDOWN, MAPATOMS and so forth. So far, the only circularities I've come across are on LISPXISTORY. I need better leak-finding tools. One would be to do the mark & sweep part of garbage collection, including the freelist as "accessible" during the mark. The list that the sweep delivers is therefore all the cons cells that got leaked. By looking through the ones with atomic cars and cdrs, I could probably figure out from the pnames where the leaks came from.

4. Of course, having found all the lost storage, it could be put back on the freelist, saving me a reload (but probably still taking 20 minutes). Since there is plenty of array space around, the mark & sweep could be written

simply, using its own array to hold the mark bits. I don't see any reason to do the "copy" part of a "stop & copy" since swapping doesn't seem to be a problem on the Dorado (according to temporal intuition and control T).

5. If the mark's bit array is the screen bitmap, one could probably learn a lot about the storage use and maybe the chronology of the leaks by seeing not only where in vmem the leaks are, but watching the mark propagate out from specific atoms. A quick calculation has it that marking the current Mds, assuming it's mostly cons cells (95% in my program), would take a 900 by 900 bitmap.

I'd be willing to help code such a tool, or any other.

Any tools or diagnostic ideas would be welcomed with extreme enthusiasm.

Date: 8 JUL 1982 1155-PDT
From: ROACH.PA
Subject: DEFINEQ and MACROS
To: LISPSUPPORT
cc: ROACH

Dear Interlisp Support,

I think MACROS, DEFINEQ, and the Interlisp compiler interact incorrectly. I would like to define macros such as DEFEXPR, DEFFEXPR, etc. that can appear in files and will expand out into DEFINEQ forms which go on to be compiled like other DEFINEQ forms. I've been told by Ronald Kaplan that this won't work, and in fact, it doesn't. What is Interlisp's problem? I might point out that Maclisp does allow you to do this sort of thing. This is a pretty serious deficiency on Interlisp's part.

Secondly, instead of compiling expressions in the order in which they occur, the Interlisp compiler gathers functions definitions into a separate group from all other expressions. This is also a bug. (Again Maclisp does the right thing.) Compiled forms ought to load in the same order in which the uncompiled forms loaded.

I am hopeful that action will be taken on these problems.

Date: 27 JUL 1982 0935-PDT
From: KAPLAN.PA

. . .

We probably also ought to implement a device info interface that could, for example, tell how many pages are left on the disk, in an ifs directory, or in the ifs as a whole.

00348 00024 UU
Date: 10 Aug. 1982 8:48 am PDT (Tuesday)
From: VanLehn.PA
Subject: MARKASCHANGED
To: lispsupport
cc: VanLehn

MARKASCHANGED apparently doesn't inform the compiler that the function needs to be compiled. Reference manual doesn't specify whether it does or not. Obviously, it should tell the compiler to recompile.

Date: 26-AUG-82 11:46:38 PST
Subject: AWFUL CODE FROM CREATE USING
To: LispSupport

(create FOO using X) when FOO is a RECORD translates as a forest of CONS of CARS of CDRs rather than COPY. Produces awfully large chunks of code that doesn't even run fast. Bug?

 Date: 8-Sep-82 14:03:40 PDT (Wednesday)
 From: Masinter.PA
 Subject: Re: RAISE for files

Users want to be able to read a file in lower case as if it were in upper case.

Why don't we put a translation table into READ tables? We already have them for FILEPOS and FFILEPOS etc. This would make a lot of sense. The cost is relatively small.

 Date: 9-Sep-82 16:18:21 PDT (Thursday)
 From: Masinter.PA
 Subject: RESETLST vs RESETFORM
 To: Bobrow
 cc: LispSupport, Masinter.PA

This should be in the manual, or maybe we should fix RESETFORM.

Example: this doesn't work:
 (RESETFORM (DEFPRINT 'A 'FOO) stuff)

This is what DOES work:

(RESETLST (RESETSAVE NIL (LIST 'DEFPRINT 'A (DEFPRINT 'A 'FOO)))
 stuff)

Date: 10-Sep-82 8:48:59 PDT (Friday)

There sentiment for making TY not elide comments

Date: 14 Sept. 1982 9:41 am PDT (Tuesday)
 From: JonL.pa

I've never used TY, but if it does the obvious thing, then one might expect that TY* would be the command which doesn't elide comments (e.g., PF and PF*?)

We need to have an inbound CHAT server so you can CHAT to a machine from a remote terminal over the PUP and NS Ethernet.

We need to handle UNIX filenames better
 We need device synonyms and pseudo-devices
 We need to be able to delete 1100 {DSK} files without building the whole map

We need to document the facilities for doing Binary i/o for bitmaps, integers, floats. AOUT/AIN.

Compiler: [14 NOV 1981 1815-PST] (FUNCTION (LAMBDA --)) expression used as a value to be stored into a record slot. The compiler did produce a suitable subfunction, but then compiled as the value of the (FUNCTION --) expression the free variable NEWVALUE.

We need to fix Sandbarring

Storage management:

We need to fix the GC so that it collects items which are only held as keys of hash-tables. (CLISPARRAY in particular).

we need unwindprotect

many system functions have Names which can conflicts with user fns

We need to unify the handling of meta, blank keys

we need to clean up GLOBALVARS situation

We need more diagnostics and system tests

We need to fix it so that expanding macros doesn't take so long

* FUNARG doesn't work

* MASTERSCOPE:

WHO USES FILE FREE causes funny error messages

Date: 16 SEP 1982 1802-PDT

From: BURTON.PA

Subject: compiler bug

The bind merge optimization getts carried away with the function DSPDESTINATION on <LISPCORE>WIND>LLDISPLAY and binds the variable \INTERRUPTABLE with the variable DS. The effect is that the \DTEST is called in a context that is uninterruptable leading to a call to RAID when DSPDESTINATION is given a bad argument such as (DSPDESTINATION 123 (DSPCREATE)).

Date: 20 SEP 1982 1954-PDT

From: SHEIL.PA

Subject: Compiler bug - EVERY

Attempting to compile the expression (EVERY xxx (FUNCTION ATOM)) generates the compiler warning message (ATOM: Too many args for macro). Code seems to be OK but the message is distracting, especially if the code has come from a type? from either a record or DECLtype.

Beau

Date: 24-Sep-82 8:47:08 PDT (Friday)

From: Masinter.PA

Subject: Re: Problem with open leaf files -- and patch

In-reply-to: BOBROW's message of 21 SEP 1982 1517-PDT

To: LISPSUPPORT

cc: Bobrow, Stefik

Danny's patch to FINDOPENFILE seems to get around the immediate problem, but some more permanent fixes are needed. Ron and I talked about these problems for a while; I thought I would send out some notes on our conversation and some additional thoughts.

There are currently three separate problem areas in the current system:

- a) READ/PRINT given file names which are not fully qualified scans the directory every time, which is TERRIBLY SLOW
- b) for LEAF files, the file CANNOT BE FOUND by INFILEP/OUTFILEP/FINDFILE if it is already open
- c) There are a number of inconsistencies having to do with the use of

DIRECTORIES and the error mechanism to implement search paths.

Proposals:

- a) files which are presented to READ/PRINT (i.e., in a context where an OPEN file is required) will ONLY scan against the set of files which are open with appropriate access.

This is a change from Interlisp-10 semantics, where if you do an INFILE(FOO), and then create a NEW VERSION of FOO, and then do READ(FOO), you will get a FILE NOT OPEN error.

- b) we should implement the notion of a "search path" device, e.g. {LISPUSERS} and {SOURCES}. The system will support assigning search paths to a device (new function), so that one can say {SOURCES} = {PHYLUM}<LISPCORE>WIND> , {PHYLUM}<LISPCORE>SOURCES>

Doing an INFILEP on {SOURCES}xxxx will return a full filename of {PHYLUM}<LISPCORE>WIND>xxxx, i.e., the name returned will be fully qualified. OUTFILE on {SOURCES} will write on the FIRST directory on the search path.

The general idea here is to take what is currently done via the error mechanism and DIRECTORIES and FINDFILE and instead build it in in at a lower level. This will allow some more rational implementations of the facilities.

It will also make more logical the link between the "connected" directory and the search path; that is, one can either connect to {SOURCES} or to {WIND} or to {LISPUSERS}. It will remove the distinction between FINDFILE and INFILEP in the non-spelling-correction case.

Finally, all of this is relatively easily implemented in Interlisp-10! Interlisp-10 already supports a (undocumented) feature where if you PUTPROP(LISPUSERS DIRECTORIES (<LISPUSERS> <LISP>)) and attempt to FINDFILE(LISPUSERS:filename), it will in fact search those directories.

Comments?

Date: 24-Sep-82 8:49:48 PDT (Friday)
From: Masinter.PA
Subject: Re: Bitmap editor
In-reply-to: SHEIL's message of 21 SEP 1982 1603-PDT
To: SHEIL
cc: burton, lispsupport

I always wanted to do EDITBM on a "window".

One general way of handling this is to have a general "coersion" function which coerces to "clipped bitmap", with the relatively obvious coersions for windows/displaystreams/bitmaps/regions....

Date: 24 Sep 1982 1538-PDT
From: Friedland
Subject: interlisp d bug
To: cschmidt, rindfleisch

renamefile on {DSK} doesn't work. If you have a file A 100 pages long and a file B 200 pages long and (RENAMEFILE A B), you end up with a file B 200 pages long, its first 100 being the old A and the last 200 being garbage from the old B. You have to

(DEFINE B) before the renamefile. This despite what the manual says.

Peter

 Date: 29 SEP 1982 2012-PDT
 From: JONL.PA
 Subject: Undefined Function

Once in a while, I mistype a DEFINEQ, and wind up with an s-expression in the definition cell of some litatom which is *almost* what I wanted -- it just lacks the word LAMBDA. The error message you get when you try to run such a function is "Undefined Function" -- wouldn't it be better to reserve that msg for the case of a definition cell which is either NIL or NOBIND, and print something more informative for the case where it contains something like ((X) (LIST X (TIMES 2 X))), or ((() (PRINT 5))).

 Date: 29 SEP 1982 2016-PDT
 From: JONL.PA
 Subject: PUNting a broken compilation
 To: lisppbug↑

- 1) There needs to be an advertised way to do a BCOMPL which ignores errors which occur during the compilation of a single function, so that a single call to BCOMPL will proceed thru the while file, finding perhaps other errors before ending.
- 2) I tried the unadvertised function (PUNT) after one of my macros caused a BREAK during compilation, and it appeared as though the then-current break window became the normal TTY display stream.

 Date: 30 SEP 1982 1119-PDT
 From: SHEIL.PA
 Subject: two comments on the RESETFORM macro

Currently, the RESETFORM macro evaluates the resetform (a) outside of errorset protection and (b) some time (during which an interrupt can occur) before the resetlst entry for undoing it is made. This could be disastrous if one is resetting, for example, a display stream clipping region and the user bombed you out. [(a) may or may not be a bug or non-feature, depending on how one reads the manual; (b) is nasty to fix and probably requires interrupt protection]

Also, the RESETFORM macro doesn't do a very good job if one passes it a LAMBDA expression as the reset function (two copies wind up in the compiled code).

[The motivation for this is for two arg fns like DSPCLIPPINREGION, as
 (RESETFORM ((LAMBDA (X) (DSPCLIPPINGREGION X window)) NEWREG)
 forms)

is much more elegant than

(RESETLST (RESESAVE NIL (LIST 'DSPCLIPPINGREGION
 (DSPCLIPPINGREGION NEWREG window)
 window))
 forms)

In fact, since I just realized that, it might be worth noting this trick in the manual for other slow thinkers.]

*** I really want to shift to this notation in DEDIT, so this patch would be greatly appreciated ***

Beau

PS: From a slightly broader point of view, it would be nice to have a wizard scrutinize these macros as (a) this is not the first non-feature report for them (b) they don't look to be as good code as they could be. Last time I raised this, the discussion quickly expanded to include respecifying the

whole error handling machinery (and thus nothing happened). Perhaps a useful intermediate step would be to define a few more useful abstractions, such as CATCH and THROW, which we could start using in our code to replace the convoluted RESETLST constructions that tend to generate these discussions in the first place.

Date: 1 OCT 1982 1430-PDT
 From: SHEIL.PA
 Subject: Glitch in RENAME
 To: LISPSUPPORT

If one has a variable FOO which is used in some file BAR and you wish to rename it to FUM, (RENAME 'FOO 'FUM 'VARS 'BAR) will bomb with complaint "no VARS defn for FOO" unless FOO has a top level binding.

Beau

 00705 00024 UU
 Date: 1 OCT 1982 1611-PDT
 From: SHEIL.PA
 Subject: PP and PRETTYPRINT glitches
 To: LISPSUPPORT

Some time ago, PP (and PP* and PPT) had LOCALVARS declarations added so that their variables would not interfere with EVALVs from PRETTYPRINT. Unfortunately, the ERRORSET implementation causes all these to be SPECIAL in Interlisp-D anyway. Pending a more general resolution of this problem, it would be nice if these fns were patched to avoid the fact that (PP X) for example, does absolutely nothing. [Major motivation: This is irritating if you know what is going on but absolutely inexplicable if you dont].

Manual note: ARGLIST of PRETTYPRINT does not match manual spec.

 Date: 3-Oct-82 21:05:18 PDT (Sunday)
 From: Masinter.PA
 Subject: Re: PP and PRETTYPRINT glitches
 In-reply-to: SHEIL's message of 1 OCT 1982 1611-PDT
 To: SHEIL
 cc: LISPSUPPORT

An additional (more general) fix is for the compiler to rename the variables which are auto-SPECVARED because of the ERRORSET hack.

 Date: 3 OCT 1982 2327-PDT
 From: KAPLAN.PA
 Subject: Clispify/dwimify bug

If (don't ask why) the atoms < and > have top-level values, then CLISPIFY((LIST (FOO))) is (< (FOO) >) which then doesn't dwimify back.

As a minimum, this (and all) clisp transformations should not be performed in environments where they won't dwimify properly.

 Date: 4 OCT 1982 0514-PDT
 From: JONL.PA
 Subject: MASTERSCOPE Message

If you edit a macro, MasterScope will correctly tell you that

certain functions depend upon it; but when calling UNSAVEFNS, it always prints the message "Loading FooFunction", regardless wof whether it is really loading it, or merely UNSAVEDEFing it.

Date: 4 Oct. 1982 8:26 am PDT (Monday)
From: Stefik.PA
Subject: Re: Interlisp-D planning

Larry and Bill, -

The Lisp features {desired by Mark et al.) were

- (1) Ability to have functions without naming them.
- (2) Ability to hash on strings (or uninterred atoms?) for our LOOPS uids.
- (3) Access to an efficient (say B-tree) database (perhaps cedar?).

Of more immediate importance will be some participation by yous guys in design reviews of LOOPS, and consulting on performance tuning. Mark

Subject: lisp.tasks
* UNBREAK work on internal block functions just like TRACE
* DUMMYFRAMEP definition correction
* compiler optimization NEWOPTFLG=T
* free code deleterefs pointers therefrom

Interlisp-10 problems
* DELFILE BUG
* GETSTREAM
* CALLSCCODE returns duplicate values
* add ALLOCSTRING
* make NCREATE allocate system datatypes too

Date: 6 Oct. 1982 9:44 am PDT (Wednesday)
From: Bobrow.PA
Subject: Re: Interlisp-D planning
In-reply-to: Masinter's message of 5-Oct-82 19:45:57 PDT (Tuesday)
To: Masinter
cc: Stefik, Bobrow, vanMelle

1) Must function call always go through an atom? Perhaps the name slot on the stack could be made in the "naked" case to point to the fn data object.
Inspect
macros might allow at least finding out about arguments expected.
One might even have a string in such an object to name it (this would work for our methods).

2) The current atom hash table code made available as string hashing would do quite nicely at this stage for us, I think. More than 2^{16} atoms would be nice,
but is not right, since we want separate name spaces with overlapping names, and I don't think we need anything but string hashing.

3) Eventually we might want to use file based indexes for knowledge bases (e.g. BTrees) when they get too large. The current Alpine project is NOT planning to provide a BTree index interface at the moment (I checked with Mark Brown) Most of the indexing stuff is done now on the clients side of the Cedar database.

Stats runs show that our GetValue and PutValue are remarkably slow (about 250 microseconds on a Dorado) and take most of the time, as we expected. Help on redesign on that would be most welcome.

danny

Date: 4 OCT 1982 0514-PDT
 From: JONL.PA
 Subject: MASTERSCOPE Message
 To: LISPBUG↑

If you edit a macro, MasterScope will correctly tell you that certain functions depend upon it; but when calling UNSAVEFNS, it always prints the message "Loading FooFunction", regardless of whether it is really loading it, or merely UNSAVEDEFing it.

Date: 4-Oct-82 12:53:41 PDT (Monday)
 From: Masinter.PA
 Subject: lisp.tasks
 To: masinter
 cc: , Masinter.PA

- * Compiler: [14 NOV 1981 1815-PST] (FUNCTION (LAMBDA --)) expression used as a value to be stored into a record slot. The compiler did produce a suitable subfunction, but then compiled as the value of the (FUNCTION --) expression the free variable NEWVALUE.
- * Name conflicts with system fns
- * EVAL edit macro in editor needs ERSETQ instead of NLSETQ.
- * correct handling of meta, blank keys
- * periodicallyreclaim be sensitive to mouse events, process suspension.
- * Breakcheck problem associated with heavy swapping on first uba or udf.
- * improve interface to stats for other things than fn call
- * make STACK FULL non-fatal error
- * (APPEND circular) bug
- * Can create ARRAYS > 2↑16.
- * STORAGE) function prints garbage negative numbers in the 3rd column.
- * UNBREAK work on internal block functions just like TRACE
- * BRKDOWNRESULTS print out
- * interaction of code which rewrites filemaps and RADIX
- * rename low level functions
- * memory map diagnostics
- * DUMMYFRAMEP definition correction
- * Interlisp-10 problems:
 - * DELFILE BUG
 - * samedir has problems on tops20: directoryname neg filenamefield
 - * CALLSCCODE returns duplicate values
 - * add ALLOCSTRING
 - * make NCREATE allocate system datatypes too
- * Code for generating Interpress from Tops-20 and Vax (Troff, Scribe, ...)
- * small demo
- * LISPXSTATS
- * compiler optimization NEWOPTFLG=T
- * free code deleterefs pointers therefrom

Misc bugs

Masterscope recursion with long names
 BQUOTE

Date: 6-Oct-82 13:21:14 PDT (Wednesday)
 From: Masinter.PA
 Subject: questions & comments from Schoen
 To: LispSupport
 cc: Masinter.PA

document VRAID package; I think as a LispUsers package.

we should provide FLOUT and READ/WRITEBINARYBITMAP for fast i/o of floatps

and bitmaps.

Schoen has a VAXPRINT package (possibly similar to VanBuers) which causes the vax to hardcopy screen bitmaps. He doesn't know what the Versatec they use is exactly, but it has 2112 dots across, he says.

Date: 8 OCT 1982 1620-PDT
 From: SHEIL.PA
 Subject: Lisp task list
 To: MASINTER
 cc: lispcore↑

I just spent 15 mins reading it. Very comprehensive; fine job; don't envy you the task of prioritizing it! A couple of additional points:

ERRORSET compilation.

Possibility of marking the stack rather than introducing new function call. If not, the making compiler suppress or rename forced new specvars.

Error handling

Proposal to make ↑D work by ↑E (incompatible but worth it?)
 Some improvement over current mess.
 Failing that, debugging optimization of RESETLST/SAVE/FORM

Global vars

If RESETVARSLST is known not to be declared global, lets fix it rather than documenting it! Mike Sannella needs some help to get new manual to indicate which system params are global - maybe he could propose a list derived from the manual and we could dispose of this one.

Garbage collection

Compiled code blocks (pointers therefrom)
 Hash arrays [urgent; current incompatibility + perf problem]

File package

Hasdef problems
 Fixing EDIT interfaces to use same.

Beau

Date: 11-Oct-82 16:41:58 PDT (Monday)
 From: vanMelle.PA
 Subject: CASEARRAY for READ
 To: LispSupport

Yet another (perhaps the same) request for (RAISE T) for files...

 Mail-from: Arpanet host SU-SCORE rcvd at 11-OCT-82 1414-PDT
 Date: 11 Oct 1982 1402-PDT
 From: David E. Smith <CSD.SMITH at SU-SCORE>
 Subject: lower case
 To: vanmelle at PARC-MAXC
 Stanford Phone: (415)497-1809

I need a way of forcing interlisp to be case independent for file input as well as terminal input. Read macros won't do it because I don't want the lower case letters to be break characters and "ALWAYS" forces this. Advising or rewriting READC presumably wouldn't work either because strings and characters prefaced by "%" would then get upcased.

How do I do this? Am I forced to rewrite LOAD and all of its accomplices? Crufty and/or release dependent solutions will not be sneezed at. Help!

-- de2

 Date: 12-Oct-82 13:11:06 PDT (Tuesday)
 From: vanMelle.PA
 Subject: Font assumptions
 To: Burton
 cc: LispSupport

If you do (FONTSET 'STANDARD) to turn off fonts (e.g., to make a fontfree file), subsequent calls to the inspector die in a \DTEST of FONTDESCRIPTOR because DEFAULTFONT is NIL.

I wonder how many other places make such assumptions.

Incidentally, herewith a reminder that \DTESTFAIL desperately needs to produce a better error message, at least incorporating its second arg (the intended type).

Bill

Date: 14-Oct-82 16:36:57 PDT (Thursday)
 Subject: Re: Schlumberger URGENT
 In-reply-to: Raim.EOS's message of 14 Oct. 1982 10:24 am PDT (Thursday)

Eric (and users in general) should avoid doing (APPLY 'IMAX LST) and instead write (for X in LST maximum X).

The limit of number of arguments to a function is indeed 80; it is possible that we could bump it, but there still would be a fixed limit.

Larry

Date: 15-Oct-82 15:05:36 PDT (Friday)
 From: vanMelle.PA
 Subject: LARGEST/SMALLEST
 To: LispCore↑

It has been pointed out that these names are confusing, due to the ambiguity of what you might want the iterative to return. I propose that LARGEST be called MAXIMIZING, SMALLEST be MINIMIZING, and that there also be ops MAXIMUM and MINIMUM. Since FIND is a synonym of FOR, we could thus have:

```
(find X in L maximizing (FOO X))
      returns the X for which FOO is largest, and
(for X in L maximum (FOO X))
      returns the largest value of FOO over L.
```

Bill

Date: 23 Jan 89 16:12
From: Will Snow: AISNorth: Xerox
Subject: Places that use hiloc/loloc and what for
To: sybalsky: AISNorth: Xerox, shih: AISNorth: Xerox
cc: Will Snow: AISNorth: Xerox

I spent some time finding out who uses hiloc/loloc and what they use them for. the following is a summary:

10MBDECLS:

D0ETHERIOCB
DLEETHERIOCB

All iocb's are in the lowest addresses, so 10MBdecls takes advantage and only puts in the piece of the pointer necessary (loloc)

10MBDRIVER:

\RELEASE.IOCB change the "next iocb" field of the given iocb to a null value.
\INIT.ETHER.BUFFER.POOL change the "next iocb" fields to null.
\QUEUE.INPUT.IOCB next iocb ptr of last iocb = this iocb, or make this the first one.
\QUEUE.OUTPUT.IOCB rearranging the queue of iocb's

ABASIC:

EQUAL uses loloc as an optimization on fixp vs smallp comparisons.

ACODE:

CHANGECCODE uses hiloc, loloc during refcount operations
CODEBLOCKP uses hiloc to determine what segment of storage the piece being looked at is in.

ADDARITH:

MACRO .XUNBOX.

APRINT:

\PRINTADDR tries to print a lisp address nicely. Uses both HILOC and LOLOC.

ASTACK:

SETSTKNAME ? (HILOC)

CMLARRAY-SUPPORT:

MACRO %SMALLFIXP-SMALLPOSP converts smallfixp to a number. (LOLOC)

CMLCHARACTER:

ACCESSFNS CHARACTER how to create and access a common lisp character. (loloc)

CL:CHAR-CODE change a character into a #(Loloc)

DEFOPTIMIZER CL:CHAR-CODE fast changing of char into code. (loloc)

CL:CODE-CHAR fast checking for smallposp (hiloc)

DEFOPTIMIZER CL:CODE-CHAR fast checking for smallposp (hiloc)

CMLEVAL:

DEFSTRUCT CLOSURE :print-function to print the ptr (hiloc,loloc)

DEFSTRUCT ENVIRONMENT :print-function to print the ptr (hiloc,loloc)

set-symbol checks if environment is the stackhi

CMLSTRING

%%STRING-BASE-COMPARE-EQUAL get the character code from a string of CL:CHARS (loloc)

CMLUNDO

undoably-set-symbol determine if at top of stack. (hiloc)

D-ASSEM

FIXUP-PTR, FIXUP-PTR-NO-REF - ? (loloc,hiloc)

INTERN-DCODE ? (loloc)

DEBUGGER

PRINT-ENTRY-MESSAGE print the condition number...

DLAP - LOLOC/HILOC optimizers to u-code.

DOVEDECLS:

DEFMACRO \DoveIO.IORegionOffset get the right IO region on a dove.(loloc)

DOVEETHER

\DoveEther.Enqueue fill in the "next packet" field.(loloc)

DOVEINPUTOUTPUT

\DoveIO.MakeOpieAddress make the correct opie address out of a lisp addr.(hi,lo)

DTDECLARE:

COMPILEDREPLACEFIELD figure out what to do with an X pointer.

Note:: Both DLION and DOVE disk code also uses LOLOC and HILOC...

—End of message—

EVERYTHING YOU WANTED TO KNOW ABOUT THE AR DATA BASE - BUT WERE AFRAID TO QUERY

This document on {Pogo:}<Release Management>Kat>Doc>AR-How-To-Edit.tedit.
Last edited by Kat Kohlsaas on 9-Mar-1988 15:48:55

INTRODUCTION

The Action Request data base is the primary vehicle through which the state of Xerox Lisp, including outstanding problems, requested features, and the like, is tracked. Since ARs are the primary channel of communication between the user, customer support, marketing, and development, it is important that the maximum amount of correct information be compressed into each AR. This allows technical information to get to development, and just as importantly, get back out. This process can be facilitated by correct use of the fields of the AR.

THE AR FORM - THE FIELDS AND WHAT THEY MEAN

The basic component of the AR data base is the individual AR. An AR is the melding of a blank AR form with the data specifying a need. The AR form provides 31 areas, or fields, for the input of information giving a concise summary of the need. A need can be either a problem with the Xerox Lisp system that must be corrected, or a request for a feature that would improve the system if implemented. Since the structure of the AR form must be standardized to allow entry of a wide variety of needs, the data detailing the needs becomes an important component of the AR system. Correct use of the various fields comprising an AR facilitates the exchange of information between the submitter of the AR and the developer who will act upon the AR.

The FillInDefaults option of the left button menu associated with the AR Bug Report Editor title bar will fill in the **Submitter:**, **Source:**, **Status:**, **Machine:**, **Microcode Version:** and **Memory Size:** fields, and will place MAKESYSNAME as well as MAKESYSDATE in the **Lisp Version:** field. Please fill in the **Lisp Version:** field when submitting an AR, either by typing it in or by using FillInDefaults from the pop-up menu. The version of software the bug is being found in is important data.

Number:	Generated by the AR data base, every AR has a unique number. AR numbers are <i>never</i> recycled. ARs are never deleted. The AR number cannot be changed by the user.
Date:	The date the AR was originally submitted. <u>This is filled in by the system.</u>
Submitter:	The login name of the person who submitted this AR. <u>This is filled in by the system.</u>
Source:	The name of the person reporting the problem being documented in the AR. The name or names appearing in this field must give enough information to enable contact if needed, i. e., Doe.PASA or Doe@Berkeley.edu.
Subject:	A terse summary of the problem, providing both enough information to identify it uniquely and enough keywords for querying. "FOO doesn't work" or "Floppy problem" is not good enough. Think of yourself as a newspaper headline writer: "Attempt to write file when floppy door is open causes awful noise." Implementors may change the Subject: field as more details about the true nature of the problem become apparent.

As much as possible, relevant keywords should be included in the **Subject:** to facilitate querying on the data base. If the problem relates to a specific package, that package name should be mentioned in the **Subject:**. File names, commands, functions, error messages, etc., are good examples of relevant keywords. For example, rather than "Floppy breaks when using mailfile", a better subject would be "Loading mail file from floppy causes break in FLOPPY.OPEN with error ILLEGAL ARG: 42."

Assigned To: The name of the person or persons who took some action based on the AR.

Attn: The name of the person or persons responsible for fixing the AR.

Status: This field shows the status of the AR. This changes as action is taken on the AR.

<i>New</i>	All ARs are generated with a default Status of <i>New</i> when submitted. <i>New</i> ARs have not been reviewed.
<i>Open</i>	This reviewed AR describes an outstanding problem with released software.
<i>Open/Unreleased</i>	This reviewed AR describes a problem with unreleased software.
<i>Fixed</i>	Problem has been fixed in an Internal loadup. Developers marking ARs as <i>Fixed</i> should mark the In/By: field according to the release into which the fix is being incorporated. At this time, the developer should also fill in the Release Note: field.
<i>Closed</i>	System with fix in it has been tested, documented, & released.
<i>Declined</i>	ARs can be declined for any of a variety of reasons. Perhaps it's a request for feature that is officially "never" going to be implemented (e.g., we think it's a bad idea). Perhaps the bug report is considered spurious (development doesn't think it is a bug). The reason for the AR being declined should be included in the Description: field. <i>Declined</i> ARs will be reviewed periodically so that old ARs may be re-opened.
<i>Superseded</i>	Another AR already includes the problem described in this one. The In/By: field of the superseded AR should include the AR number of the one that supersedes it (ex., 7064), and the beginning of the Subject: field should be edited to include a notation such as: "Superseded by AR #7064". The superseding AR should contain the information contained in the AR it supersedes, with a notation in the Description: such as: "[Supersedes AR #7911.]".
<i>Obsolete</i>	The problem reported is no longer a problem, e.g. the module containing the reported problem is no longer supported.
<i>Incomplete</i>	The information submitted is not enough to take action, i.e., there is not enough information to identify the bug, or the feature request doesn't give enough detail about what is wanted. This is different from <i>Declined</i> in that the request is considered valid, but the AR remains open awaiting more detail.

<i>Internal</i>	This status is used to report problems with internal software.
<i>Wish</i>	This status is usually used to request new features, change of features, Design-Impl or Design-UI .
Problem Type:	Defines the type of problem described in the AR. Possibilities for the Problem Type: field follow:
<i>Bug</i>	The system does not work as documented.
<i>Design-Impl</i>	The system works, but the internal implementation is wrong. (This type is generally submitted by other developers.)
<i>Feature</i>	Used to indicate a feature request.
<i>Design-UI</i>	The design of the user interface is wrong. This includes problems in the way in which things display, as well as program callable structures.
<i>Documentation</i>	The system works, but the documentation is wrong, unclear, or incomplete. The System: and Subsystem: fields should reflect the area in which there is a problem with the documentation. The System: should <u>not</u> be <i>Documentation</i> unless there is a specific problem with the documentation, apart from the system, e.g. "need better index".
<i>Performance</i>	The system works, but it is too slow doing the described operation.
Difficulty:	A rough estimate of the difficulty of the problem. <i>This field is to be filled in by developer only.</i> Categories within Difficulty: follow:
<i>Easy</i>	< 1 week to fix
<i>Moderate</i>	< 1 month to fix
<i>Hard</i>	< 6 months to fix
<i>Very Hard</i>	> 6 months to fix
<i>Impossible</i>	can't be fixed
In/By:	Used to specify the release for which an AR is/will be fixed or to indicate the number of a superseding AR.
Impact:	How seriously does it affect your ability to get work done, value of Xerox Lisp, etc. The items apply to bug reports, but feature requests should be rated along analogous lines. The categories within Impact: follow:
<i>Fatal</i>	Causes the system to crash, causes a loss of work, etc. Problem resolution is a requirement for project completion.
<i>Serious</i>	The problem can be worked around but it seriously interferes with work. This type of problem usually requires substantial reimplementation.

<i>Moderate</i>	The problem is tolerable, but clearly a problem, and the responsibility of Interlisp development.
<i>Annoying</i>	The problem is annoying, a minor request for a new feature that "would be nice".
<i>Minor</i>	May be some dispute about whether it is even a bug, or a very minor feature request.
Frequency:	How reproducible is the problem? If it is not known or is irrelevant to the AR, leave it blank. This is generally only relevant for bug reports. Frequency: can be one of:
<i>Everytime</i>	Reproducible every time.
<i>Intermittent</i>	Doesn't always happen.
<i>Once</i>	Saw it happen once.
Priority:	The perceived priority of this problem relative to the next release. A submitter may fill in their desired priority when submitting the AR. <i>However, priorities are approved/changed only by the Change Control Board.</i> Four different priorities are possible:
<i>Absolutely</i>	A showstopper. The pending release will be held if this AR is not completed. Requirements for this rating are: 1) Work lost with no workaround; 2) Highly embarrassing to Xerox; or 3) Marked <i>Hopefully</i> for previous release.
<i>Hopefully</i>	Preferable to be in the pending release, otherwise will be in next release.
<i>Perhaps</i>	Will get implemented if other revisions in same area are completed.
<i>Unlikely</i>	Unlikely to be included in the next release.
System: Subsystem:	The category and sub-category of the Xerox Lisp system that is pertinent to this AR. System: and Subsystem: categories are:
<i>Communications</i>	NS Protocols NS Filing NS Printing PUP Protocols PUP FTP Grapevine Leaf RS232 VAX Server DEI EVMS/RPC Lisp Servers Clearinghouse TCP/IP Centronics TTYPort Chat

	Chat Interface Pup Chat Driver NS Chat Driver RS232 Chat Driver TTYPort Chat Driver Chat DM2500 Emulator Chat VT100 Emulator NSMaintain Other
<i>Windows and Graphics</i>	Window System Library Fonts Printing Color Bitmaps Demos Menus Other
<i>Operating System</i>	Virtual Memory Generic File Operations DLion Disk Daybreak Disk DLion Floppy Daybreak Floppy Dolphin/Dorado Disk Processes Streams Keyboard Mouse Other
<i>Language Support</i>	Arithmetic Compiler, Code Format For/If Microcode Storage Formats/Mgt Garbage Collection Read and Print Stack and Interpreter Bootstrapping and Teleraid Diagnostics Other
<i>Programming Environment</i>	Break Package Code Editor DWIM Inspector File Package History Masterscope PSW Record Package Performance Tools Edit Interface Exec Presentations

	Stepper Other
<i>Text</i>	TEdit TTYIN Lafite AR Database Other
<i>Common Lisp</i>	Type System Declarations Macros Control Structure Evaluator Symbols/Packages Arithmetic Characters/Strings Sequences Lists Arrays Structures Hash Tables Streams and I/O File System Interface Error System Compiler Tamarin Support Microcoded Operations Common Loops Other
<i>CLOS</i>	Language Browsers Methods Classes Meta Classes Other
<i>Port</i>	Other
<i>Maiko</i>	Bytecode Emulation Native Code I/O System Host Integration Host User Interface Foreign Fn Interface Installation Procedure Documentation Other
<i>LOOPS</i>	Active Values Composite Objects Objects Browsers User Interface Virtual Copy Other

<i>PCE</i>	Monochrome Display Color Display Keyboard Emulated Rigid Disk Floppy Disk Printer Port User Interface Programmatic Interface File System Interface Memory Ethernet Configuration Tools Other
<i>PROLOG</i>	Arithmetic Dinfo Microcode Editor Interface Compiler Interpreter I/O Debugging Prolog-Lisp Interface Other
<i>4045</i>	XLPStream Remoteserver HQStream PSO Other
<i>Rooms</i>	Window Types Overview Suites Buttons Documentation Other
<i>Library</i>	Cash-File CharCode Tables Copyfiles DEdit DatabaseFns FX-80 Printer Support Filebrowser Font Samples GCHax GraphZoom Grapher Hash Hash-File Image Object Interface Kermit Masterscope Browser MatMult Press Printer Support SameDir Sketch

	SysEdit/EXPORTS.ALL Tablebrowser Virtual Keyboards Where-Is Other
<i>BusMaster</i>	Speech Color Other
<i>Documentation</i>	Tools 1108 Users Guide 1186 Users Guide Primer Product Descr/Tech Summary Hardware Installation Guide Programmers Introduction Interlisp Reference Manual Library Package Manual Internal System Documentation Other
<i>Other Software</i>	Installation Utility Release Procedure Other
Machine: Disk:	The value of these fields should be the type of Xerox hardware that is pertinent to this AR, i.e., the machine and disk on which the problem is happening. Machine: and Disk: categories are:
<i>1108</i>	SA1000 (10 MB) SA4000 (29 MB) Q2040 (43 MB) Q2080 (80 MB) T80 (80 MB) T300 (300 MB) Other
<i>1132</i>	T80 (80 MB) Century315 Other
<i>1186</i>	ST212 (10 MB) TM703 (20 MB) TM702 (20 MB) ST4026 (20 MB) Q530 (20 MB) Q540 (40 MB) Micropolis 1303 (40 MB) Micropolis 1325 (80 MB)
Lisp Version:	This field should identify the Xerox Lisp sysout in which the problem occurs (or the feature doesn't occur). The sysout should be identified by the name associated with the release (Koto, Lyric, Medley, etc.,) and/or MAKESYSDATE.

Microcode Version:	This information may be found by typing (MICROCODEVERSION) in an Interlisp Exec or (il:microcodeversion) in a Common Lisp Exec.
File Server:	What type of file server, if any, is involved with this problem. The menu contains the following items: 8037 - IFS - NS - VAX/VMS - 3 MB - VAX/VMS - 10 MB - VAX/UNIX - Micro VAX/VMS - Other
Source Files:	The source files pertinent to the problem being reported in this AR.
Memory Size:	This value is the amount of "real memory", or RAM, in pages. This information may be found by typing (REALMEMORYSIZE) in an Interlisp Exec or (il:realmemorysize) in a Common Lisp Exec.
Server Software Version:	The version of software running on the server.
Disposition:	The record of who has changed which fields of this AR and when it was done. <u>This is filled in by the system.</u>
Release Note:	This field should contain the information to be included in the Release Notes for a given release. It should be release specific, such as: "Medley: In the debugger, the frame inspector window . . ." If a release note isn't required, that should also be explicitly mentioned, example: "Lyric LOOPS: None needed."
Description:	This field should contain a complete description of the problem or request, including any subsequent discussion after the AR submission. If the bug report came via electronic mail, the entire report should be added into this field. In cases where there are a number of electronic mail messages discussing this problem, all messages should be appended into this field.
Workaround:	This field should contain a known procedure to work around the problem until it is fixed. This would generally be a short recipe.
Test Case:	This field should contain a list of the files needed to recreate the problem. Please note that any Common Lisp or Interlisp recipes for reproducing the problem should be in the Description: field, not in the Test Case: field. When the problem is <i>Fixed</i> the Test Case: field should include any appropriate information that can be used to confirm the fix (or a note that a Test Case is not applicable, ex. "N/A").
Edit-By:	The login name of the last person to edit the AR. <u>This is filled in by the system.</u>
Edit-Date:	The date of the last change made to the AR. <u>This is filled in by the system.</u>

WHAT HAPPENS TO AN AR AFTER IT IS SUBMITTED?

Change Control Boards have been established for each XAIS product to bring AR priorities more in line with customer needs. The membership of each Change Control Board consists of the Product Development Project Leader, a member of Customer Support, and a member of Release Management. Incoming ARs are reviewed weekly by the appropriate board. At this meeting priorities are assigned for each AR, and other pertinent information, such as who will deal with the AR, is gathered. This information is input to the AR data base and summaries of ARs are generated for each responsible developer.

When a problem is resolved, the **Status:** field of the associated AR is changed to *Fixed*. At this point, the software is incorporated into the Development environment, which is the precursor for the next release. The *Fixed* AR is sent to the Documentation group for incorporation into the appropriate part(s) of the product documentation. When a release of software to customers occurs, all *Fixed* ARs that have been incorporated into that release, software and documentation, are marked *Closed*.

Array Space

Misc Notes

Date: 3 Apr 89 11:16
From: vanMelle:PA:Xerox
Subject: Re: Number of Locked pages?
In-Reply-to: Ingalls.wbst's message of 2 Apr 89 19:40 EDT
Reply-to: vanMelle:PA:Xerox
To: Ingalls:WBST:Xerox
cc: LispFolklore↑:X:RX

(\COUNTREALPAGES 'LOCKED) is what you want. Other interesting arguments to \COUNTREALPAGES include DIRTY, REF (pages that have been referenced since the last sweep of the real page table) and OCCUPIED (pages inhabited by Lisp vmem pages, as opposed to things like the Alto emulator (Dorado), the real page table and low-level system buffers, thus less than (REALMEMORYSIZE)).

Bill

Subject: A proposed design for big bitmaps (for Maiko Color (aka Kaleidoscope))
From: shih:mv:envos

Here's a proposed design for big bitmaps (for Maiko Color). This message is also filed as BigBitmaps.TEedit, under {Pogo:MV:envos}<DSUNLISP>Documents>Development>Color>, and {Eris}<lispcore>internal>doc>.

There are basically two alternatives, either change allocblock to allow blocks bigger than 65K, or to change bitblt (and perhaps many other functions) to allow a new datatype, BIGBM.

The following describes the second BIGBM alternative.

DESIGN MOTIVATION

The color code currently uses ALLOCPAGEBLOCK to create a non-GC'able large bitmap for the color screen. Windows on that screen, I believe point to that screen bitmap, but Window "backing bitmaps" (the thing that holds what is behind an open window, and what is inside a closed window), is a separate bitmap.

The problem is that the backing bitmap currently cannot be as big as the window would like it to be.

The BIGBM design would keep the noncollectible color screen bitmap, but would allow windows to have BIGBM backing bitmaps. In addition, this design is more likely to be portable backwards (e.g. into Medley1.0 or Medley1.1) since no existing system datatypes need to be changed (unlike the plan to change ALLOCBLOCK).

Since XAIE does not (I believe) allow (currently) DIG operations on closed windows, then there are no DIG operations (except BITBLT) which need to occur on BIGBM backing bitmaps.

Therefore, a BIGBM need merely be a GC'able datatype which supports BITBLT between itself and all other legal datatypes (principally windows and bitmaps, but possible streams (e.g. Interpress), and possible other datatypes).

Longer term, having a generalized BITBLT will be useful, for the following reasons:

Color - 8 bpp bitmaps will begin to push Medley's 32Mb address limitation, and 24 bpp bitmaps certainly will (the screen alone will take up 3Mb = 10% of the address space!).

Remote Bitmaps - there may be applications which need bitmaps outside of the address space (color above for example). Nick Brigg's Maple color processor for the 1186 had a generalized remote bitblt, which "did the right thing" when source and destination bitmaps were both remote (e.g. remote bitblt).

XWindow Medley- might require remote bitmaps.

NonSun Medley - might require remote bitmaps, especially because it is doubtful the SunOS system call MMAP exists elsewhere. When not, the screen itself will need to be remote.

BIGBM DESIGN

A straightforward design would be to have a BIGBM simply be a collection of ordinary bitmaps ("slices of the big bitmap"), and then generalize BITBLT to handle the collection (by repeated bitblts).

More elaborate designs would allow BIGBMs to be a collection of any BitBlt'able objects (streams, windows, bitmaps, bigbitmaps), or perhaps collections of raw blocks (to save some of the BITBLT initialization overhead).

Probably at least one dimension of the slices should be a multiple of the LCD (least common denominator) of any TEXTURE (currently 16), so that textures do not show seams across the slices.

Probably also the slices should be "short and fat" (e.g. the BIGBM is made up of several rows of bitmaps), since the ucode inner loop runs in the X direction, but this partially depends on the application.

For example, font bitmaps are short and *very* wide. If large color fonts need a BIGBM font bitmap, then each character would cut across several slices, slowing down each character. For fonts, column slices might be better. Note that each character is also more "coherent" in memory this way, which might improve paging behaviour.

Having BIGBMs be composite objects (rather than a large coherent allocblock) might also improve GC behaviour, because a BIGBM can be allocated out fragmented free space. There may not be enough coherent free space to allocate a large allocblock, due to fragmentation.

Another design elaboration would be to provide full Imagestream capabilities onto BIGBMs, to that (DSPCREATE dest) will work on BIGBMs.

BIGBM IMPLEMENTATION

There is currently a draft implementation of BIGBMs on {Eris}<lispcore>internal>library>BIGBM. It has several limitations:

0. It is intended to be a subfunction of BITBLT (which explains the limits 1 & 2 below)
1. It doesn't handle clipping (I'm not sure, but clipping might be handled generically in BITBLT by changing the srcex, srcey, destx, desty, width, & height args).
2. It doesn't handle default arguments (BitBlt does this).
3. It currently only makes the slices 16 bits tall (it should make the slices as big as possible).
4. Its not very efficient. It only walks the slices linearly (the first relevent slice could be found by algebra), and it doesn't stop when the last relevent slice is bltted.
5. Because of 3 & 4, for large bitmaps its roughly 2x slower than normal bitmap blts. For small bitmaps, it may be much worse. This may be OK though for backing color windows, because color windows are large.

Mostly, this code needs to be folded into BITBLT & BLTSHADE, at the appropriate spots, and optimized if necessary.

Abstract: This document is a general introduction to the Interlisp-D computing environment, slanted towards the needs and interests of newcomers to the Intelligent Systems Laboratory and Xerox Artificial Intelligence Systems. This is definitely in *DRAFT* form, since it still talks a lot about Cedar and hardly any about Interlisp-D!!!!

For Internal Use Only

Raison d'Etre

The purpose of this document is to help immigrants adapt to the local computing community. "The local community" primarily means Interlisp-D users within PARC's Intelligent Systems Laboratory and the development group of Xerox Artificial Intelligence Systems. Immigrants to other computing communities within Xerox may also find this document of interest no guarantees are made. I shall assume herein that said immigrants know quite a bit about computers in general. Hence, I shall concentrate upon discussing the idiosyncratic characteristics of the local hardware environment, software environment, social environment, linguistic environment, and the like. This document was "ripped off" from a similar one written for the Computer Sciences Laboratory of PARC, whose members primarily use another environment -- Cedar, in the great PARC tradition of developing many different programming environments.

There is a great deal of useful information available on-line at Xerox in the form of documents and source programs. Reading them is often very helpful, but finding them can be a nuisance. Throughout this document, references to on-line material are indicated by <reference>, where n is a citation reference in the bibliography at the end of this document. Standard citations to the open literature appear as [reference].

Reading a document from front to back can be mighty boring. Fortunately, this document is so disorganized that it is not at all clear that it really has a front and a back in any normal sense. You might as well just browse through and read the parts that look interesting. To help out the browsers in my reading community, I have more or less abandoned the custom of being careful to define my terms before I use them. Instead, all the relevant terms, acronyms, and the like have been collected in a separate Glossary. Some information is contained only in the Glossary, so you may want to skim through it later (or now, for that matter). While writing the Glossary, I assumed that you have a basic knowledge of computer science, and a modicum of common sense: don't expect to find terms like "computer" and "network" in the Glossary.

Naming Things

At the outset, you should know something about the names of the creatures that you will find here. The prevailing local philosophy about naming systems is perhaps somewhat different from the trend elsewhere. We do have our share of alphabet soup, that is, systems and languages that are named by acronyms of varying degrees of cuteness and artificiality; consider, for example: PARC, FTP, IFS. But we are trying to avoid making this situation any worse. To this worthy end, names for hardware and software systems are frequently taken from the Harvard Concise Dictionary of Music, or the Sunset Western Garden Book [sunset]; Grapevine servers are named after wines; Dorados are named after hotels, spices, philosophers or ships (depending on who owns them); XDE releases are named after California rivers. This convention about names does not meet with universal approval.

Local Hardware

Most of the offices and some of the alcoves around have personal computers in them of one flavor or another. The first of these was the Alto. There are more than a thousand Altos in existence now, spread throughout Xerox, the four universities in the "old" University Grant program (U. of Rochester, CMU, MIT, and Stanford), and other places. (There's a "new" University Grant program in progress.) In recent years, most of the local Altos have been replaced by various flavors of D-machines: Dorados, Dolphins, and Dandelions. Both D-machines and Altos come equipped with bitmap displays, mice, and Ethernet interfaces. Let's discuss these components first, and then turn our attention to the various personal computers that contain them.

Bitmap Displays

First, let's talk about displays. Different displays use different representations of images. A character display represents its image as a sequence of character codes. This is a very compact representation, but not a very flexible one; text is all you can get, and probably in only a limited selection of fonts. A vector display represents its image as a list of vector coordinates. This works very well for certain varieties of line drawings, but not so well for filled areas or text. A bitmap display, on the other hand, produces an image by taking a large matrix of zeros and ones, and putting white where the zeros are and black where the ones are (or vice versa). The great advantage of bitmap displays are their flexibility: you can specify a tremendous number of images by giving even a relatively small array of bits. Cursors and icons are two large classes of prominent examples. Of course, you do have to supply enough memory to hold all those bits. Altos and D-machines store their bitmaps in main storage. An alternative would be to provide a special chunk of memory on the side where the display's image sits; such a memory is often called a frame buffer.

The primary display of the Alto is a bitmap that is 608 pixels wide by 808 pixels high. Such a display is almost

large enough to do a reasonable job of rendering a single 8.5" by 11" page of text. The CRT on a D-machine has the long axis horizontal instead of vertical, giving a bitmap display that is 1024 pixels wide by 808 high. It had to be 808 high so that D-machines could emulate Altos, of course. The extra space allows you to have something else on the screen as well as the somewhat scrunched page of text that you are editing.

Ere I leave you with a mistaken impression, let me note in passing that bitmap displays are not the final solution to all of the world's problems. Raster displays that can produce various levels of gray as well as black and white can depict images free of the "jaggies" and other artifacts that are inherent in bitmap displays [grayscale]. And, for some purposes, color is well worth its substantial expense.

Mice

But now on to mice. A mouse has two obvious properties--it rolls and it clicks. Inside the machine, the mouse position and the display cursor position are completely unrelated; but most software arranges for the cursor to "track" the mouse's movements. Some mice have two buttons, others three. The three mouse buttons go by various names; "left", "middle", and "right" is one set of names. The mouse buttons have at some point also been called "red", "yellow", and "blue" respectively, even though physically they are nearly always black. These colorful names were proposed at an earlier time when some of the mice had their buttons running horizontally instead of vertically. Using colors (even imaginary ones!) worked better than switching back and forth between the nomenclatures "top-middle-bottom" and "left-middle-right".

Mice also come in two basic flavors: mechanical and optical. Our current mechanical mice roll on three balls: two small ones, and one large one. Motion of the large ball is sensed by two little wipers inside the mouse, one sensing side to side rolling while the other senses forward and backward rolling. The motion of each wiper drives a commutator, and little feelers slide along the commutator, producing the electrical signals that the listening computer can decode. Building one of these little gadgets is not quite as hard as building a Swiss watch, but it's in the same league. The optical mice are a more recent innovation. An optical mouse lives on a special pad, covered with little white dots on a black background. A lens in the mouse images a portion of the pad onto the surface of a custom integrated circuit. This IC has sixteen light-sensitive regions, some of which notice that they are being shined on by the image of a white dot on the pad. As the mouse slides along the pad on its Teflon-coated underbelly, the images of the white dots move across the IC; it is subtly constructed so as to observe this phenomenon, and take appropriate electrical action. For more details on this interesting application of a custom chip, you might enjoy checking out the blue-and-white report on the subject [opticalmouse].

The Ethernet

Two's company, three's a network. A collection of machines within reasonable proximity is hooked together by an Ethernet; if that doesn't sound familiar, I know of some blue-and-whites that you might like to browse [ethernet]. Ethernets are connected to each other by Gateways and phone lines, which for most purposes allow us to ignore the topology of the resulting network. The resulting network as a whole is called an Internet. Occasionally, it's nice to know where things really are, and that's when a map <netmap> is helpful.

Ethernets come in two flavors: old and new. The old one runs at 3 MBits/sec, and should now be referred to as the "Experimental Ethernet". The unqualified name "Ethernet" should be reserved for the new one, the standardized version used in OSD products; it runs at 10 MBits/sec.

We all know how uncommunicative computers can be when left to their own devices. That's why we invent careful protocols for them to use in talking to each other. There are two entire worlds of protocols that are spoken on our various Ethernets as well: old and new. The old ones are called PUP-based (PARC Universal Packet) [PUP]. The new ones are known by the acronym NS (Network Systems) [NS]. I'm sure that the NS protocols must be documented, but I don't know where; sorry. Each protocol world includes a hierarchy of protocols for various purposes such as transporting files, or sending and receiving mail.

In addition to connecting up all of the personal computers, the network also includes a number of machines generically called servers. Normally, servers have special purpose, expensive hardware attached to them, such as large-capacity disks, or printers. Their purpose in life is to make that hardware available to the local community. We tend to identify servers by function, so we talk about print servers, file servers, name lookup servers, mailbox servers, tape servers, and so on. Many of the protocols for use of the Ethernet were developed precisely so that personal computers could communicate effectively with servers.

The Alto

The innards of the Alto are wonderfully described in a clear and informative blue-and-white report [ALTO]. For our purposes, suffice it to say that the Alto is a 16-bit minicomputer whose primary claim to fame is that it comes equipped with a bitmap display, a mouse, and an Ethernet interface.

D-Machines

The D-machines are a family of personal computers, each member of which has a name starting with the letter "D". As long as you don't look too closely, D-machines look a lot alike. In particular, they are all 16-bit computers with a microprogrammed processor that handles most of the I/O as well running the user's programs. And they all generally come equipped with a hard disk, a bitmap display, a keyboard, a mouse, and an Ethernet interface. There are differences of course: in size, in speed, and in flexibility.

The Dolphin (formerly called the D0)

The Dolphin was one of the early D-machines, and there are still some of them around. Dolphins are housed in the same sized chassis as Altos. You can tell that they aren't Altos because they have wide screen terminals, and because they don't have a slot on top for a removable disk pack. Instead, they use a 28MByte Winchester disk drive made by Shugart. Dolphins can talk to both 3 MBit and 10 MBit Ethernets. It was sold by Xerox AI systems as the Xerox 1100 with Interlisp-D and Smalltalk.

The Dandelion

The Dandelion is the D-machine processor that is used in the Star products and the Xerox 1108. It comes in a box about half the width of an Alto chassis, and roughly the same height and depth. Dandelions are both faster and much cheaper than Dolphins. Dandelions talk only to 10 MBit Ethernets. Recently a number of hardware enhancements have been developed for Dandelions, including the ability to add a lot more memory (up to 4MByte) and hardware floating point, IBM PC Bus controller, etc.

The Dorado

The Dorado was (and probably still is) the most powerful personal computer around. The Computer Science Lab built the Dorado as the current high-performance model in the D-machine line. The processor, the instruction fetch unit, and the memory system of the Dorado have been written up in papers for your enjoyment [DORADO]. Dorados come equipped with a 315 MByte Winchester drive; older models have an 80 MByte removable-pack disk drive at present. Dorados talk only to 3 MBit Ethernets at present.

A Dorado is roughly three to five times faster than an Alto when emulating an Alto, that is, running BCPL. Dorados are generally 3-5 times faster than DLions running Interlisp, except for some notable exceptions (Dorados have an incredible memory bandwidth, and so can move the bits around the screen **really** fast; the DLions have special hardware for floating point which makes them faster than the (microcoded) Dorados.). One primary difficulty about Dorados is that they're a lot more expensive than DLions, and don't fit in your office. As a result, while some people have personal Dorados, there are also **pool** machines. When you borrow a Dorado, you generally also want to borrow at least some of the space on that Dorado's local disk. In order for this sharing to work out well, certain social taboos and customs concerning the use of such local disks have emerged, under the general rubric of "living cleanly". More on this topic anon.

In a return to the ways of the past, the Dorado processors are rack mounted in a remote, heavily air-conditioned machine room. It was initially intended that the Dorado, like the Alto, would live in your office. To prevent its noise output from driving you crazy, a very massive case was designed, complete with many pounds of sound-deadening material. (This case was known as the APC, or Armored Personnel Carrier, which it resembled.). But experience indicated that Dorados ran too hot when inside of these cabinets, and the concept of having Dorado processors in offices was abandoned. With progress in general and VLSI in particular, there is hope that the **next** high-performance contender will come out again and live in your office.

The Dicentra

The Dicentra is another D-machine of which there aren't a lot around. Essentially, it consists of the processor of the Dandelion with the tasking stuff striped out squeezed onto one Multibus card. It communicates with its memory and with I/O devices over the Multibus. Dicentras will talk to any Ethernet, or any I/O device for that matter, for which you can supply a Multibus interface card; that's one of the Dicentra's strengths. The initial application of the Dicentra is as a processor for low cost Internet gateways. The Dicentra and the Dandelion are named after wildflowers partially because they are outgrowths of an initial design called the Wildflower.

The Dragon

The Dragon is a high-performance processor based on custom integrated circuits that is being designed in CSL; confusingly enough, though, the Dragon is not really a D-machine. For example, the Dragon word size is 32 bits rather than 16. The underpinnings of Cedar will be adjusted as necessary so that Cedar will run on a Dragon; but this will take some doing.

Other D-machines

There may well be other D machines brewing in laboratories and development groups. This document won't talk about them, as they are more closely tied with some product plans that should not be widely circulated.

A few comments about Booting

All of the local processors come equipped with a hidden button called the "boot button" that is used to reinitialize the processor's state. The Alto had just one boot button, hidden behind the keyboard; pushing it booted the Alto. On Dolphins, the situation is only slightly more complex: there are two boot buttons, one at the back of the keyboard, and the other on the processor chassis itself. They perform roughly the same function, but the one on the chassis is a little more potent. On Dorados, there is a lot more going on. There are really two computers involved,

the main Dorado processor and a separate microcomputer called the baseboard. It is the baseboard computer's job to monitor the power supplies and temperature and to stage-manage the complex process of powering up and down the main processor, including the correct initialization of all of its RAM's. The boot button on a Dorado is actually a way of communicating with this baseboard computer. You encode your request to the baseboard computer by pushing the boot button repeatedly: each number of pushes means something different. For details, see Ed Taft's memo on the subject <DORADOBOOT>. If the baseboard computer of the Dorado has gone west for some reason (as occasionally happens), your only hope is to push the real boot button, a little white button located on the processor chassis itself, far, far away. Just as the boot button on the keyboard is essentially a one-bit input device for the baseboard computer, the baseboard computer also has a one-bit output device: a green light located on the processor chassis. Various patterns of flashing of this light mean various things, as detailed in <DORADOBOOT>.

DLions have a boot button located right under the Maintenance Panel. It's labeled B-reset. There's another button next to it, labelled Alt-B, which is used in conjunction with B-reset to boot the machine in various ways -- you hold down both, let up on B-reset, and the maintenance panel lights cycle with numbers 0000, 0001, 0002, 0003, etc. When you let up, you get the "boot option" that you let up on: 0000 (or 0-boot) usually boots Lisp from the disk (or just B-reset), 1-boot boots Mesa from the disk, 2-boot boots from floppy, 3-boot boots from ethernet, 4-boot boots something that I forget, 5-boot gets "diagnostics" from a floppy, 6-boot gets an alternate Ethernet Mesa, and a few others. (0010-boot with a floppy-cleaning-diskette in the floppy drive will clean the floppy heads. This is useful you have a broken machine with a broken ethernet, and you want to floppy boot diagnostics but can't because of dirty heads!)

There is one more bit of folklore about booting Dorados that's fun to mention--every once in a while, I have to throw in some subtle tidbit to keep the wizards who read this from getting bored. Our subject this time is the "long push boot". Suppose that you have been working on a Dorado for a while, and you walk away to go to the bathroom. When you return and reach toward your keyboard, you get a static shock. You are only mildly annoyed at this until you notice that the cursor is no longer tracking the mouse, and the machine doesn't seem to hear any of your keystrokes. The screen looks OK, but the Dorado is ignoring all input. What has probably happened is that the microprocessor in your terminal has been knocked out by the static shock. Yes, Virginia! In addition to the Dorado itself, and the baseboard computer, there is also a microprocessor in your terminal (located in the display housing), which observes your input actions and sends them on to the main processor under a protocol referred to as "the seven-wire interface". What you want to do now is to reboot the terminal microprocessor without disturbing the state of the Dorado at all--after all, you were in the process of editing something, and you are now in danger of losing those edits. What you should do is to depress the boot button and hold it down for quite a while (more than 2.5 seconds); and then release it. This is known as a "long push boot", and it does just what you want under these conditions: it reboots your terminal without affecting anything higher up.

Local Programming Environments

Various programming environments have grown up around the various pieces of hardware mentioned above. You can get a software merit badge simply by writing one non-trivial program in each environment.

Mesa

Mesa is a strongly typed, PASCAL-like implementation language designed and built locally. It first ran on Altos. Herein, I shall call that system Alto/Mesa. Dolphins and Dorados (but not Dandelions) can run Alto/Mesa by impersonating an Alto at some level. More recent instances of Mesa now run on all of our D-machines under the Pilot operating system. In passing, I should observe that Pilot is an operating system written in Mesa by folk in SDD. It is a heavier-weight operating system than the Alto OS, providing its clients with multiprocessing, virtual memory, and mapped files.

The Pilot version of Mesa is the home to lots of active programming in several locations. First, it is the system in which the Star product was and is being implemented by OSD. The programmers in OSD have developed a set of tools for programming in Mesa variously called the "Tools Environment" or "Tajo" or "XDE" or "Basic Workstation". This body of software may soon be marketed under the name "the Xerox Development Environment". In addition, Pilot Mesa is the current base of the Cedar project in CSL. More on Cedar later. Although Mesa programs look a lot like PASCAL programs when viewed in the small, Mesa although Mesa provides and enforces a modularization concept that allows large programs to be built up out of smaller pieces. The Mesa language is described by a manual [MESA].

Smalltalk

Smalltalk was developed by the folk who now call themselves the Software Concepts Laboratory (formerly known as the Learning Research Group and then the Software Concepts Group). The Smalltalk language is the purest local embodiment of "object-oriented" programming:

A computing world is composed of "objects".

The only way to manipulate an object is to be polite, and ask it to manipulate itself. One asks by sending the object a message. All computing gets done by objects sending messages to other objects.

Every object is an "instance" of some "class".

The class definition specifies the behavior of all of its instances--that is, it specifies their behavior in response to the receipt of various messages.

Genealogists will recognize that ideas from both Simula and Lisp made their way into Smalltalk, together with traces of many other languages.

For some years now, the folk in SCG have been working at trying to get the Smalltalk language and system out into the great wide world. The first public event that came out of this effort was the August 1981 issue of Byte magazine; it was devoted to Smalltalk-80, including a colorful cover drawing of the now famous Smalltalk balloon. In addition, the SCG folk are writing several books about Smalltalk, and they are planning to license the system itself to various outside vendors. The first of the books, entitled Smalltalk-80: The Language and Its Implementation, emerged from the presses at Addison-Wesley just recently [21]. Future books will include Smalltalk-80: The Interactive Programming Environment, and Smalltalk-80: Bits of History, Words of Advice.

Interlisp-D

LISP is the standard language of the Artificial Intelligence community. Pure LISP is basically a computational incarnation of the lambda calculus; but the LISP dialects in common use are richer and bigger languages than pure LISP. Interlisp is one dialect of LISP, an outgrowth of an earlier language called BBN-LISP; for more historical details, read the first few pages of the Interlisp Reference Manual [22]. One of the biggest strengths of Interlisp is the large body of software that has developed to assist people programming in Interlisp. Consider the many features of Interlisp: an interpreter, a compatible compiler, sophisticated debugging facilities, a structure-based editor, a DWIM (Do What I Mean) error correction facility, a programmer's assistant, the CLISP package for Algol-like syntax, the Masterscope static program analysis database, and the Transor LISP-to-LISP translator, to name a few.

Interlisp itself has been implemented several times. Interlisp-10 is the widely-used version that runs on PDP-10's. Interlisp-D is an implementation of Interlisp on the D-machines [23], produced by folk at PARC. In the process of building Interlisp-D, the boundary between Interlisp and the underlying virtual machine was moved downward somewhat, to minimize the dependencies of Interlisp on its software environment; that is, functions that were considered primitive in Interlisp-10 were implemented in Lisp itself in Interlisp-D. But the principal innovations of Interlisp-D are the extensions that give the Interlisp user access to the personal machine computing environment: network facilities and high-level graphics facilities (including a window package) among them.

[MORE]

Cedar

Back in 1978, folk in CSL began to consider the question of what programming environment we would use on the emerging D-machines. A working group was formed to consider the programming environments that then existed (Lisp, Mesa, and Smalltalk) and to form a catalog of programming environment capabilities, ranked by both by value and by cost. A somewhat cleaned-up version of the report of that working group is available as a blue-and-white for your perusal [EPE]. After pondering the alternatives for a while, CSL chose to build yet another programming environment, based on the Mesa language. That new environment was named "Cedar". The programming language underlying Cedar is essentially Mesa with garbage collection added. Now, adding garbage collection actually changed things quite a bit. First of all, it changes programming style in large systems tremendously. Without garbage collection, you have to enforce some set of conventions about who owns the storage. When I call you and pass you a string argument, we must agree whether I am just letting you look at my string, or I am actually turning over ownership of the string to you. If we don't see eye to eye on this point, either we will end up both owning the string (and you will aggravate me by changing my string!) or else neither of us will own it (and its storage will never be reclaimed--a storage leak). Once garbage collection is available, most of these problems go away: God, in the person of the garbage collector, owns all of the storage; it gets reclaimed when it is no longer needed, and not before. But there is a price to be paid for this convenience. The garbage collector takes time to do its work. In addition, all programmers must follow certain rules about using pointers so as not to confuse the garbage collector about what is garbage and what is not.

Thus, programs in the programming language underlying Cedar look a lot like Mesa programs, but they aren't really Mesa programs at all, on a deeper level. To avoid confusion, we decided to use the name "Cedar" to describe the Cedar programming language, as well as the environment built on top of it. Cedar is really two programming languages: a restricted subset called the safe language, and the unrestricted full language. Programmers who stick to the safe language can rest secure in the confidence that nothing that they can write could possibly confuse the garbage collector. Their bugs will not risk bringing down the entire environment around them in a rubble of bits. Those who choose to veer outside of the safe language had better know what they are doing.

Those who want to know more about Cedar are once again encouraged to dredge up a copy of the Cedar Manual <25>. It includes documentation on how Cedar differs from Mesa, annotated examples of Cedar programs, manuals for many of Cedar's component parts, a Cedar catalog, and lots of other good stuff. By the way, the most authoritative source for what the current Cedar compiler will do on funny inputs can be found in a document called the Cedar Language Reference Manual, also known by the acronym CLRM. This is logically part of the Cedar Manual, but it is currently bound separately, and only available in draft form. The CLRM suggests a particular design philosophy for building a polymorphic language that is a superset of the current Cedar, since that is the direction in which the authors of the CLRM, Butler Lampson and Ed Satterthwaite, would like to nudge the Cedar language.

Local Software

This section is a once-over-lightly introduction to some of the major software systems that are available in the Interlisp world.

In Interlisp, the current best sources are the Interlisp Manual mentioned above.

Filing

When programming in the Alto world, or in current Cedar, you are dealing with two different types of file systems: local and remote. The local file system sits on your machine's hard disk. Remote file systems are located on file servers, machines with big disks that are willing to store files for you. Local file systems have several unpleasant characteristics in comparison with the remote systems: they are small, and they aren't very reliable. Both of these problems have consequences.

Some people believed that, because local file systems are small, it wasn't in general practical to store more than one version of a file on the local disk. Thus, some local file systems don't support versions, and in programs that use them, writing a "new version" of a file really means writing on top of the old one. Nearly everyone who isn't accustomed to this (particularly Interlisp programmers) gets burned by it at least once. Some text editors in XDE and the Alto world *do* maintain one backup copy of each file being edited as a separate file, whose name ends with a dollar sign. That is, the backup copy of "foo.bravo" is stored in the file "foo.bravo\$". Note that our remote file servers do maintain multiple versions of files. Letting old versions of things accumulate is one easy way to overflow your disk usage allocation on a remote server.

No disk is completely reliable. Our remote file servers have automatic backup facilities that protect us from catastrophic disk failures. But the local file systems have no such automatic protection. Since this protection isn't provided automatically, it behooves you to adjust your behavior appropriately: make sure that, on a regular basis, backup copies of the information on your local disk are put in some safe place, such as on a remote file server where suitable precautions are constantly being taken by wizards to protect against disk failure. Doing this is one facet of what is meant by the phrase Living Cleanly, which deserves its own section.

Living Cleanly

The phrases "living cleanly" refer to a particular style of use of your local file system. In order to understand the cosmic issues involved, we should pause to discuss the ways in which local and remote file systems have been used over the years.

Back in the Alto days, personal files were usually stored on one's Alto disk pack, while project-related and other public files were stored on remote servers. Careful folk would occasionally store backup copies of their personal files on remote servers as well, in case of a head crash. But, as a general rule, one thought of one's Alto pack as the repository of one's electronic state. This made sharing Altos quite convenient, since you could turn any physical Alto into "your Alto" just by spinning up your disk pack.

In the glorious world of the future, all of your personal files as well as all public files will live on file servers in the network. The disk attached to your personal computer will, from time to time, contain copies of some of this network information, for performance reasons; but you won't have to do anything to achieve this, and you won't have to worry about how it is done. From the user's point of view, all files will act as if they were remote at all times. Indeed, except in a few funny cases, there won't even be any notion of "local file"; "file" will mean "remote file".

At the moment, we are sitting in an unpleasant transitional phase somewhere between these two styles of usage of the local disk: we are attempting to simulate the latter state by means of manual methods and social pressure. We want you to think of your data as really living out on the file servers. That is the proper permanent home for your personal files as well as for public files. You will have to bring copies of these files, both private and public, to your local disk in order to work on them. But, at the end of each editing session, you should store the new versions of files that you have created back out to their permanent remote homes. None of this happens automatically at present; you have to make it happen manually by using various file shuffling tools, such as the "DF files" discussed

below. Using these tools is a hassle, and learning how to use them can be confusing. But, there are four important benefits to be reaped from adopting a clean living life-style.

First, you are taking a step towards the glorious future.

Secondly, you are protecting yourself against failures of the local disk. A clean liver only holds information on her local disk for the duration of an editing session. This puts a reasonable bound on the amount of information that she can lose because of a disk crash.

Local file systems

The local file system in the Alto world is called either the "Alto file system" or the "BFS", the latter being an acronym for Basic File System. The biggest that a BFS can be is 22,736 pages. This is substantially bigger than the entire disk on an Alto. However, Dolphins and Dorados have much bigger local disks. Hence, when a Dolphin or Dorado is emulating an Alto, its local disk is split up into separate worlds called partitions, each containing a maximum-sized BFS. Dolphin disks can hold two full partitions, while Dorado disks can hold nineteen. What partition you are currently accessing is determined by the contents of some registers that the disk microcode uses. There is a command called "partition" in the Executive and the NetExec that allows you to change the current partition -- it necessarily "boots" the machine.

When operating in the Pilot world, a disk pack is called a physical volume, and it is divided into worlds called logical volumes.

All of our local file systems use a representation for files that drastically reduces the possibility of a hardware or software error destroying the disk's contents. The basic idea is that you must tell the disk not only the address of the sector you want to read or write, but also what you think that sector holds. This is implemented by dividing every sector into 3 parts: a header, a label, and a data field. Each field may be independently read, written, or compared with memory during a single pass over the sector. The Alto file system stuffs a unique identification of the disk block, consisting of a file serial number and the page number within the file, into the label field. Now, when the software goes to write a sector, it typically asks the hardware to compare the label contents against data in memory, and to abort the writing of the data field if the compare fails. This makes it pretty difficult, though not impossible, to write in the wrong place. Furthermore, it distributes the structural information needed to reconstruct the file system over the whole disk, instead of localizing it in one place, the directory data structures, where a local disaster might wipe it out. Each local file system also has a utility program called a Scavenger that rebuilds the directory information by looking at all of the disk labels.

Remote file systems

The most important local file servers are IFS's, an acronym for Interim File System (one of the crown jewels of the BCPL programming environment). Like I always say, "temporary" means "until it breaks", and "permanent" means "until we change our minds". Indigo and Ivy are two prominent local IFS's; Indigo stores mostly project files, while Ivy stores mostly personal files. MAXC also serves as a file server for some specialized applications. Juniper was CSL's first attempt to build a distributed transactional file server; it was one of the first large programs written in Mesa. Alpine is a new effort to build such a beast in the context of Cedar, in support of distributed databases and other such wonderful things. Some Walnut users have been storing their mail databases on Alpine for a month or more.

There is no coherent logic to the placement of "general interest" files and directories, nor even to the division between Maxc, Indigo, and Ivy. Browse through the glossary at the end of this document to get a rough idea of what's around. If something was made available to the universities in the University Grant program, then it is probably on Maxc (or archived off of Maxc), since Maxc is the machine that the university folk can access.

IFS supplies a general sub-directory structure which the Maxc file system lacks, and as a result there are lots of place to look for a file on an IFS. For example, on Maxc you might look for

```
[Maxc]<AltoDocs>MyFavoritePackage.press
```

while on IFS you would probably look for

```
[Indigo]<Packages>Doc>MyFavoritePackage.press, or
```

```
[Indigo]<Packages>MyFavoritePackage>Documentation.press,
```

or perhaps some other permutation. This requires a bit of creativity and a little practice. However, if you get in the habit of using "*"s in file name specifications, you will find all sorts of things you might not otherwise locate. Note that a "*" in a request to an IFS will expand into all possible sequences of characters, including right angle brackets and periods. Thus, for example, a request for

```
<Packages>*press
```

refers to all files on all subdirectories of the Packages directory that end with the characters "press". A "*" won't match a left angle bracket, by the way. Thus, if you ask for "*.press", you are referring to all Press files on the current directory. If you ask for "<*.press", you are referring to all of the Press files on the entire IFS (expect such a search to take a long time!).

There is a movement afoot in the Cedar world to simplify our file naming conventions by replacing the various flavors of brackets with a UNIX-like slash. Thus, in some Cedar systems, such as the FileTool, the documentation file mentioned above could be referred to as

/Indigo/Packages/MyFavoritePackage/Documentation.press.

File Properties

The "size" of a file is its length measured in disk pages; the "length" of a file is its length measured in bytes. The "create date" of a file is the date and time at which the information in that particular version of the file was "created", that is, the date when this that sequence of bytes came into being. Copying a file from one file system to another does not change the create date, since the information in the file, the sequence of bytes, is not affected. The create date is almost always what you want to know about a file. Some of our systems also maintain a "write date" or a "read date", but they are less well defined, and not as interesting.

Editing and Typesetting

In the outside world, document production systems are usually de-coupled from text editors. One normally takes the text that one wants to include in a document, wraps it in mysterious commands understood by a document processor, feeds it to that processor, and puzzles over the resulting jumble of characters on the page. In short, one programs in the document processor's language using conventional programming tools--an editor, a compiler, and sometimes even a debugger. Programmers tend to think this is neat; after all, one can do anything with a sufficiently powerful programming language. (Remember, Turing machines supply a sufficiently powerful programming language too.) However, document processors of this sort frequently define bizarre and semantically complex languages, and one soon discovers that all of the time goes into the edit/compile/debug cycle, not careful prose composition.

Bravo is the editor and typesetter in the Alto world, and it represented a modest step away from the programming paradigm for document production. A single program provided both the usual editing functions and a reasonable collection of formatting tools. You can't program Bravo as you would a document "compiler", but you can get very tolerable results in far less time. The secret is in the philosophy: what you see on the screen is what you get on paper. You use the editing and formatting commands to produce on the screen the page layout you want. Then, you tell Bravo to ship it to a print server and presto! You have a hardcopy version of what you saw on the screen. Sounds simple, right?

Of course, it isn't quite that easy in practice. There are dozens of subtle points having to do with fonts, margins, tabs, headings, and on and on. Bravo was a success because most of these issues are resolved more or less by fiat--someone prepared a collection of configuration parameters and a set of forms that accommodated most document production. Many of the configuration options aren't even documented, so it is hard to get enough rope to hang yourself. The net effect is that one spent more time composing and less time compiling.

In Bravo's wake, several new editors of unformatted text appeared: the Laurel editor, and the editor in the Tools Environment are prominent examples. The Laurel editor is particularly noteworthy in that it pioneered the development of a modeless (or at least less modal) user interface for an editor. The Star product editor, Tedit and Tioga are more recent local editors in the full Bravo tradition: they can handle formatting and multiple fonts. Tioga is the editor within Cedar, and its user interface is very close to the widely beloved Laurel modeless interface--try going back to Bravo after using Tioga for a while, and see how horrible it feels to have to remember to type "i" and "ESC" all the time. Tioga shows formatted text on the screen. To get a hardcopy of that text, the current path involves running a companion program called the TSetter, which will compose your pages for printing and send them to a print server. Tioga's documentation is particularly convenient, since it is usually available in iconic form at the bottom of the Cedar screen <29>.

Dealing with editor bugs

All text editors have bugs. Furthermore, you are often most likely to tickle one of the remaining bugs in an editor when you are working furiously on a hard problem, and hence, have been editing for a long time without saving the intermediate results. As fate would have it, these are exactly the times when it is most damaging and most upsetting to lose your work. There is nothing quite like the sinking feeling you get when a large number of your precious keystrokes gurgle away down the drain. Both Bravo and Tioga have mechanisms that can, in some cases, save you from the horrible fate of having to do all those hours of editing over again. Bravo attempts to safeguard you by keeping track of everything that you have done during the editing session in a log file; in case of disaster, this log can be replayed to recapture most of the effects of the session. If you have a disaster when editing in Bravo, be careful NOT to respond by running Bravo again to assess the damage. By running Bravo again in the normal way, you will instantly sacrifice all chance of benefiting from the log mechanism, since the log allows replay only of the most recent session. What you want to do instead is run the program "BravoBug" ("Bravo/R" is not an adequate

substitute). It wouldn't be a bad idea to ask a wizard for help also. While you are looking for a wizard, try and think of some good answer to the question "Why are you using Bravo, anyway?", which said wizard will almost certainly ask.

Printing

In general, our printers are built by taking a Xerox copier and adding electronics and a scanning laser that produce a light image to be copied. There are many different types of such printers, and there are multiple instances of each printer type as well. There are also many different programs that would like to produce printed output. The Press print file format was our first answer to the problem of allowing every printing client to use every printer. Press files are the Esperanto of printing. Most print servers demand that the documents that you send to them be in Press format. This means you have to convert whatever you have in hand (often text) to Press format before a server will deign to print it.

Press file format <PRESS> is hairy, and some print servers don't support the full generality of Press. Generally, however, such servers will simply ignore what they can't figure out, so you can safely send them any Press file you have.

A Press file can ask that text be printed in one of an extensive collection of standard fonts. Unfortunately, you must become a wizard in order to print with your own new font. You can't use a new font unless it is added to the font dictionary on your printer, and adding fonts to dictionaries is a delicate operation: a sad state of affairs. If the Press file that you send to a printer asks for a font that the printer doesn't have, it will attempt a reasonable substitution, and, in the case of Spruce, tell you about the substitution on the break page of your listing. If you have chronic font difficulties of this sort, contact a wizard.

There is a new print file format, called Interpress. The print servers that are part of the Star product speak a dialect of Interpress as will most other new Xerox printers. A print file in Interpress format is called a master. CSL's local plans for printing Interpress masters involve converting them first into a printer-dependent print file in so-called PD format (with conventional extension ".pd"). From there, a relatively simple driver program on each printer should be able to produce the final output.

PARC has a variety of printers available for your hardcopy needs. We have high volume printers for quantities of text, listings, and documentation; we have slower printers with generally higher quality for more complex files; and we have very slow printers for extremely high quality. All of our current printers except Platemaker offer 384 spots per inch and share a common font dictionary. We use two different software systems for printing Press files, both running on Altos: one is called Spruce, and the other is called (confusingly) Press. Spruce offers speed and spooling, but it can only image characters and rules, and not too many of them. This makes it limited in graphics applications. Furthermore, Spruce is limited to the particular sizes of fonts that it has stored in its font dictionary: it does not know how to build new sizes by converting from splines. Press is slower, but can handle arbitrary bitmaps, and can produce odd-sized fonts from splines.

CSL is developing Interpress printing capabilities. Printing ".pd" files is now an option on most Press printers (that is, on printers running the program Press as opposed to Spruce). Just ship your ".pd" file to the printer in the standard way: it is smart enough to figure out whether what you have sent it is in PD or Press format, and it will invoke PDPrint or Press as appropriate. Documentation on these two printing programs is available, by the way <PDPRINT, PRESS>. PD printing should not be undertaken without consultation with a wizard.

Dover printers run Spruce for high volume printing, producing a page per second. CSL's Dover, named Clover, is found in room 2106; ISL's Dover, named Menlo, is in room 2305. Samples of the Dover font dictionary may be found next to Clover and Menlo. Instructions for modifying the queue and generally running these Spruce printers are to be found next to their Alto terminals.

Lilac is CSL's color Press printer and may be found in room 2106 with Clover. It is a three color, composite-black machine; it generally produces good quality output, but is occasionally temperamental. Anyone interested in color printing or the state of Lilac should join the distribution list LilacLovers[↑].pa.

In room 2301, there are an assortment of black and white Press printers, answering variously to the names of RockNRoll, Quoth, and Stinger. The printers are two Ravens (Raven is a Xerox product), one Hornet, and one Gnat (the latter two are prototypes). The print quality is normally excellent. Instructions for interpreting status displays are posted locally. To be informed of which printer is functioning and where, join the list ISLPrint[↑].pa. There should be three printers up for most of the summer. Periodically one or another of these or Lilac are pre-empted for debugging.

Our best quality printer is Platemaker, which is normally operated at 880 spots per inch, but can be run up to 2200 spi; it is not normally useful to go beyond 1600. Platemaker uses a laser to write on photographic paper or film. Color images can be done in individual separations, which are then merged using the Chromalin process. The Platemaker printing process is used for final prints of fine images or for printing masters for publication. If you wish to have something printed, speak to Julian Orr, Eric Larson, or Gary Starkweather.

Sending and Receiving Mail

We rely very heavily on an electronic mail system. We use it for mail and also for the type of announcement that might, in other environments, be posted on a physical or electronic bulletin board. In our environment, a physical bulletin board is pretty useless, since people spend too much of their days staring at their terminals and too little wandering the halls. Electronic bulletin boards might work satisfactorily. But a bulletin board, being a shared file to which many people have write access, is a rather tricky thing in a distributed environment. It probably presupposes a distributed transactional file server, for example. Mumble. For whatever reason, the fact remains that we don't have an electronic bulletin board facility at the moment. As a result, announcements of impending meetings, "for sale" notices, and the like are all sent as messages directed at expansive distribution lists. If you don't check your messages once a day or so, you will soon find yourself out of touch (and saddled with a mailbox full of obsolete junk mail). And conversely, if you don't make moves to get on the right distribution lists early, you may miss lots of interesting mail. This business of using the message system for rapid distribution of announcements can get out of hand. One occasionally receives notices of the form: "meeting X will start in 2 minutes--all interested parties please attend".

Grapevine is the distributed transport mechanism that delivers the local mail [33]. When talking to Grapevine, individuals are referred to by a two-part name called an "R-name", which consists of a prefix and a registry separated by a dot; for example, "Ramshaw.pa" means Ramshaw of Palo Alto. In addition to delivering the mail, Grapevine also maintains a distributed database of distribution lists. A distribution list is also referred to by an R-name, whose prefix conventionally ends in the character up-arrow, as in "CSL↑.pa". Distribution lists are actually special cases of a construct called a Grapevine "group". Groups can be used for such purposes as controlling access to IFS directories. There is a program named Maintain that allows you to query and update the state of the distribution list database. In fact, there are two versions of Maintain: the documented one with the unfortunate teletype-style user interface is used from within Laurel or the Mesa Development Environment <34>; the undocumented one with the futuristic menu interface is used from within Cedar. Some distribution lists are set up so that you may add or remove yourself using Maintain. If you try to add yourself to Foo↑.pa and Maintain won't let you, the proper recourse is to send a message to the distribution list Owners-Foo↑.pa, asking that you please be added to Foo↑.

At the moment, Grapevine pretty much has a monopoly on delivering the mail. But there are several different programs that give users access to Grapevine's facilities from different environments. From an Alto, one uses Laurel, which is mentioned elsewhere as a pioneer of modelless editor interfaces. Even if you aren't a Laurel user, I recommend that you read Chapter 6 of the Laurel Manual [35], which is an enlightening and entertaining essay on proper manners in the use of the mail system. In the Mesa Development Environment, the program Hardy provides services analogous to Laurel's. From within Interlisp, most folk use Lafite, whose documentation appears as <LAFITE.PRESS>. Finally, in case travel should take you away from your multi-function personal workstation, there are servers on the Internet known by the name "Lily" to whom you can connect from any random teletype in order to peruse the mail sitting in your Grapevine mailbox.

Some Tidbits of Lore

About CSL

CSL has a weekly meeting on Wednesday afternoons called Dealer, starting at 1:15. The name comes from the concept of "dealer's choice"--the dealer sets the ground rules and topic(s) for discussion. When someone says she will "give a Dealer on X", she means that she will discuss X at some future weekly meeting, taking about 15 minutes to do so (plus whatever discussion is generated). Generally, such discussions are informal, and presentations of half-baked ideas are encouraged. The topic under discussion may be long-range, ill-formed, controversial, or all of the above. Comments from the audience are encouraged, indeed, provoked. More formal presentations occur at the Computer Forum on Thursday afternoons; the Forum is not specifically a CSL function, and it is open to all Xerox employees, and sometimes also to outsiders. Dealers are also used for announcements that are not appropriate for distribution by electronic mail. Members of CSL are expected to make a serious effort to attend Dealer.

Some Code Phrases

You may occasionally hear the following incomprehensible phrases used in discussions, sometimes accompanied by laughter. To keep you from feeling left out, we offer the following translations:

"Committing error 33"

(1) Predicating one research effort upon the success of another. (2) Allowing your own research effort to be placed on the critical path of some other project (be it a research effort or not). Known elsewhere as Forgie's principle.

"You can tell the pioneers by the arrows in their backs."

Essentially self-explanatory. Usually applied to the bold souls who attempt to use brand-new software systems, or to use older software systems in clever, novel, and therefore unanticipated ways ... with predictable consequences. Also heard with "asses" replacing "backs".

"We're having a printing discussion."

Refers to a protracted, low-level, time-consuming, generally pointless discussion of something peripherally interesting to all. Historically, printing discussions were of far greater importance than they are now. You can see why when you consider that printing was once done by carrying magnetic tapes from Maxc to a Nova that ran an XGP.

Fontology

The body of knowledge dealing with the construction and use of new fonts. It has been said that fontology recapitulates file-ogeny.

"What you see is what you get."

Used specifically in reference to the treatment of visual images by various systems, e.g., a Bravo screen display should be as close as possible to the hardcopy version of the same text. Also known as some circles by the acronym "WYSIWYG", pronounced "whiz-ee-wig".

"Pop!"

THIS phrase means that the conversation has degenerated in some respect, often by becoming enmeshed in nitty-gritty details. Feel free to shout out one or more of these phrases if you feel that a printing discussion has been going on long enough. If two participants in a large meeting begin discussing details that are of interest to them but not of interest to the group as a whole, shout "Off-line!" instead.

"Life is hard"

Two possible interpretations: (1) "While your suggestion may have some merit, I will behave as though I hadn't heard it." (2) "While your suggestion has obvious merit, equally obvious circumstances prevent it from being seriously considered." The charm of this phrase lies precisely in this subtle but important ambiguity.

"What's a spline?"

"You have just used a term that I've heard for a year and a half, and I feel I should know, but don't. My curiosity has finally overcome my guilt." Moral: don't hesitate to ask questions, even if they seem obvious.

Hints for Gracious Living

There are a couple of areas where life at PARC can be made more pleasant if everyone is polite and thoughtful enough to go to some effort to help out. Here are a few words to the wise:

Coffee

Most groups have coffee alcoves where tea, cocoa, and several kinds of coffee are available. All coffee drinkers (not just the secretaries or some other such barbarism) help out by making coffee. If you are about to consume enough coffee that you would leave less than a full cup in the pot, it is your responsibility to make a fresh pot, following the posted instructions. There are lots of coffee fanatics around, and they get irritated beyond all reason if the coffee situation isn't working out smoothly. For those coffees for which beans are freshly ground, the local custom is to pipeline grinding and brewing. That is, you are expected to grind a cup of beans while brewing a pot of coffee from the previous load of ground beans. This speeds up the brewing process for everyone, since a load of ground beans is--at least, had better be--always ready when the coffee pot runs out.

Sharing Office Space

Be warned as well that some people are unbelievably picky about the state of their offices. The convention is that any Alto in an empty office is fair game to be borrowed. Private machines may be borrowed only by prior arrangement with their owners, because of the problems of sharing disk space. If you use someone's office for any reason, take care to put everything back exactly the way it was. Don't spill crumbs around, or leave your half-empty cocoa cup on the desk, or forget to put the machine back in the state that you found it, or whatever. Of course, lots of people wouldn't mind even if you were less than fanatically careful. But some people do mind, and there is no point in irritating people unnecessarily.

Sharing printers

When you pick up your output from a printer, it is considered antisocial merely to lift your pages off the top of the output hopper, and leave the rest there. Take a moment to sort the output into the labelled bins. Sorting output is the responsibility of everyone who prints, just as making coffee is the responsibility of everyone who drinks (coffee). Check carefully to make sure that you catch every break page: short outputs have a way of going

unnoticed, and hence being missorted, especially when they come out right next to a long output in the stack. The rule for determining which bin is to use the first letter that appears in the name on the break page. Thus, "Ramshaw, Lyle" should be sorted under "R", while "Lyle Ramshaw" should be sorted under "L". A trickier question is what to do with output for "Noname", or the like. Following the rule would suggest filing such output under "N", but that doesn't seem very helpful, since the originator probably won't find it. Check the contents and file it in the right box if you happen to recognize whose output it is; otherwise, either leave it on top of the printer or stick it back in the output hopper.

The phone system

If you make a significant number of personal long-distance phone calls from Xerox phones, it is your responsibility to arrange to reimburse Xerox for them. This may not be that easy, either, since phone bills take quite a while (six weeks or so) to percolate through the bureaucracy upstairs, and the said bureaucracy also has a lot of trouble figuring out where to send the phone bills of new people, and people who move around a lot. Just because it is easy to steal phone service from Xerox doesn't make it morally right; if you think you aren't being paid enough, you should start agitating for a raise. If enough suspicious calls are made without restitution, PARC (being a bureaucracy) will impose some bureaucratic "solution" on all of us.

So as not to end on a sour note, let's discuss how the phone system works, anyway. The offices within PARC have four-digit extensions within the 494 exchange, a system known as Centrex; to dial another office, those four digits suffice. Dialing a single 9 as the first digit gives you an outside line, and you are now a normal customer of Ma Bell: see a phone book for more details (Oh, come now, surely you know about phone books!). Dialing a single 8 gives you different sounding dial tone, and puts you onto the IntelNet (not to be confused with the InterNet). The IntelNet is a Xerox-wide company phone system, complete with its own phone book, and its own phone numbers. If you are calling someone in some remote part of Xerox, you can save Mother Xerox some bread by using the IntelNet instead of going straight out over Ma Bell's lines. On the other hand, you may not get as good a circuit to talk over--although this situation is frequently said to be improving. Furthermore, through the wonders of modern electronics, you can dial any long-distance number over the IntelNet. Just use the normal area code and Ma Bell number: the circuitry is smart enough to take you as far as possible towards your destination along IntelNet wires, and then switch you over to Ma Bell lines for the rest of the trip. Using the IntelNet doesn't start to save money until the call is going a fair distance; therefore, the IntelNet doesn't let you call outside numbers in area codes 408, 415, and 916--better to just dial 9.

One more thing: after you have dialed a number on the IntelNet, you will hear a funny little beeping. At that point, you are being asked to key in a four-digit number to which the call should be billed. You should use the four-digit extension number for your normal office phone under most circumstances. Calls made by dialing 9 instead of 8 are always charged to the phone from which they are placed.

The first three rings (roughly speaking) of an incoming call occur only in your office. The next roughly three rings happen both at your office phone and at a receptionist's phone, centrally located in the laboratory. During normal business hours, the receptionist's phones are staffed; thus, someone will at least take a message for you, and leave it on a little slip of paper in your physical message box. If the second three rings go by without either of those two phones answering, the call is then forwarded to the guards desk downstairs (I believe).

If you are expecting a call but won't be near your normal phone, a call forwarding facility exists: dial 106 and then the number to which you want your calls to be forwarded. Later on (try not to forget), you dial 107 on your normal phone to cancel the forwarding. When I forward my phone, I turn the phone around physically, so that the touch-pad faces the wall. This helps me to remember to cancel the forwarding again later, at which point I turn the phone back the normal way. There is also a way to transfer incoming calls to a different Xerox number: Depress the switch hook once, and dial the destination number; when the destination answers, you will be talking to the destination but the original caller won't be able to hear your conversation; depressing the switch hook again puts all three of you on the line; then you can hang up when you please. If the destination doesn't answer, depressing the switch hook once again will flush the annoying ringing or busy signal.

References

Reference numbers in [square brackets] are for conventional hardcopy documents. Many of them are Xerox reports published in blue and white covers; the CSL blue-and-whites are available on bookshelves in the CSL Alcove. Reference numbers in <angle brackets> are for on-line documents. The path name for such files is given herein in the form

[FileServer]<Directory>SubDirectory>FileName.Extension

for backward compatibility with earlier systems. Recently, the simpler alternative form

/FileServer/Directory/SubDirectory/FileName.Extension

has begun to come into local currency, but some systems still demand brackets rather than slashes.

<n>: The generic form for a reference to an on-line document.

[n]: The generic form for a reference to a hardcopy document.

[SUNSET]: Sunset New Western Garden Book. Lane Publishing Company, Menlo Park, CA, 1979. The definitive document on Western gardening for non-botanists; 1200 plant identification drawings; comprehensive Western plant encyclopedia; zoned for all Western climates; plant selection guide in color.

[GRAYSCALE]: John E. Warnock. The Display of Characters Using Gray Level Sample Arrays. blue-and-white report CSL-80-6.

[OPTICALMOUSE]: Richard F. Lyon. The Optical Mouse, and an Architectural Methodology for Smart Digital Sensors. blue-and-white report VLSI-81-1.

[ETHERNET]: The Ethernet Local Network: Three Reports. blue-and-white report CSL-80-2.

[ETHERNET]: John F. Shoch, Yogen K. Dalal, Ronald C. Crane, and David D. Redell. Evolution of the Ethernet Local Computer Network. blue-and-white report OPD-T8102.

<TOPOLOGY>: [Indigo]<AltoDocs>NetTopology.press. Contains a picture of the entire internetwork configuration in seven pages. It is out of date. All such documents are always out of date.

[PUP]: David R. Boggs, John F. Shoch, Edward A. Taft, and Robert M. Metcalfe. Pup: An Internetwork Architecture. blue-and-white report CSL-79-10.

[NS]: Internet Transport Protocols. Xerox System Integration Standard report X SIS 028112, December 1981.

[NS]: Courier: The Remote Procedure Call Protocol. Xerox System Integration Standard report X SIS 038112, December 1981.

[ALTO]: C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs. Alto: A personal computer. blue-and-white report CSL-79-11.

[DORADO]: The Dorado: A High-Performance Personal Computer; Three Papers. blue-and-white report CSL-81-1.

<DORADOBOOTING>: [Indigo]<DoradoDocs>DoradoBooting.press. Describes how to boot a Dorado, and how to configure it to boot in various ways.

[MESA]: Mesa programmers manual.

[SMALLTALK]: Adele Goldberg and David Robson. Smalltalk-80: The Language and Its Implementation. book published by Addison-Wesley, 1983.

[22]: Interlisp Reference Manual. 1983.

[23]: Papers on Interlisp-D. blue-and-white report CIS-5 (also given the number SSL-80-4), Revised version, July 1981.

[EPE]: L. Peter Deutsch and Edward A. Taft, editors. Requirements for an Experimental Programming Environment. blue-and-white report CSL-80-10.

<CEDAR>: [Indigo]<Cedar>Documentation>Manual.df. Hardcopies are entitled The Cedar Manual.

[ALTO]: Alto User's Handbook. Internal report, published in a black cover. The version of September, 1979 is identical to the version of November, 1978 except for the date on the cover and title page. Includes sections on Bravo, Laurel, FTP, Draw, Markup, and Neptune

<ALTOSYSTEMS>: [INDIGO]<AltoDocs>SubSystems.press. Documentation on individual Alto subsystems, collected in a single file. Individual systems are documented on [INDIGO]<AltoDocs>systemname.TTY, and these files are sometimes more recent than SubSystems.press.

[BRAVO]: Jerome, Suzan. Bravo Course Outline. Internal report, published in a red cover. Oriented to non-programmers.

<30>: [Indigo]<PrintingDocs>PressFormat.press. Describes the Press print file format.

<PRESS>: [Indigo]<PrintingDocs>PressOps.press. Describes the Press printing program.

<PD>: [Maxc]<PrintingDocs>PDPrintOps.press. Describes the PDPrint printing program.

[GRAPEVINE]: Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder. Grapevine: an Exercise in Distributed Computing. blue-and-white report CSL-82-4.

<MAINTAIN>: [Ivy]<Laurel>Maintain.press. Documentation for the teletype version of Maintain, the version that is used from within Laurel or Tajo.

[LAUREL]: Douglas K. Brotz. The Laurel Manual. blue-and-white report CSL-81-6.

The Call-C-Function MISCN opcode

This opcode calls the specified C function, performing conversion of arguments and result as needed, and returning an indication of any errors it encounters.

(MISCN CALL-C *Function Conversion-spec Return-Code* &REST *Args-To-C-Fn*)

Function is a Lisp integer (FIXP or SMALLP) containing the address of the function to be called. CALL-C checks for some special values, 0 (meaning the function was never loaded) and -1 (meaning the function was loaded once, and subsequently unloaded at user request), and -2 (meaning that the function has been loaded, but there are unresolved externals).

Conversion-spec specifies how the arguments and function-result are to be converted.

This is a Lisp pointer to a block of 16-bit entries:

```
+-----+
+ Result Conversion Spec |
+-----+
|   Arg 0 Conversion   |
+-----+
|   Arg 1 Conversion   |
+-----+
|           etc.       |
+-----+
|           -0-        |
+-----+
```

Possible values for the conversion fields:

```
0    VOID (return only, return NIL)
1    int (Lisp SMALLP/FIXP <=> 32-bit integer)
2    char (Lisp SMALLP/CHARACTER <=> char)
3    float
4    long
5    short
6    lisp
7    cpointer
```

Return-Code is a FIXP cell into which CALL-C places a return value. Possible values are:

```
0    Successful call and return
+n   conversion error on argument n
-1   conversion error on result
-2   signal encountered while running C??
```

DEFFOREIGN—Define a foreign function for lisp.

This macro tells Medley about a foreign function—its arguments, what type of result it returns, etc. It also creates a Medley function you can call to invoke the foreign function.

```
(DEFFOREIGN Function Result-Type ArgList &KEY :function-name)
```

Function is a symbol, the Lisp name for the function. *Function* is given a definition that results in the foreign function being called. *Function* returns what the foreign function returns, after conversion to a Lisp datatype.

Result-Type is a symbol specifying what type of data the foreign function returns, and how it is to be converted to a Lisp type. Possible values are:

- :void The function returns no interesting value. Function will always return NIL.
- :long
- :short
- :int The function returns an integer. It is converted to a FIXP or a SMALLP.
- :char The function returns a character. It is converted to a SMALLP.
- :float The function returns a floating-point number. It is converted to a Lisp FLOATP.
- :lisp The function returns a lisp-pointer, which isn't converted, but DOES get reference counted.
- :byte ?? same as character, but converted to what??
- :cpointer The function returns a pointer to a block of storage not in the Lisp virtual memory image. This may be a pointer to a C structure, or whatever. It is intended for use with CBLOCKRECORD and DEFCSTRUCT
- <c type> Where <C type> is a type defined using DEFCSTRUCT. The result is a pointer to a block of storage no in the Lisp virtual memory image....

ArgList is a list of symbols, each specifying what kind of data the foreign function expects for a given argument. The possible values are as above.

Function-name is a symbol or string containing the true name (as far as the foreign language is concerned) of the function you want to call when the Lisp *Function* is called.

Chat Streams

A *chat stream* is a connection between two processes oriented towards terminal service, but not necessarily restricted to that. A chat stream is inherently bi-directional so it is represented by two Interlisp-D streams; one each for input and output. The input stream is considered the primary handle on the connection and is used wherever operations are preformed that are not inherently only input or output. The following operations are available for chat streams (as well as the normal stream operations). In general these operations return true if the operation was successful, NIL if it could not be done:

(**CHAT.SETDISPLAYTYPE** INSTREAM CODE) tells the remote process that a particular type of terminal (designated by the numeric parameter CODE) is being emulated.

(**CHAT.LOGININFO** INSTREAM HOST NAME) determines if LOGIN or ATTACH should be used when logging into HOST with username NAME. This fn is only called by CHAT if it knows how to ATTACH. Returns LOGIN, ATTACH, NIL, or WHERE (when there are multiple jobs to attach to).

(**CHAT.SENDSCREENPARAMS** INSTREAM WIDTH HEIGHT) sends the dimentions of the virtual screen to the remote process.

(**CHAT.FLUSH&WAIT** INSTREAM) insures that the remote process as seen (but not necessarily acted upon) all data written so far.

(**CHAT.OPTIONMENU** INSTREAM) returns a MENU of protocol specific options that can be presented to the user.

Adding a new protocol

To add a new type of chat stream a filter function should be added to the list CHAT.PROTOCOLS which, when given a host name, will return a function. When this function is applied to the host name, it should return a dotted pair (INSTREAM . OUTSTREAM). STREAMPROP should be used to associate the methods for implementing the chat stream operations with INSTREAM. The property names are SETDISPLAYTYPE, LOGININFO, FLUSH&WAIT, SENDSCREENPARAMS, and OPTIONMENU. Only methods for implemented operations need be present.

The function (**ADD.CHAT.MESSAGE** STREAM MSG) is available for getting protocol specific messages to the user. Messages will be typed out on chat window by CHAT, or can be found on the STREAM property MESSAGE by programs. Delimiting white space is added to MSG.

Caveat: CHAT changes the ENDOFSTREAMOP of INSTREAM to return -1 the first time it is called, and then restore the original ENDOFSTREAMOP with the expectation that the -1 will get returned as the value of the BIN that caused the EOS op to be called. While this works for all the protocols I've tried so far, it might not work in general. This kludge is necessary in order to avoid the time-consuming alternative of looping on READP and EOF. It also did not seem appropriate that the chat stream "interface" implement this kludge, though it might be necessary to resort to that.

Errors and related matters in CommonLoops - A Proposal

Henry Thompson
11 August 1985

This proposal represents an attempt to provide a set of control primitives for CommonLoops which will

- 1) Support the existing Interlisp error handling mechanisms (including ERROR and friends, ERRORSET and friends, RESETLST and friends, ERRORTYPELST, BREAKCHECK and its consequences, and the relationships between ERRORX, FAULT1 and BREAK1, all in the context of spaghetti stacks and the existing process mechanisms;
- 2) Support the CommonLisp constructs catch, throw, unwindprotect, the relationship of unwindprotect to go and return(-from), error, cerror and warn;
- 3) Substantially reproduce the functionality of the ZetaLisp signalling facility;
- 4) Be a reasonably plausible attempt to take the high ground wrt whatever proposals the CommonLisp working party on error handling come up with;
- 5) Be a Good Thing in its own right.

Needless-to-say trying to satisfy all these goals simultaneously is not possible without some compromises, but I think what follows is a good first cut.

The starting point for this design is the Mesa signal and error mechanism, with one key idea borrowed from ZetaLisp. We start with the notion that it must be possible to unwind the stack, either as a part of non-local transfers of control, or as a consequence of abnormal termination. We add to this the notion that at certain points on the stack we may wish to take some action if such an unwinding is underway. Finally we discriminate between actions mandated at some point on the stack but taking place before the unwinding actually starts, and actions which occur at a point on the stack as the unwinding goes by.

Some terminology is in order at this point.

We call the unwinding process **unwinding**. We call the points on the stack at which action wrt unwinding may be specified *Unwind Control Points*, or *UWCPoints* for short. The process by which the user or the system announce circumstances which may provoke unwinding is called signalling, and the concrete representation of the circumstances at issue is called a condition. A condition is a CommonLoops class, and should be a sub-type of class Condition.

Unwind Control Points

UWCPoints are central to this proposal. They provide a vehicle for all activities associated with unwinding, both before the fact in the **context** of signalling, and as the stack unwinds. A UWCPoint is created with a call to the spread lambda UWCP:

```
(uwcp body catch exit always-do-exits).
```

Its definition is simple:

```
[progl (apply body nil)
      (cond (always-do-exits
              (apply exit '(normal nil)))],
```

but it is what goes on behind the scenes which is important. A UWCPoint is basically a way of evaluating body (actually applying it as a function of no arguments, to allow for closures) in a context which affects what happens in the case of signalling and/or unwinding occurring within the dynamic extent of that UWCPoint. If no signalling or unwinding occurs, the value of uwcp is the value of body. If the stack is unwound past this point, (apply exit '(unwind <condition>)) will be performed on the way past. This

suffices for unwindprotect and RESETLST (note I use (f:l args . body) throughout as short for (function (lambda args . body))):

```
(unwindprotect form . cleanups) =>
(uwcp (f:l () form)
  nil
  (f:l (exit-key c)
    (selectq exit-key
      ((unwind normal) . cleanups)
      nil))
  t)
(RESETLST . forms) =>
(LET ((LISPMHIST LISPMHIST)
      (RESETX RESETVARSLST))
  (DECLARE (SPECVARS RESETX))
  (uwcp (f:l () . forms)
    nil
    (f:l (exit-key c)
      (selectq exit-key
        (normal (RESETRESTORE RESETX))
        (unwind (RESETRESTORE RESETX 'ERROR))
        nil))
    t))
```

RESETSAVE and RESETRESTORE are exactly as before. Note that this means that closures may appear on RESETVARSLST in case of e.g.

```
(RESETSAVE xx (LIST (f:l ...) yy))
```

but this is presumably just what is wanted.

Unwinding

The actual unrolling of the stack is performed by the spread lambda UNWIND!:

```
(unwind! frame exit-key condition).
```

frame is a stack pointer to the frame to unwind to. condition is an instance of some subclass of Condition, descriptive of what is causing the unwinding. In the simple case frame will be a UWCPPoint, in which case exit-key will determine what happens when we get there.

unwind! works by scanning the stack upwards via c-links [Larry - should this be a-links? I notice that e.g. GO and RESET chase a-links, not c-links?] from its own frame until it gets to frame. Along the way, whenever it encounters a UWCPPoint, it applies the exit argument of that UWCPPoint to (list 'unwind condition). When it gets to frame there are two cases. If frame is a UWCPPoint, then the unwinding completes with (retapply frame <the exit argument of frame> (list exit-key condition) t). If frame is not a UWCPPoint, then the unwinding completes with (apply exit-key (list frame)). This latter case is for non-local 'go's and 'return(-from)'s, see Note on Non-local Xfers below.

A crucial implementation point is that unwind! releases frame before scanning the stack, and uses raw pointers during its scan. This is to prevent the stack from inadvertently being tied down if some unwind clause pre-empts the unwinding by doing its own non-local transfer, something which cannot (and indeed probably should not) be ruled out. This has the further consequence that if frame is not a UWCPPoint then it must be re-constituted before being having exit-key applied to it, see above. What would make sense is for unwind! to be defined to get the raw pointer out of the stack pointer, invoke an opcode which does the actual stack scan in microcode, and then reconstruct the stack pointer and do the final apply or retapply.

Signalling

Conditions are signalled with the spread lambda raise-signal:

```
(raise-signal condition can-resume neednt-catch) .
```

condition must be an instance of some sub-class of Condition. It identifies the circumstances which provoked the signalling, and may contain relevant parameters. If can-resume is non-nil, then the signal may be resumed, otherwise not (see below). If neednt-catch is non-nil, then the signal need not be caught, otherwise a condition Uncaught will be signalled if it is not caught.

raise-signal works by scanning the stack upwards looking for UWCPPoints. When it finds one it applies the catch argument thereof to (list condition). If the result is nil, it continues the scan. If the result is non-nil we say the signal has been caught at that UWCPPoint.

What happens next depends on the type of the result. If it is not a list it is called the exit key, and raise-signal exits the signal by causing the stack to unwind to the UWCPPoint which caught the signal by calling (unwind! <the UWCPPoint> <the exit key> condition). If it is a list then its first element is considered a resume value. If can-resume is non-nil, then the resume value is returned as the value of the call to raise-signal, otherwise an error condition CantResume will be signalled.

If the stack is scanned all the way to the top without the signal being caught, then if neednt-catch is non-nil, the value of raise-signal is nil. Otherwise, the condition Uncaught will be signalled, with instance variables recording the parameters to raise-signal. If it is exited, fine. If it is resumed, the value returned is the value of the call to raise-signal. Otherwise a break is caused around the call to raise-signal.

Note that the application of the catch argument at each UWCPPoint is done in the dynamic context of the call to raise-signal, but as the catch argument is lexically in the context of the call to uwcp, if it is a closure its non-special variable references will be to that context.

Three macros are provided for the common cases:

```
(signal condition) => (raise-signal condition t nil) - can
resume, must be caught
```

```
(error condition) => (raise-signal condition nil nil) - can't
resume, must be caught
```

```
(notify condition) => (raise-signal condition t t) - can resume,
needn't be caught
```

ERROR! and ERRORSET

ERROR! and control-E are now defined as (error \Abort), where \Abort is an instance of class Abort.

ERRORSET is now defined as

```
(lambda (form flag)
  (uwcp (f:l () (list (eval form)))
        (f:l (condition)
              (select-type condition
                (Abort t)))
        (f:l (exit-key condition)
              nil))).
```

t is by convention the 'do-nothing' exit key. Note the semantics of ERRORSET are subtly changed by this definition. It is no longer the case that ERRORSET flatly stops the stack from unwinding. What it does is catch Abort, which means it short-stops ERROR!/control-E/^, as it used to, but not unwinding associated with other signals which have been caught overhead. This seems to me to be what is wanted. One could

of course write a catch phrase for Condition to insure catching anything and everything, but that would be pretty dangerous.

catch and throw

These CommonLisp functions are implemented in terms of a sub-class of Condition called Throw:

```
(catch tag . body) =>
(uwcp (f:l () body)
      (f:l (condition)
            (select-type condition
                          (Throw (cond ((eq condition:tag tag) 'caught))))))
(f:l (exit-key condition)
      (selectq exit-key
                (caught condition:value)
                nil)))

(throw tag form) =>
(let ((value form))
  (raise-signal (create Throw tag_tag value_value) nil nil))
```

ERROR, ERRORX and BREAKCHECK

ERROR has a name conflict with the new (and CommonLisp) error - I propose changing its name to old-error and in the short term dicriminating on the basis of the type of the first argument. The only change to old-error is that it now passes its nobreak argument on to ERRORX, which passes it to ERRORX2. ERRORX is unchanged except for that. FAULT1 calls ERRORX2 instead of replicating it - more on this later. ERRORX2 calls BREAKCHECK as before, and then constructs an instance of class SystemError, which is a sub-class of condition, including the error number, message, position, BREAKCHK and PRINTMSG as instance variables, and signals it.

At the top of every process there is a UWCPPoint, which inter alia catches Abort, and also handles most of what used to be in ERRORX2. It catches SystemError in order to implement both the built-in and user specified error type list clauses, declining to catch the signal if they don't apply. It catches Uncaught if what wasn't caught was a SystemError, and then either produces the appropriate call to BREAK1 or raises Abort, depending on the recorded value of BREAKCHK. BREAK1, however invoked, signals AboutToBreak before doing anything else. This is all a bit hairy, but the code has been worked out and will be forthcoming.

Non-Local Transfers of Control

Lexical scoping of goto tags and block labels in CommonLisp represents a bit of bother in the Interlisp context. For instance

```
(prog      ((damnFun (f:l (arg)
                          (if (weird arg)
                              then (go bother)
                              else (process arg))))))
  (return (unwindprotect (apply* damnFun 'foo) (cleanup)))
  bother
  (return 'lost))
```

works not only in the sense that if 'foo is weird, the value of the prog is 'lost, but also that in that case the unwindprotect is observed and cleanup is called. Now I don't understand how closures are to be implemented in CommonLoops, but I assume the following must be true:

- 1) The interpreter will continue to exist independently of the compiler (if this is false that just simplifies things a bit).

- 2) There is a way of identifying on the stack lexical scoping boundary points - that is to say, I presume, frames created by the application of a function definition or a non-quoted argument to apply.
- 3) The definition of function is such that a closure knows of every non-local lexical variable reference, goto tag and block label within it. This implies inter alia that when running interpreted all macros are expanded by function, and that evaluating or compiling calls to function may produce uba, no such tag or no such label errors.
- 4) The stack entry for a local variable which is referenced by a closure contains not its value but an invisible pointer to a 'free-floating' value cell, which is also pointed to by the closure.
- 5) When the interpreter needs the value of a lexical variable, it scans the stack up to the first boundary point and no further. The compiler will presumably be pretty much as now - collapsing all bindings upwards to the boundary frame in so far as possible, and building in references to the right points in the right frames.
- 6) Thus when a closure is applied to anything, a frame can be built which has entries for all its non-local lexical variable references which will do the right thing.
- 7) A similar, although messier, approach will work for labels and tags. Messier because although such 'free-floating' value cells may persist after their 'home' frame has gone, 'free-floating' labels and tags must be invalidated when the frame they are based on goes away (see e.g. page 41 of the CommonLisp book). Most of this hair is probably necessary simply to allow 'go's from inside nested progs in any case:
 - a) Compiled PROG and interpreted \PROG0 frames have two new sorts of entry for tags and labels. Each has two fields, an atom number and a pointer. Every PROG or \PROG0 frame has one tag entry with the atom number for each goto tag it owns, and one label entry for the label of the block, usually nil. In the case where no closures are involved, the pointer field of each entry corresponding to a goto tag contains the appropriate p-counter to transfer to in the case of compiled PROG frames, and the appropriate tail of the PROG body in the case of interpreted \PROG0 frames. The pointer field of the label entry contains nil. By convention (see page 120 of the CommonLisp book) every bounding frame also has a label entry for its frame-name.
 - b) When a go, return, or return-from is *evaluated*, the stack is scanned to the first boundary point looking for an appropriate tag/label entry. When one is found we call


```
(unwind! <the frame> (f:l (frame) (do-go frame tag
<value of entry>)))
```

 if evaluating a go, otherwise


```
(unwind! <the frame> (f:l (frame) (do-return frame label
<value of entry>
<arg to return>)))
```
 - c) When a closure is constructed whose body refers to non-local tags or labels, it constructs (or finds, see below) a stack pointer for the frame in which the tag/label is bound, and includes that together with the tag/label in the closure. A pointer to this stack pointer is left in a distinguished part of the frame, so that it can be re-used by other closures, and so that it can be released when the frame goes away. Note that this means it is a special sort of stack pointer, in that its reference to the frame must not be counted.

This also implies an additional cost both in size and time to return for every frame, but I don't see how it can be avoided.

- d) When a closure is applied which includes such tags/labels, tag and label entries containing the appropriate atom numbers and the associated stack pointer are included in the constructed frame.
- e) It follows from all this that do-go and do-return implement the distinction between local and non-local transfers. If the value of the entry they are passed is not a stack pointer, then they effect the local transfer, via retfrom for do-return, and by appropriate hacking of the PROG (compiled) or \PROG0 (interpreted) frame followed by retto in the case of do-go. If they do get a stack pointer, then they convert themselves into the local case by getting the entry from for the tag/label from the frame pointed to, and doing a further unwind! to that frame with an appropriate re-call of themselves as the exit-key argument. Needless-to-say, if the stack pointer has been released, we get an error.
- f) All this is unnecessary for compiled transfers which don't cross any frame boundaries, which can still be coded open.
- g) It is not clear to me what will happen in the case of e.g. (eval '(go foo)). If we take the CommonLisp manual seriously, this will fail, as it probably should, because eval will set up a bounding frame with no lexical variables or tags, but then so will most existing uses of eval... I guess this gets beyond what I can reasonably hope to second-guess...

enable

A special form is provided which will be the standard way of producing UWCPoints. It is modelled on the Cedar Mesa ENABLE form, and looks like this:

```
(enable
  c1 => a1 a2 a3 ...
  .
  .
  .
  cn => n1 n2 n3 ...
form
  k1 -> ea1 ea2 ...
  .
  .
  .
  kn -> en1 en2 ...)
```

The double arrow lines above are called catch phrases, the single arrow lines are called exit phrases. Evaluates form so as to catch conditions c1, ... cn if they are signalled during its evaluation. If e.g. c1 is signalled, the forms a1 ... an (the catch phrase for c1) will be evaluated in the context of the call which signalled c1. Catch phrases are implemented with select-type, so the order of the condition names is significant. For a catch phrase to be well formed, all control paths through it must end with one of the following four quit forms:

```
(exit)
```

Causes the stack to unwind back through the enclosing enable form, which is exited with value NIL.

```
(resume form)
```

Returns from the call which did the signalling with the value of form as the value of that call, if it is resumable, otherwise generates an error.

```
(goto exit-key)
```

Causes the stack to unwind back to the enclosing enable form, where the exit phrase for exit-key is evaluated. The value of the last form in the phrase is the value of the enable. Exit phrases are implemented with SELECTQ, so lists of exit keys may precede ->.

(reject)

Causes the signal handling process to act as if the catch phrase had not been there at all.

There is a special exit key whose name is unwind, which has a special meaning. The exit clause for the unwind key will be evaluated whenever the stack unwinds upwards past this point.

There is another special exit key whose name is normal. Its exit clause will be executed in case of a normal return from the enclosed form, without affecting the value of the enable, which will still be the value of the enclosed form.

There is another special exit key whose name is always. always is a just a synonym for (normal unwind). Thus its exit clause will be executed if the stack ever unwinds past it and it will also be executed in case of a normal return from the enclosed form.

Calls to enable translate into calls to uwcp as follows, taking the above template for enable as the input:

```
(uwcp (f:l () form)
      (f:l (condition)
            (select-type condition
                          (c1 a1 a2 a3 ...)
                          . . .
                          (cn n1 n2 n3 ...)))
      (f:l (exit-key condition)
            (selectq exit-key
                      (k1 ea1 ea2 ...)
                      . . .
                      (kn en1 en2 ...)
                      (t nil)
                      nil))
      <if normal or always appeared then t else nil>)

(exit) =>
t
(resume form) =>
(list form)
(goto exit-key) =>
(quote exit-key)
(reject) =>
nil
```

Built in Condition classes

Here follow the definitions of class Condition and its built in sub-classes:

```
(defstruct Condition "an unspecialised condition")
```

[Note: It may turn out to be useful to include here instance variables can-resume and neednt-catch which are used instead of the arguments to raise-signal.]

```
(defstruct (Abort (:include Condition)) "an abort condition")
```

```
(defstruct (Throw (:include Condition)) "a condition for throwing
to a catch"
```

```
  tag value)
```

```
(defstruct (SystemError (:include Condition))
```

```
  "a condition resulting from a call to ERRORX"
```

```
  number message stack-pos breakchk printmsg)
```

```
(defstruct (Uncaught (:include Condition))
```

```
"a condition resulting from an uncaught signal" condition)
(defstruct (AboutToBreak (:include Condition))
  "a condition signalled by BREAK1 on entry")
```

Relation to CommonLisp

This proposal supports everything relevant in the CommonLisp book.

Interlisp to Common Lisp Concordia

Chapter 2 IRM (Datatypes)

Interlisp Form -----	Common Lisp Form -----
(DATATYPES --)	??
(TYPENAME datum)	?? (type-of datum) -- except for strings and arrays Note that the result types are different, however, and it is necessary to check for literals in the program, e.g., (IL:TYPENAME 123) => IL:SMALLP yet (IL:TYPE-OF 123) => LISP:FIXNUM. Also LISP:TYPE-OF is definitely non-portable except for structures.
(TYPENAMEP datum typename)	?? (typep datum typename) -- except for strings and arrays and the problem of non-portability of the type names.

2.1 Datatype Predicates

**For many of these, the translation should look at the value/effect context.
If used for effect only, no need to insert the (and (<test> x) x).**

(LITATOM x)	(symbolp x)
(SMALLP x)	(and (typep x 'fixnum) x)
(FIXP x)	(and (integerp x) x)
(FLOATP x)	(and (floatp x) x)
(NUMBERP x)	(and (numberp x) x) -- but includes more sorts of numbers
(ATOM x)	(and (or (symbolp x) (numberp x)) x) Often users wrote IL:ATOM when they meant the LISP:ATOM interpretation, however.
(LISTP x)	(and (consp x) x)
(NLISTP x)	(not (consp x)) or (atom x)
(STRINGP x)	(and (stringp x) x)
(ARRAYP x)	?? How are arrays to be represented? possibly (and (vectorp x) x) BVM - "ARRAYP probably translates as vectorp. Again, the real question is how ARRAY translates, at least when the origin is 1 (the default). You could translate to make-array with a size one greater than specified (wasting the zero element), but then you can't translate ARRAYSIZE as length. Sigh."
(HARRYP x)	(and (hash-table-p x) x) -- Not quite strong enough since Interlisp hash tables are more general than CL ones BVM -- "hash-table-p is probably good enough; it's the translation of HASH-ARRAY that will need more strength."

2.2 Datatype Equality

(EQ x y)	(eq x y)
(NEQ x y)	(not (eq x y))
(NULL x)	(null x)
(NOT x)	(not x)
(EQP x y)	(or (eq x y) (and (numberp x) (number y) (= x y))) Probably (= x y) will suffice in most cases BVM - "EQP also compares compiled code, but there's not much hope there."
(EQUAL x y)	?? Probably (equal x y) will suffice in most cases (differ on number comparisons and the CL version descends more datatypes)
(EQUALALL x y)	?? Probably (equalp x y) will suffice in most cases (differ on string comparisons)

2.3 Fast and Destructive Functions

2.4.1 Using Litatoms as Variables

(BOUNDP var)	(boundp var)
(SET x y)	(set x y) Note that this is a place where free variable references might "sneak" in and ruin the automatic "only declare special things that are used free." algorithm.
(SETQ x y)	(setq x y)
(SETQQ x y)	(setq 'y)
(GETTOPVAL var)	?? (symbol-value atom) -- no concept to top level value in CL BVM -- "I would translate GETTOPVAL and SETTOPVAL as symbol-value and set (not identity and setq), with a warning that they're wrong."
(SETTOPVAL var value)	?? (set var value)
(GETATOMVAL atom)	(symbol-value atom) BVM - "{GET SET}ATOMVAL are exactly symbol-value and set, with the implicit declaration, irrelevant to common lisp, that the variable is not dynamically bound."
(SETATOMVAL atom value)	(set var value)
2.4.3 Property Lists	
(GETPROP atom prop)	(get atom prop) BVM -- "GETPROP is really (and (symbolp atom) (get atom prop)), though you'll usually want it translated directly as get. Fortunately, PUTPROP does not suffer this brain damage."
(PUTPROP atom prop val)	(setf (get atom prop) val)

(ADDPROP atom prop new flg)	?? -- no direction translation (runtime?)
(REMPROP atom prop)	(remprop atom prop)
(REMPROPLIST atom prop)	?? -- no direction translation (runtime?)
(CHANGEPROP x prop1 prop2)	?? -- no direction translation (runtime?)
(PROPNAMES atom)	?? -- no direction translation (runtime?)
(DEFLIST l prop)	?? -- no direction translation (runtime?) LMM -- "Surely obsolete and not necessary."
(GETPROPLIST atom)	(symbol-plist atom)
(SETPROPLIST atom list)	(setf (symbol-plist atom) list)
(GETLIS x props)	?? (multiple-value-bind (prop value tail) (get-properties (symbol-plist x) props) tail)

2.4.4 Print Names

Most of this section is extremely problematic -- especially since, although functions may be written that capture much of the semantic content, they tend to much more cons'y than their Interlisp counterparts, hence will disrupt the performance profile of any translated program that exploits these features.

AD -- "I'd be tempted to leave most of the atom-building functions untranslated and flag them as something that the programmer should deal with himself. Except for very simple things, you will probably want to do whatever you were doing with atoms in some other way in CL."

BVM -- "I tend to agree with Andy. However, some of these are common enough that it might be worthwhile having approximate definitions in the library. E.g., write a version of MKATOM that does ordinary strings and numbers (the definition I wrote is close; slightly better might be one that did read-from-string while binding *readtable* to a table in which all the special characters have been given alphabetic syntax). Translate SUBATOM, PACK, PACK* as (MKATOM something), and then just flag all the MKATOMs uniformly.

It doesn't seem worth even trying for UNPACK, as any use is highly likely to need manual intervention anyway."

BVM - "Given that IL is so willing, and CL so unwilling, to coerce to strings, you might introduce a coerce-to-string macro to make some "translations" more palatable. If the translator knows how to evaluate it for constant forms (such as strings), so much the better."

(MKATOM x)	?? This is hard to capture exactly -- but here's one attempt (defun mkatom (arg) (if (numberp arg) arg (values (intern (typecase arg (symbol (string arg)) (string arg) (otherwise (prin1-to-string arg))))))))

and another (due to BVM)

```
(defun mkatom (arg)
  (let ((string (typecase arg
                    (symbol (string arg))
                    (string arg)
                    (otherwise (prin1-to-string arg)))))
    (multiple-value-bind (n end)
      (parse-integer string :junk-allowed t)
      (if (and n (= end (length string)))
          n
          (values (intern string)
                  )))))
  )
```

BVM -- "Of course, this still doesn't do (mkatom "123Q") or (mkatom "12E3") correctly (yecch)."

(SUBATOM x n m)

??

Again here's a (long and cons'y) attempt at translation

```
(defun subatom (x n &optional (m -1))
  (let* ((string (symbol-name x))
        (start (if (< n 0)
                    (+ (length string) n)
                    (1- n)))
        (end (if (< m 0)
                 (+ 1 (length string) m)
                 m)))
    (values (intern (subseq string start end)))
  ))
or
(MKATOM (subseq
         (string x)
         (if (< n 0)
             (+ (length string) n)
             (1- n))
         (if (< m 0)
             (+1 (length string) m)
             m))))
```

(PACK x)

??

But try

```
(defun pack (arglist)
  (let ((new-arglist
        (mapcar
         #'(lambda (arg)
              (typecase arg
                (symbol (string arg))
                (string arg)
                (otherwise
                 (prin1-to-string arg))))
         arglist)))
    (values (intern
              (apply #'concatenate 'string new-arglist)))
  ))
or
(MKATOM
 (apply
  #'concatenate
  'string
  (mapcar
   #'(lambda (arg)
        (typecase arg
          (symbol (string arg))
          (string arg)
          (otherwise
```

```

                                (prin1-to-string arg))))
                                x)))

(PACK* x1 x2 .. xn)
??
But try
(values
  (intern (apply #'concatenate
    'string
    (mapcar #'princ-to-string
      (list x1 x2 .. xn)))))
or
(MKATOM
  (apply
    #'concatenate
    'string
    (mapcar #'princ-to-string
      (list x1 x2 ".." xn)))))

(UNPACK x flg rdtbl)
??
But try
(defun unpack (arg)
  (let ((string (typecase arg
    (symbol (string arg))
    (string arg)
    (otherwise (prin1-to-string arg))))
    (result nil)
    (ch nil))
    (with-collection
      (dotimes (i (length string))
        (setq ch (char string i))
        (collect
          (or (digit-char-p ch)
              (intern
                (string (char string i))))
          )))
    ))
A more Common Lisp'y version is:
(defun unpack (arg)
  (let ((string
    (typecase arg
      (symbol (string arg))
      (string arg)
      (otherwise (prin1-to-string arg))))
    )
    (coerce string 'list)
    ))

(DUNPACK x scatchlist flg rdtbl)
"
BVM -- "I see no need for DCHCON and DUNPACK to translate
differently than CHCON and UNPACK, though the translations may
want to be flagged (but then, you need to flag them anyway)"

(NCHARS x flg rdtbl)
??
(defun nchars
  (arg &optional (flg nil)
    (*readtable* *readtable*))
  (length
    (if flg
      (prin1-to-string arg)
      (princ-to-string arg))))
)
If flg is nil, this can be optimized to cut down on the consing.

```

(NTHCHAR x n flg rdtbl)

```
??
(let ((*readtable* (or rdtbl *readtable*)))
  (if flg
    (values (intern
              (aref (prin1-to-string x) (1- n))))
            (values (intern
                      (aref (princ-to-string x) (1- n))))))
  )
Use of this function almost surely indicates
a stylistic problem -- single letter symbols being
used as character objects
```

(L-CASE x flg)

```
??
(typecase x
  (string (if flg
              (string-capitalize x) ;;not quite
              (string-downcase x)))
  (symbol
   (values (intern
             (if flg
               (string-capitalize x)
               (string-downcase x))))))
  (cons
   (mapcar #'L-CASE x)))
```

(U-CASE x)

```
??
(typecase x
  (string (string-upcase x))
  (symbol
   (values (intern
             (string-upcase x))))))
  (cons
   (mapcar #'U-CASE x)))
```

(GENSYM char)

```
(gensym (if char (string char)))
Although this translation may well in subtle ways
```

GENNUM

```
?? -- no corresponding var in Common Lisp
```

(MAPATOMS fn)

```
(do-all-symbols (dummy-var)
  (funcall fn dummy-var))
Although do-all-symbols is not guaranteed to touch each symbol only
once.
```

2.4.5 Character Code Functions

This section forces to face squarely the problem of Interlisp's penchant of representing character objects as symbols with single letter p-names.

(PACKC x)

```
??
(MKATOM (coerce
          (mapcar #'code-char x) 'string))
```

(CHCON x flg rdtbl)

```
??
(mapcar #'(lambda (sym)
             (char (symbol-name sym) 0))
  (UNPACK x flg rdtbl))
```

(DCHCON x scatchlist flg rdtbl)

```
"
```

(NTHCHARCODE x n flg rdtbl)

```
??
(char-code (char (symbol-name
                  (NTHCHAR x n flg rdtbl)) 0))
```

Not quite right since NTHCHARCODE may return NIL in some circumstances

(CHCON1 x)

??
(char-code (char (symbol-name x) 0))
BVM - "Your translation of CHCON1 oddly assumes the arg is a symbol, rather than an arbitrary printable object"

(CHARACTER n)

??
(MKATOM (string (code-char x)))

(FCHARACTER n)

"

(CHARCODE c)

??
(defun charcode-1 (c)
 (etypecase c
 (symbol
 (case symbol
 (CR 13)
 ...
 (otherwise
 (char-code (char (symbol-name c)
 0))))))
 (string
 (char-code (char c 0)))
 (cons
 (cons (charcode-1 (car c))
 (charcode-1 (cdr c)))))
)
(defmacro charcode (c)
 (charcode-1 c))

or in many cases
(char-code "some character object")
BVM - "CHARCODE should probably *always* translate as (char-code #\somechar), to facilitate conversion to the character idiom."

(SELCHARQ e c1 .. cn default)

(defmacro (e &rest args)
 (let ((default (car (last args)))
 (clauses (butlast args 1)))
 '(SELECTQ ,e
 ,@(mapcar
 #'(lambda (clause)
 '(', (CHARCODE (car clause)) .
 ,@(cdr clause))) clauses)
 ,default))
)

2.5 Lists

(CONS x y)

(cons x y)

(CAR x)

(car x)

(CDR x)

(cdr x)

(CAAR x)

(caar x)

.....

(CDDDR x)

.....
(cdddr x)

(RPLACD x y)
(FRPLACD x y)

(rplacd x y)
"

```
(RPLACA x y)
(FRPLACA x y)
```

```
(RPLNODE x a d)
(FRPLNODE x a d)
```

```
(RPLNODE2 x y)
(FRPLNODE2 x y)
```

```
(rplaca x y)
"
```

```
(rplacd (rplaca x a) d)
"
```

```
(rplacd (rplaca x (car y)) (cdr y))
"
```

2.5.1 Creating Lists

```
(MKLIST x)
```

```
(LIST x1 x2 .. xn)
```

```
(APPEND x1 x2 .. xn)
```

```
(APPEND x)
```

```
(NCONC x1 x2 .. xn)
```

```
(NCONC1 lst x)
```

```
(ATTACH x l)
```

```
(if (listp x) x (list x))
```

```
(list x1 x2 .. xn)
```

```
(append x1 x2 .. xn)
```

```
(copy-list x)
```

```
(nconc x1 x2 .. xn)
```

```
(nconc lst (list x))
```

```
?? -- probably obsolete
```

```
(defun attach (x l)
  (if (null l)
      (cons x l)
      (progn (setf (cdr l)
                    (cons (car l) (cdr l)))
              (rplaca l x)))
  )
```

2.5.2 Building Lists from Left to Right

```
(TCONC ptr x)
```

```
??
(defun tconc (ptr x)
  (let ((head (car ptr))
        (tail (cdr ptr)))
    (if (null head)
        (let ((result (list x)))
          (cons result result))
        (progn (setf (cdr ptr)
                      (cdr (rplacd tail (list x))))
                ptr)))
  )
```

```
(LCONC ptr x)
```

```
??
(defun lconc (ptr x)
  (let ((head (car ptr))
        (tail (cdr ptr)))
    (if (null head)
        (cons x (last x))
        (progn (setf (cdr ptr)
                      (last (rplacd tail x)))
                ptr)))
  )
```

```
(DOCOLLECT item lst)
```

```
??
```

```
(ENDCOLLECT item tail)
```

```
??
```

2.5.3 Copying Lists

(COPY x)	(copy-tree x)
(COPYALL x)	??
(HCOPYALL x)	??

Note from LMM:

"I've no trouble with your LIST translations. Are you sure CL has RPLACD? I thought you have to do (progn (SETF (CDR x) y) x).

I think the Interlisp character functions point up a kind of design choice that will come up again and again, in situations where the fundamental mechanism for getting something done in CL and IL differ.

I think a the translator might offer three choices:

(a) leave the functions alone (e.g., translate to IL:DCHCON and IL:MKATOM which are defined in a "compatibility" package). This gives code that works.

(b) produce "interim" translations, which have the same effect, e.g., as you've identified in your last message.

(c) attempt to produce "natural" Common Lisp style (examples follow.)

In the case of an Interlisp program that does PACKC, CHCON, DCHCON, in some cases the "native" CL program would use strings, and others, it would use symbols. (Interlisp programmers use symbols where CL programmers would use strings.)

Usually, the "native" translation of CL functions that deal in character codes is to translate them to deal in character objects. Sometimes, where an IL programmer deals with a list of characters or character codes, the CL programmer would leave it as a string; the problem was that IL didn't have the breadth of sequence functions and so IL programmers would frequently hack lists.

If IL:character/code/atom/list == CL: character/character/string/list

then

```
(PACKC x) => (coerce 'string x)
(CHCON x) => (coerce 'list x)
(CHCON x flg rdtbl) => (coerce 'list (write-to-string x))
```

Ignore & flag RDTBL argument

(DCHCON ...) => ignore & use CHCON

(NTHCHARCODE ...) => SCHAR

CHCON1 => SCHAR ... 1

CHARACTER => no-op
FCHARACTER

CHARCODE => use #\.

SELCHARQ => CASE with #\ as case elements"

2.5.4 Extracting Tails of Lists

(TAILP x y)	(tailp x y)
(NTH x n)	((lambda (list index) (nthcdr (1- index) list)) x n) BVM - "NTH returns tails, is one-based and has stupid behavior for n < 1"
(FNTH x n)	"
(LAST x)	(last x) Although the behavior of last on non-list is not defined
(FLAST x)	"
(NLEFT l n tail)	?? (defun nleft (l n tail) (if (and tail (tailp tail l)) (let* ((length (length l)) (sub-length (length tail)) (diff (- length sub-length n))) (if (>= diff 0) (dotimes (i diff l) (setq l (cdr l))))))) BVM - "The CL translation of the Interlisp definition of NLEFT would be substantially better than the one you give."
(LASTN l n)	?? is LASTN destructive?

2.5.5 Counting List Cells

EQLLENGTH, COUNTDOWN, and EQUALN are applicable to circular lists.

BVM - "I think worrying about il:equal is a waste of energy. The subtle difference between il:equal and cl:equal should be globally noted as a potential, albeit unlikely, source of incompatibility."

(LENGTH x)	(length x) Although length is only defined for true lists
(FLENGTH x)	"
(EQLLENGTH x n)	(eql (length x) n) Although would fail to return if x were circular BVM - "For its non-circularity consideration, a more faithful translation might be ((lambda (tail) (and (consp tail) (atom (cdr tail)))) (nthcdr (1- n) x)), but it is less obvious what is going on."
(COUNT x)	?? (defun count (x) (+ (length x) (let ((sum 0)) (dolist (a x) (if (consp a) (incf sum (count a)))))))
(COUNTDOWN x n)	??
(EQUALN x y depth)	?? (defun equaln (x y depth)

```

(cond ((eq depth 0) t)
      ((consp x)
       (and (consp y)
            (equaln (car x) (car y) (1- depth))))
      (t
       (and (not (consp y))
            (equal x y))))
)
NB equal not equivalent to IL:EQUAL

```

2.5.6 Logical Operators

(LDIFF x y) (ldiff x y)
 Except if y is not a tail of x. (LDIFF would signal an error in this case while ldiff would return (copy-list x))
 NB -- if y is nil (LDIFF x y) -> x
 BVM - "You might want to recognize the idiom (LDIFF lst (NLEFT lst n)) as (butlast lst n)"

(LDIFFERENCE x y) (set-difference x y :test #'equal)
 NB. equal is not equivalent to IL:EQUAL.

(INTERSECTION x y) (intersection x y :test #'equal)
 NB. equal is not equivalent to IL:EQUAL.
 Elimination of duplicate entries is not guaranteed by CLtL.
 BVM - "The fact that INTERSECTION advertises duplicate removal suggests that the conservative translation should be (remove-duplicates (intersection x y :test #'equal) :test #'equal)"
 BVM - "Recognize the common idiom (INTERSECTION x x) as (remove-duplicates x :test #'equal)"

(UNION x y) (union x y :test #'equal)
 NB. equal is not equivalent to IL:EQUAL.
 Again -- treatment of duplicate entries may not be identical.

2.5.7 Searching Lists

(MEMB x y) (member x y :test #'eq)
 Not defined if y is not a true list

(FMEMB x y) "

(MEMBER x y) (member x y :test #'equal)
 NB. equal is not equivalent to IL:EQUAL

(EQMEMB x y) (or (eq x y) (and (consp y) (member x y :test #'eq)))

2.5.8 Substitution Functions

(SUBST new old expr) (subst new old expr :test #'equal)
 NB. equal is not equivalent to IL:EQUAL.
 With this translation, if new is a consp, then new will NOT be copied on each substitution.

(DSUBST new old expr) (nsubst new old expr :test #'equal)
 Same caveat as for SUBST

(LSUBST new old expr) ??
 (sort of an nconc subst)

(SUBLIS alst expr flg)

```
(if (null flg)
  (sublis alst expr :test #'equal)
  ??)
```

NB. The usual equal caveat holds. If (eq flg t) then SUBLIS is required to cons an entirely new tree

(DSUBLIS alst expr flg)

```
(if (null flg)
  (nsublis alst expr :test #'equal)
  ??)
```

Same caveat as SUBLIS

(SUBPAIR old new expr flg)

```
??
```

ignoring flg and the strange behavior with respect to non-nil final tails of old, roughly equivalent to:

```
(sublis (mapcar #'cons old new) expr :test #'equal)
```

2.5.9 Association Lists and Property Lists

(ASSOC key alst)

```
(assoc key alst :test #'eq)
```

But not equivalent if alst is not a true list

BVM - "For ASSOC, DREMOVE, etc, I think you should use test eql instead of eq (in fact, isn't that the default in cl?). This is actually a more widespread and difficult problem with translating IL code--the hidden assumption that a substantial class of integers are immediate and hence testable by eq."

JOP - "I'm not sure I agree with the rational for using eql rather than eq in ASSOC (and friends), for the following reasons: (a) the keys for ASSOC (etc.) are usually symbols, and (b) Although not explicitly stated in CLtL -- it's probably fairly safe to assume that eq comparisons are valid for fixnums."

BVM - "I thought CLtL did explicitly state (p. 193) that it is NOT safe to assume that eq comparisons are valid for fixnums. This is not to say that I am aware of any implementations in which eql fixnums are not eq. However, there are certainly implementations in which the fixnum range is considerably smaller than ours, another subtle obstacle in porting. As for your point (a), it is my impression that people are fairly sloppy about whether assoc keys are symbols or not. Aside from all that, there's a reason that CLtL's default for assoc, etc, is eql. I think that translating il:assoc directly as cl:assoc is appropriate style; at worst, it performs slightly less efficiently than with an eq test, but you know it won't be wrong."

LMM - "The decision of EQ vs EQL in ASSOC is probably one of those decisions to made interactively at translation time..."

(FASSOC key alst)

```
"
```

(SASSOC key alst)

```
(assoc key alst :test #'equal)
```

Usual caveat about equal -- non NIL tails of alst

(PUTASSOC key val alst)

```
??
```

```
(defun putassoc (key val alst)
  (let ((entry (assoc key alst :test #'eq)))
    (if entry
      (setf (cdr entry) val)
      (progn (nconc alst (cons key val))
              val)))
  )
```

LMM - "I've found on more than one occasion that to do a "natural" translation, I wound up changing an ALIST into a property list, e.g., so I could use (SETF (GETF x y) z) instead of (PUTASSOC x y z)."

(LISTGET lst prop)

(getf lst prop)

(LISTPUT lst prop val)

(setf (getf lst prop) val)

(LISTGET1 lst prop)

(cdr (member prop lst :test #'eq))
NB. Order of evaluation not preserved

(LISTPUT1 lst prop val)

(setf (cdr (member prop lst :test #'eq)) val)
NB. Order of evaluation not preserved

2.5.10 Other List Functions

(REMOVE x l)

(remove x l :test #'equal)
NB. equal not equivalent to IL:EQUAL

(DREMOVE x l)

(delete x l :test #'eq)
Not guaranteed to return an eq list if the result is non-nil

(REVERSE l)

(reverse l)
Not equivalent if l is not a list

(DREVERSE l)

(nreverse l)
Same caveat as REVERSE

2.6 Strings

Some thorny issues arise here. Among them: (a.) Some Interlisp string functions will clearly not be applicable to all types of strings (eg GNC GLC), (b.) Some agreement must be attained between the allowable set of character objects and string-chars -- this may confine us to the 96 standard characters, excluding control characters, NS characters, etc. (c.) Reusing string headers is a fairly inoperative idea -- although doable if the reusable string is adjustable (d.) It may be nice to have some general technology for mapping from an index-origin-one indexing scheme to a index-origin-zero indexing scheme. This may include fairly global source modifications

(STREQUAL x y)

(string= x y)

(ALLOCSTRING n initchar)

(make-string n :initial-element (char-code initchar))

(ALLOCSTRING n initchar old)

??
(adjust-array old n :initial-element (char-code initchar))

(MKSTRING x flg rdtbl)

??
(defun mkstring
 (x &optional flg (rdtbl *readtable*))
 (let ((*readtable* rdtbl))
 (if (null flg)
 (typecase x
 (string x)
 (symbol (symbol-name x))
 (otherwise
 (princ-to-string x)))
 (prin1-to-string x)))
))

(SUBSTRING x n m)

??
(defun substring (x n &optional (m -1))
 (let* ((length (length x))

	<pre> (start (if (< n 0) (+ length n) (1- n))) (end (if (< m 0) (+1 length m) m))) (make-array (- end start) :element-type 'string-char :displaced-to x :displaced-index-offset start))) </pre>
(SUBSTRING x n m oldptr)	?? Might be able to do something if oldptr were an adjustable string
(GNC x)	?? Requires x to be adjustable <pre> (defun gnc (x) (let ((holder (make-array (length x) :element-type 'string-char :displaced-to x))) (prog1 (char x 0) (adjust-array x (1- (length x)) :displaced-to holder :displaced-index-offset 1)))) </pre> I'm not sure what would happen if the translation were simply <pre> (prog1 (char x 0) (adjust-array x (1- (length x)) :displaced-to x :displaced-index-offset 1)) </pre> Note that a character object is returned rather than a symbol
(GLC x)	?? x required to have a fill-pointer <pre> (vector-pop x) </pre> Note that a character object is returned rather than a symbol
(CONCAT x1 x2 .. xn)	?? <pre> (concatenate 'string (MKSTRING x1) (MKSTRING x2) .. (MKSTRING xn)) </pre>
(CONCAT)	<pre> (make-string 0) </pre>
(CONCATLIST x)	<pre> (apply #'CONCAT x) </pre>
(RPLSTRING x n y)	?? <pre> (defun rplstring (x n y) (let ((start (if (< n 0) (+ (length x) n) (1- n)))) (do ((i 0 (1+ i)) (limit (length y)) (j start (1+ j))) ((eql j limit) x) (setf (char x j) (char y i))))) </pre>
(RPLCHARCODE x n charcode)	?? <pre> (defun rplcharcode (x n charcode) (let ((index (if (< n 0) </pre>

```

                                (+ (length x) n)
                                (1- n))))
                                (Setf (char x index) (char-code charcode))
                                x)
                                )
(STRPOS pat string start)      ??
                                roughly
                                (1+ (search pat string :start1 (1- start)))

(STRPOS pat string start skip anchor tail)  ??

(STRPOS a str strat)           ??
                                roughly
                                (1+ (search (mapcar #'code-char a)
                                              str :start1 (1- start)))

(MAKEBITTABLE l neg a)         ??

```

2.7 Arrays

Suppose Interlisp arrays are represented by Common Lisp vectors, then two strategies present themselves for translation of the array facilities.

a.) Perform everywhere suitable subtractions -- but attempt global code simplification

b.) Use an addition vector cell and preserve origin-1 indexing

I will attempt to list translations appropriate for both strategies

NB. The index origin of a translated Interlisp vector will not be knowable at run-time.

BVM - "Since you can't tell by looking at a call to **ELT** or **SETA** whether the array is 0- or 1-origin, you can only use method "a" (subtract 1 from all indices) if the user is willing to globally declare "I never use zero-origin arrays". Note that when using method "b", you have to inflate the size of the vector by 1 even on calls to **ARRAY** with origin constant zero, unless you never care about **ARRAYSIZE** translating correctly."

Interlisp array element-types may be translated as follows

BIT	bit
BYTE	(unsigned-byte 8)
WORD	(unsigned-byte 16)
FLOATP	float
POINTER	t
DOUBLEPOINTER	??
XPOINTER	??
FLAG	(member t nil)
(BITS n)	(unsigned-byte n)
FIXP	(signed-byte 32) or t
SIGNEDWORD	(signed-byte 16)

One might imagine two functions -- **translate-type** and **inverse-translate-type** -- to move from Interlisp types to Common Lisp types and back again

```

(ARRAY size type init)
a.) (make-array size :element-type
               (translate-type type) :initial-element
               init)
b.) (make-array (1+ size) :element-type
               (translate-type type) :initial-element
               init)
Of course, if the array origin is explicitly specified as zero (0), then
translation a.) may always be employed

```

(ELT a n)	a.) (aref a (1- n)) b.) (aref a n)
(SETA a n v)	a.) (setf (aref a (1- n)) v) b.) (setf (aref a n) v)
(ARRAYTYP a)	(inverse-translate-type (array-element-type a))
(ARRAYSIZE a)	a.) (array-total-size a) b.) (1- (array-total-size a))
(ARRAYORIG a)	?? always 1? BVM - "I can't imagine any use for ARRAYORIG other than as an ORIGIN argument to ARRAY, where it will be fine to throw it out; any other use is untranslatable. Well, maybe some index checker would use it, in which case zero would be a safe translation."
(COPYARRAY a)	(copy-seq a)
(ARRAYP a)	(vectorp a)

2.7 Arrays Interlisp-10 Arrays

Probably, no functions in this section need be supported by the translator. I list those not mentioned elsewhere here for completeness.

(ELTD a n)	?? BVM - "ELTD and SETD can only be used on arrays of type DOUBLEPOINTER. You could tediously translate them as index (+ n / (1- (length a)) 2) 1), but it doesn't seem worth it. ARRAYBEG is blatantly untranslatable."
(SETD a n v)	??
(ARRAYBEG a)	??

2.8 Hash Arrays

Interlisp Harry's will most likely be represented as Common Lisp hash-tables even though Interlisp Harry's support options more extensive than those of their counterparts.

BVM - "All the hash functions need to watch out for harray = NIL for the bogus SYSHASHARRAY feature. Probably a global note in the translator's guide is sufficient; anyone who actually wrote a program depending on the feature deserves to lose."

(HARRAY len)	(make-hash-table :size len :test #'eq) or (make-hash-table :size len) BVM - "In the case of HARRAY, you need to watch out for (list (harray len)) and (cons (harray len) overflow) and turn them into (make-hash-table :size len :rehash-size overflow). HARRAY all by itself is strictly speaking untranslatable, because it implicitly has overflow action ERROR."
(HASHARRAY minkeys)	"
(HASHARRAY minkeys overflow)	(make-hash-table :size minkeys :rehash-size overflow) BVM - "I believe the overflow arg to HASHARRAY is a superset of

the allowable values to :rehash-size, though the commonly-used numeric values are compatible (hasharray also supports the values ERROR and arbitrary function)."

(HASHARRAY minkeys overflow nil nil nil rehash-threshold)	(make-hash-table :size minkeys :rehash-size overflow :rehash-threshold rehash-threshold)
(HASHARRAY minkeys overflow hashbitsfns equivfn nil rehash-threshold)	(make-hash-table :size minkeys :test (get-know-test-fn hashbitsfns equivfn) :rehash-size overflow :rehash-threshold rehash-threshold)
(HARRAYSIZE harray)	??
(CLRHASH harray)	(clrh hash harray)
(PUTHASH key val harray)	((lambda (x y z) (if y (setf (gethash x z) y) (remhash x z))) key val harray) BVM - "This is another good place for a simplifier, since val=nil is usually a constant. (Unfortunately, you can rarely get rid of the remhash arm--only if the value being stored is a non-nil constant.)"
(GETHASH key harray)	(values (gethash key harray)) BVM - "I hope the simplifier knows about eliminating (values &) in non-mv context."
(REHASH oldharray newharray)	??
(MAPHASH harray maphfn)	((lambda (x y) (maphash #'(lambda (key val) (funcall y val key)) x)) harray maphfn) BVM - "Yet another place where a simplifier with sufficient smarts about lambdas would make the translation more pleasant in the common case where the maphfn is a lambda expression. Alternatively, arrange for the translator to manually permute the arg list."
(DMPHASH harray1 ... harrayn)	(progn (print '(setq harray1 ,harray1)) ... (print '(setq harrayn ,harrayn)))
(HARRAYPROP harray prop)	?? BVM - "... the only instance of which we can translate is (HARRAYPROP a 'NUMKEYS) => (hash-table-count a)."
(HARRAYPROP harray prop nv)	??

2.9 Numeric and Arithmetic Functions

Since Common Lisp arithmetic functions are fully generic -- the type specific Interlisp arithmetic functions pose a problem. They can either be a.) Correctly translated with a substantial cost in performance and complexity or b.) incorrectly translated to their generic counterparts. I will give translation for both possibilities.

BVM - "I suspect most people will want the type-specific operations to translate generically (in code I've looked at, I virtually always do), even though this will occasionally cause subtle bugs."

There may be redundancy in the following section for completeness.

Many of the following predicates could be simplified in a test context.

(SMALLP x)	((lambda (x) (and (typep x 'fixnum) x)) x)

(FIXP x)	((lambda (x) (and (integerp x) x)) x)
(FLOATP x)	((lambda (x) (and (floatp x) x)) x)
(NUMBERP x)	((lambda (x) (and (numberp x) x)) x)
MIN.SMALLP MAX.SMALLP	most-negative-fixnum most-positive-fixnum BVM - "MAX.SMALLP is often used as a synonym for 2^{16-1} , so this translation should be flagged."
MIN.FIXP MAX.FIXP	?? ??
MIN.INTEGER MAX.INTEGER	?? ?? BVM - "MIN.INTEGER & MAX.INTEGER are obviously untranslatable, but I think we've even de-documented them."
(OVERFLOW flg)	??
(IPLUS x1 ... xn)	a.) (+ (truncate x1) ... (truncate xn)) b.) (+ x1 .. xn)
(PLUS x1 .. xn)	(+ x1 ... xn)
(FPLUS x1 ... xn)	a.) (+ (float x1) (float xn)) b.) (+ x1 .. xn)
(IMINUS x)	a.) (- (truncate x)) b.) (- x)
(MINUS x)	(- x)
(FMINUS x)	a.) (- (float x)) b.) (- x)
(IDIFFERENCE x y)	a.) (- (truncate x) (truncate y)) b.) (- x y)
(DIFFERENCE x y)	(- x y)
(FDIFFERENCE x y)	a.) (- (float x) (float y)) b.) (- x y)
(ITIMES x1 ... xn)	a.) (* (truncate x1) ... (truncate xn)) b.) (* x1 .. xn)
(TIMES x1 .. xn)	(* x1 ... xn)
(FTIMES x1 ... xn)	a.) (* (float x1) (float xn)) b.) (* x1 .. xn)
(IQUOTIENT x y)	(truncate x y)
(QUOTIENT x y)	a.) ?? b.) (/ x y) -- although this is likely to be wrong more often than not BVM - "QUOTIENT -- I think it should only be translated as / in the case where you know that one of its args is floatp; usage tends not to be very consistent. So (if (or (floatp x) (floatp y)) (/ x y) (truncate x y)) is better, if ugly."
(FQUOTIENT x y)	a.) (/ (float x) (float y)) b.) (/ x y) -- fairly safe

(IGREATERP x y)	a.) (> (truncate x) (truncate y)) b.) (> x y)
(GREATERP x y) (FGREATERP x y)	(> x y) ""
(ILESSP x y)	a.) (< (truncate x) (truncate y)) b.) (< x y)
(LESSP x y) (FLESSP x y)	(< x y) ""
(IGEQL x y)	a.) (>= (truncate x) (truncate y)) b.) (>= x y)
(GEQL x y) (FGEQL x y)	(>= x y) ""
(ILEQL x y)	a.) (<= (truncate x) (truncate y)) b.) (<= x y)
(LEQL x y) (FLEQL x y)	(<= x y) ""
(IEQL x y)	a.) (= (truncate x) (truncate y)) b.) (= x y)
(EQL x y)	(= x y) Strictly incorrect, but probably good enough
(FEQL x y)	""
(IREMAINDER x y)	a.) (rem (truncate x) (truncate y)) b.) (rem x y)
(REMAINDER x y)	(rem x y)
(FREMAINDER x y)	a.) (rem (float x) (float y)) b.) (rem x y)
(IMIN x1 ... xn)	a.) (min (truncate x1) ... (truncate xn)) close, but not correct since (IMIN 1.2 1.1) returns 1.2 b.) (min x1 ... xn) BVM - "For IMIN, it happens to be a bug that (IMIN 1.2 1.1) returns 1.2, so I wouldn't sweat it. Ditto IMAX and IABS."
(MIN x1 ... xn)	(min x1 ... xn)
(FMIN x1 ... xn)	a.) (min (float x1) ... (float xn)) b.) (min x1 ... xn)
(IMAX x1 ... xn)	a.) (max (truncate x1) ... (truncate xn)) close, but not correct since (IMAX 1.1 1.2) returns 1.1 b.) (max x1 ... xn)
(MAX x1 ... xn)	(max x1 ... xn)
(FMAX x1 ... xn)	a.) (max (float x1) ... (float xn)) b.) (min x1 ... xn)

(IABS x)	a.) (abs (truncate x)) Not quite right, since (IABS -0.1) returns -0.1 b.) (abs x)
(ABS x)	(abs x)
(FABS x)	a.) (abs (float x)) b.) (abs x)
(ADD1 x)	(1+ x)
(SUB1 x)	(1- x)
(ZEROP x)	(zerop x)
(MINUP x)	(minusp x)
(FIX x)	(truncate x)
(GCD x y)	(gcd x y)

2.9.2 Logical Arithmetic Functions

(LOGAND x1 .. xn)	(logand x1 .. xn)
(LOGOR x1 .. xn)	(logior x1 .. xn)
(LOGXOR x1 .. xn)	(logxor x1 .. xn)
(LSH x n)	(ash x n)
(RSH x n)	(ash x (- n))
(LLSH x n)	?? usually (ash x n) will suffice
(LRSH x n)	?? usually (ash x (- n)) will suffice
(INTEGERLENGTH n)	(if (< n 0) (1+ (integer-length n) (integer-length n))
(POWEROFTWOP n)	?? roughly (zerop (logand n (1- n)))
(EVENP x)	(evenp x)
(EVENP x y)	(zerop (mod x y))
(ODDP x)	(oddp x)
(ODDP x y)	(not (zerop (mod x y)))
(LOGNOT n)	(lognot n)
(BITTEST n mask)	(logtest n mask)
(BITCLEAR n mask)	(logandc2 n mask)
(BITSET n mask)	(logior n mask)

(MASK.1'S position size)	((lambda (x y) (ldb (byte y x) -1)) position size)
(MASK.0'S position size)	((lambda (x y) (dpb 0 (byte y x) -1)) position size)
(LOADBYTE n position size)	((lambda (x y z) (ldb (byte z y) x)) n position size)
(DEPOSITBYTE n position size byte)	((lambda (w x y z) (dpb z (byte y x) w)) n position size byte)
(ROT x n fieldsize)	??
(BYTE size position)	(byte size position)
(BYTESIZE bytespec)	(byte-size bytespec)
(BYTEPOSITION bytespec)	(byte-position bytespec)
(LDB bytespec val)	(ldb bytespec val)
(DPB n bytespec val)	(dpb n bytespec val)

2.9.3 Floating Point Arithmetic

MIN.FLOAT	most-negative-single-float
MAX.FLOAT	most-positive-single-float
(FLOAT x)	(float x)

2.9.5 Special Functions

(EXPT m n)	(expt m n)
(SQRT n)	(sqrt n)
(LOG x)	(log x)
(ANTILOG x)	(exp x)
(SIN x)	(sin (degrees-to-radians x)) where (defun degrees-to-radians (degrees) (* (/ pi 180) degrees))
(SIN x t)	(sin x)
(COS x)	(cos (degrees-to-radians x))
(COS x t)	(cos x)
(TAN x)	(tan (degrees-to-radians x))
(TAN x t)	(tan x)
(ARCSIN x)	(radians-to-degrees (asin (degrees-to-radians x))) where (defun radians-to-degrees (radians) (* (/ 180 pi) radians))
(ARCSIN x t)	(asin x)

(ARCCOS x)	(radians-to-degrees (acos (degrees-to-radians x)))
(ARCCOS x t)	(acos x)
(ARCTAN x)	(radians-to-degrees (atan (degrees-to-radians x))) NB: The IRM claims the range of ARCTAN is [0, pi] -- while in the most current loadup the range is [-pi/2, pi/2]. The later situation agrees with Common Lisp.
(ARCTAN x t)	(atan x)
(ARCTAN2 x y)	(radians-to-degrees (atan (degrees-to-radians x) (degrees-to-radians y)))
(ARCTAN2 x y t)	(atan x y)
(RAND lower upper)	((lambda (x y) (+ x (random (1+ (- y x))))) lower upper) NB. The 1+ to generate an inclusive upper bound is not correct if either x or y is of type float
(RAND)	(random (1+ most-positive-fixnum))
(RANDSET X)	(defun randset (x) (case x ((t) (setq *random-state* (make-random-state))) ((nil) *random-state*) (otherwise (setq *random-state* x))))

Daybreak Software Installation and Operation

This document is preliminary. It probably won't really be the way things work when the product is released.

The power switch and reset button are located on the front of the daybreak. There is no "ALT-B" button to specify alternate booting choices. Instead, the blank function keys along the top of the keyboard are used to specify the boot device.

When the daybreak is first turned on, the screen will be filled with a gray pattern with a solid white cursor in the upper left corner. Press one of the function keys according to the following table:

F1: Disk boot (Lisp)
F2: Floppy boot (doesn't work)
F3: Ethernet boot
F4: Alternate ether boot
F5: Diagnostic Disk boot
F6: Diagnostic Floppy boot
F7: Diagnostic Ether boot
F8: Reserved

Getting into Othello:

To get into othello, boot the machine then do an Alternate Etherboot-6 by pressing F4 followed by the number 6 (not F6 and not the keypad's 6). Sometime later, the cursor will change to 0900 and Othello will come up.

Installing Lisp:

The physical volume is partitioned into at least 3 volumes for Lisp.

uCode: This is where the Lisp microcode lives (about 150 pages)

Lisp: Where the sysout lives.

LispFiles: For those who want to use the local file system.

Lisp booting on the daybreak is similar to Mesa, so the microcode is stored as "Pilot Microcode" and there is a dummy "germ" file which is there just so the microcode can load something for the germ.

To bring up a new lisp on a fresh volume, do the following:

```
> Initial Microcode Fetch
Drive Name: RD0
File Name: <LispCore>Dove>DiskInitialDove.db
```

```
> Germ Fetch
Logical Volume Name: uCode
File Name: <LispCore>Dove>Dummy.Germ
```

```
> Lisp Microcode Fetch
Logical Volume Name: uCode
File Name: <LispCore>Dove>LispDove.db
```

Finally, fetching the Lisp sysout is more or less normal:

```
> Lisp Sysout Fetch
Logical Volume Name: Lisp
File Name: <LispCore>Next>Full.sysout
```

Shall I expand this lisp to fill the volume? Yes
 Shall I make this the default? Yes

To boot Lisp:

Press the reset button and then press F1

Lisp caveats:

If you need the local file system, do:

```
(DEFINEQ (MACHINETYPE ()(QUOTE DOVE)))  

then DIR {DSK} or (CREATEDSKDIRECTORY 'LISPPFILES)
```

MACHINETYPE will return DOVE later, but the file system's eventfn crashes on daybreaks and I'm afraid to touch the file system code to disable it any other way.

LOGOUT doesn't work right. If you want to save work and resume it later, do a SAVEVM. However, if you want to resume a frozen sysout, it must be expanded and the file system must be disabled (redefine MACHINETYPE to return DAYBREAK).

If the ethernet seems to go away, use (RESTART.ETHER) to start it up again.

Lots of programs like LAFITE create their status windows off the screen.

I'm not sure what will happen if you SYSOUT from a daybreak and load into a DLion/dorado.

The Implementation of Device-Independent Graphics Through Imagestreams

filed as {Eris}<LispCore>Internal>Doc>DIGguide.TEdit
written by Herb Jellinek on 2 November 1984
last revision on 26 February 1985

The Interlisp-D system does all image creation through a set of functions and data structures for **device-independent graphics**, known popularly as DIG. DIG is achieved through the use of a special flavor of stream, known as an imagestream.

An imagestream, by convention, is any stream that has its `IMAGEOPS` field (described in detail below) set to a vector of meaningful graphical operations. Using imagestreams, we can write programs that draw and print on an output stream without regard to the underlying device, be it window, disk, Dover, 8044 or Diablo printers. For example, the following have imagestreams backing them: windows, Press streams, Interpress streams, and Iris streams.

Imagestream structure

As indicated above, imagestreams use a field that no other stream does: `IMAGEOPS`. `IMAGEOPS` is an instance of the `IMAGEOPS` datatype, and contains a vector of the stream's graphical methods. The methods contained in the `IMAGEOPS` can make arbitrary use of the stream's `IMAGEDATA` field, which is provided for their use, and may contain any data needed.

`IMAGEOPS`

[Datatype]

The `IMAGEOPS` datatype has the following fields:

`IMAGETYPE` The name of image type. Monochrome display streams have an `IMAGETYPE` of `DISPLAY`; color display streams are identified as `(COLOR DISPLAY)`. The `IMAGETYPE` is informational, and can be set to anything the implementor chooses.

`IMFONTCREATE` The device name to pass to `FONTCREATE` when fonts are created for the stream.

The following fields are all stream methods, and are presented with their arguments, in the manner of a function definition. With the exception of `IMCLOSEFN`, each method that follows has a corresponding function that consists of the method's name with the "IM" prefix removed. All coordinates that refer to points in a display device's space are measured in the device's units. (The `IMSCALE` method provides access to a device's scale.)

(`IMCLOSEFN` *STREAM*)

What to do before stream is `CLOSEF`ed, e.g. flush buffers, write header or trailer information, etc.

(`IMDRAWLINE` *STREAM X1 Y1 X2 Y2 WIDTH OPERATION COLOR*)

Draws a line of width *WIDTH* from (*X1*, *Y1*) to (*X2*, *Y2*). (Dashing is currently handled at a higher level, and thus is not an argument).

(`IMDRAWCURVE` *STREAM KNOTS CLOSED BRUSH DASHING*)

Draws a curve through *KNOTS*.

(`IMDRAWCIRCLE` *STREAM CENTERX CENTERY RADIUS BRUSH DASHING*)

Draws a circle of radius *RADIUS* around (*CENTERX*, *CENTERY*).

(`IMDRAWELLIPSE` *STREAM CENTERX CENTERY SEMIMINORRADIUS SEMIMAJORRADIUS ORIENTATION BRUSH DASHING*)

Draws an ellipse around (*CENTERX*, *CENTERY*).

(*IMFILLCIRCLE STREAM CENTERX CENTERY RADIUS TEXTURE*)

Draws a circle filled with texture *TEXTURE* around (*CENTERX*, *CENTERY*).

(*IMBLTSHADE TEXTURE STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT OPERATION CLIPPINGREGION*)

The texture-source case of *BITBLT*. *DESTINATIONLEFT*, *DESTINATIONBOTTOM*, *WIDTH*, *HEIGHT*, and *CLIPPINGREGION* are measured in *STREAM*'s units. This method is invoked by the functions *BITBLT* and *BLTSHADE*.

(*IMBITBLT SOURCEBITMAP SOURCELEFT SOURCEBOTTOM STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION CLIPPEDSOURCELEFT CLIPPEDSOURCEBOTTOM*)

The bitmap-source cases of *BITBLT*. *SOURCELEFT*, *SOURCEBOTTOM*, *CLIPPEDSOURCELEFT*, *CLIPPEDSOURCEBOTTOM*, *WIDTH*, and *HEIGHT* are measured in pixels; *DESTINATIONLEFT*, *DESTINATIONBOTTOM*, and *CLIPPINGREGION* are in the units of the destination stream.

(*IMSCALEDBITBLT SOURCEBITMAP SOURCELEFT SOURCEBOTTOM STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION CLIPPEDSOURCELEFT CLIPPEDSOURCEBOTTOM SCALE*)

A scaled version of *IMBITBLT*. Each pixel in *SOURCEBITMAP* is replicated *SCALE* times in the X and Y directions; currently, *SCALE* must be an integer.

(*IMMOVETO STREAM X Y*)

Move to (*x*,*y*). This method is invoked by the functions *MOVETO* and *RELMOVETO*. If it is not supplied, a default method composed of calls to the *IMXPOSITION* and *IMYPOSITION* methods is used.

(*INTERPRI STREAM*)

(As yet unused.) Issue a newline. When implemented, this method will be invoked by the function *TERPRI*. It defaults to (`\OUTCHAR STREAM (CHARCODE EOL)`) .

(*IMSTRINGWIDTH STREAM STR RDTBL*)

Returns the width of string *STR* in *STREAM*'s units, using *STREAM*'s current font. If this method is not supplied, it defaults to calling `\STRINGWIDTH.GENERIC`.

(*IMCHARWIDTH STREAM CHARCODE*)

Returns the width of character *CHARCODE* in *STREAM*'s units. If this method is not supplied, it defaults to calling `\STRINGWIDTH.GENERIC`.

The following methods all have corresponding *DSPxx* functions (e.g. *IMYPOSITION* corresponds to *DSPYPOSITION*) that invoke them. They also have the property that they return their previous value; when called with *NIL* they return the old value without changing it.

(*IMXPOSITION STREAM XPOSITION*)

Sets new x-position on *STREAM*.

(*IMYPOSITION STREAM YPOSITION*)

Sets new y-position on *STREAM*.

(*IMFONT STREAM FONT*)

Sets *STREAM*'s font to be *FONT*.

(IMLEFTMARGIN *STREAM* *LEFTMARGIN*)

Sets *STREAM*'s left margin to be *LEFTMARGIN*. The left margin is defined as the x position set after newline.

(IMRIGHTMARGIN *STREAM* *RIGHTMARGIN*)

Sets *STREAM*'s right margin to be *RIGHTMARGIN*. The right margin is defined as the maximum x position at which characters will be printed; printing beyond it causes a newline.

(IMLINEFEED *STREAM* *DELTA*)

Sets *STREAM*'s linefeed distance (distance to move vertically after a newline) to be *DELTA*.

(IMNEWPAGE *STREAM*)

Causes a new page to be started; the position is set to (*DSLEFTMARGIN*, *DSPTOPMARGIN*). If not supplied, defaults to (`\OUTCHAR STREAM (CHARCODE ↑L)`).

(IMSCALE *STREAM* *SCALE*)

Returns the number of device points per screen point (a screen point being ~1/72 inch). In a later release of Interlisp-D the conversion factor will be specifiable. (I.e. right now *SCALE* is ignored.)

(IMTOPMARGIN *STREAM* *YPOSITION*)

Sets *STREAM*'s top margin (the y-position of the tops of characters that is set after newpage) to be *YPOSITION*.

(IMBOTTOMMARGIN *STREAM* *YPOSITION*)

Sets *STREAM*'s bottom margin (the y-position beyond which any printing causes a newpage) to be *YPOSITION*.

(IMSPACEFACTOR *STREAM* *FACTOR*)

Sets the amount by which to multiply the natural width of all following space characters on *STREAM*: used for justification of text. The default value is 1. For example, if the natural width of a space in *STREAM*'s current font is 12 units, and the spacefactor is set to 2, spaces will appear 24 units wide. The values returned by `STRINGWIDTH` and `CHARWIDTH` will also be affected.

(IMOPERATION *STREAM* *OPERATION*)

Sets the default `BITBLT OPERATION` argument. See the `DSPOPERATION` and `BITBLT` documentation for more information.

(IMBACKCOLOR *STREAM* *COLOR*)

Sets the background color of *STREAM*.

(IMCOLOR *STREAM* *COLOR*)

Sets the default color of *STREAM*.

In addition to the `IMAGEOPS`-borne methods described above, there are two other important methods, which are contained in the stream itself.

`STRMBOUTFN`

[Stream Method]

Function called by BOUT. You can install a STRMBOUTFN in a stream *STREAM* using the form (**replace** (STREAM STRMBOUTFN) **of** STREAM **with** (FUNCTION MYBOUTFN)).

OUTCHARFN

[Stream Method]

This is the function that is called to output a single byte. This is like STRMBOUTFN, except for being one level higher: it is intended for text output. Hence, this function should convert (CHARCODE EOL) into the stream's actual end of line sequence, and should adjust the stream's CHARPOSITION appropriately before invoking the stream's STRMBOUTFN (by calling BOUT) to actually put the character. Defaults to \FILEOUTCHARFN, which is definitely NOT what you want. OUTCHARFNs are installed using a form like (**replace** (STREAM OUTCHARFN) **of** STREAM **with** (FUNCTION MYOUTCHARFN)).

IMAGEDATA

[Record field]

Used to hold data pertaining to this type of imagestream; the content is completely up to the implementor. For Interpress devices, this is an instance of the datatype INTERPRESSDATA; for Press, PRESSDATA; for the display, \DISPLAYDATA.

Creating imagestreams

(OPENIMAGESTREAM *FILE* *IMAGETYPE* *OPTIONS*)

[function]

Opens and returns an output stream of type *IMAGETYPE* (PRESS, INTERPRESS, DISPLAY or other types) on a destination specified by *FILE*. *FILE* can name a file either on a normal storage device or on a printer device. In the latter case, the file is sent to the printer when the stream is closed. Because of the way that defaulted arguments are interpreted, OPENIMAGESTREAM provides a convenient and standard interface for interpreting user output-destination specifications.

If *IMAGETYPE* is NIL, the image type is inferred from the extension field of *FILE* and the EXTENSIONS properties in the list PRINTFILETYPES. Thus, a PRESS extension denotes a Press-format stream, while IP, IPR, and INTERPRESS indicate Interpress format. If *FILE* has no extension but is a file on the printer device {LPT}, then *IMAGETYPE* will be the type that the indicated printer can print. If *FILE* has no extension but is not on the printer device, then *IMAGETYPE* will default to the type accepted by the first printer on DEFAULTPRINTINGHOST.

FILE = NIL is equivalent to *FILE* = {LPT}. Names for printer files are of the form {LPT}PRINTERNAME.TYPE, where PRINTERNAME, TYPE, or both may be omitted. PRINTERNAME is the name of the particular printer to which the file will be transmitted on closing; it defaults to the first printer on DEFAULTPRINTINGHOST that can print *IMAGETYPE* files. As just described, the TYPE extension supplies the *IMAGETYPE* when it is defaulted. OPENIMAGESTREAM will generate an error if the specified printer does not accept the kind of file specified by *IMAGETYPE*.

Examples assuming IP: is an Interpress printer, P is a Press printer, and DEFAULTPRINTINGHOST is (IP: P):

(OPENIMAGESTREAM) returns an Interpress image stream on printer IP:

(OPENIMAGESTREAM NIL 'PRESS) returns a Press stream on P

(OPENIMAGESTREAM '{LPT}.INTERPRESS) returns an Interpress stream on IP:

(OPENIMAGESTREAM '{CORE}FOO.PRESS) returns a Press stream on the file {CORE}FOO.PRESS

For completeness and consistency, if *IMAGETYPE* is inferred to be DISPLAY, then the user is prompted for a window to open. The file name in this case will be used as the title of the window.

OPTIONS is a list in property list format that may be used to specify certain attributes of the image stream; not all attributes are meaningful or interpreted by all types of streams. Among the properties are:

REGION value is the region on the page (in stream scale units, 0,0 being the lower-left corner of the page) that text will fill up. It establishes the initial values for *DSPLEFTMARGIN*, *DSPRIGHTMARGIN*, *DSPBOTTOMMARGIN* and *DSPTOPMARGIN*.

FONTs value is a list of fonts that are expected to be used in the stream. Some streams (e.g. Interpress) are more efficient if the expected fonts are called out in advance, but this is not necessary. The first font in this list will be the initial font of the stream, otherwise the *DEFAULTFONT* for that image type will be used.

HEADING the heading to be placed automatically on each page, *NIL* means no heading.

IMAGESTREAMTYPES

[a-list]

Describes how to create a stream for a given image type. Contains *OPENSTREAM*, *FONTCREATE*, *FONTSAVAILABLE* methods. The main a-list is indexed by the image-stream type name (e.g., *DISPLAY*, *PRESS*, or *INTERPRESS*) to get another a-list that associates device-dependent functions with generic operation names.

Format of a single a-list entry:

```
(imagetype
 (OPENSTREAM function-to-open-the-stream)
 (FONTCREATE function-to-create-a-fontdescriptor)
 (FONTSAVAILABLE function-to-return-available-fonts))
```

For example, for Interpress, the a-list entry is:

```
(INTERPRESS
 (OPENSTREAM OPENIPSTREAM)
 (FONTCREATE \CREATEINTERPRESSFONT)
 (FONTSAVAILABLE \SEARCHINTERPRESSFONTS))
```

The *OPENSTREAM* function is called with arguments:

```
(openstreamfn FILE OPTIONS)
```

FILE is the file name as it was passed to *OPENIMAGESTREAM*, and *OPTIONS* is the *OPTIONS* property list passed to *OPENIMAGESTREAM*. The result must be a stream of the appropriate imagetype.

The *FONTCREATE* function is called with arguments:

```
(fontcreatefn FAMILY SIZE FACE ROTATION DEVICE)
```

FAMILY is the family name for the font, e.g. *MODERN*. *SIZE* is the body size of the font, in printer's points. *FACE* is a 3-element list describing the weight, slope, and expansion of the face desired, e.g. (*MEDIUM* *ITALIC* *EXPANDED*). *ROTATION* is how much the font is to be rotated from the normal orientation, in minutes of arc. For example, to print a landscape page, fonts would have rotation 5400 (=90 degrees). The function's result must be a *FONTDESCRIPTOR* with the fields filled in appropriately.

The *FONTSAVAILABLE* function is called with arguments:

```
(fontsaveavailablefn FAMILY SIZE FACE ROTATION DEVICE)
```

This function returns a list of all fonts agreeing with the *FAMILY*, *SIZE*, *FACE*, and *ROTATION* arguments; any of them may be wildcarded (i.e. equal to ' * ', which means "any"). Each

element of the list should be a 5-tuple of the form (FAMILY SIZE FACE ROTATION DEVICE).

Where the function looks is an implementation decision: the `fontsaveavailablefn` for the display device looks at `DISPLAYFONTDIRECTORIES`, the Interpress code looks on `INTERPRESSFONTDIRECTORIES`, and implementors of new devices should feel free to introduce new search path variables.

Imagestream predicates

(IMAGESTREAMP *X* *IMAGETYPE*) [function]

Returns *X* (possibly coerced to a stream) if it is an output image stream of type *IMAGETYPE* (or of any type if *IMAGETYPE* = `NIL`), otherwise `NIL`.

(IMAGESTREAMTYPE *STREAM*) [function]

Returns the image type of *STREAM*.

(IMAGESTREAMTYPEP *STREAM* *TYPE*) [function]

Returns `T` if *STREAM* is an imagestream of type *TYPE*.

Creating your own flavor of imagestream

In accomplishing a task as complex as building a new flavor of imagestream, no document can contain all of the answers, tricks, or shortcuts. There is no substitute for studying a working implementation in doing your own. Therefore, we recommend you look at the `FX80STREAM` package as an example of how to create a new imaging device. `FX80STREAM` is a DIG interface for the Epson FX-80 printer - a device simple enough to drive that its details will not obscure the fundamentals of how its imagestream works.

The dld-link SUBR opcode

This opcode links in a relocatable object file or a library file. The return value is an error code.

(SUBR DLD-LINK *path*)

Path is the path of the file to be loaded expressed as a string.

Return-Code is a FIXP cell into which DLD-LINK places a return value. See below for possible values.

Opcode is #o250

The dld-unlink-by-file SUBR opcode

This opcode unlinks in a relocatable object file or a library file. The return value is an error code.

(SUBR DLD-UNLINK-BY-FILE *string hard-flag*)

String is the name of the file to be unlinked.

Hard-flag is zero if the file shouldn't be unlinked if it is referenced by others. A nonzero value means that the file should be unlinked regardless of other references.

Return-Code is a FIXP cell. See below for possible values.

Opcode is #o251

The dld-unlink-by-symbol SUBR opcode

This opcode unlinks in a relocatable object file or a library file. The return value is an error code.

(SUBR DLD-UNLINK-BY-SYMBOL *string*)

String is the name of a symbol contained in the file to be unlinked.

Hard-flag is zero if the file shouldn't be unlinked if it is referenced by others. A nonzero value means that the file should be unlinked regardless of other references.

Return-Code is a FIXP cell. See below for possible values.

Opcode is #o252

The dld-get-symbol SUBR opcode

This opcode returns a pointer to a symbol in C-space.

(SUBR DLD-GET-SYMBOL *string*)

String is the name of a symbol.

Opcode is #0253

The dld-get-func SUBR opcode

This opcode returns a pointer to a function in C-space.

(SUBR DLD-GET-FUNC *string*)

String is the name of a function.

Opcode is #0254

The dld-function-executable-p SUBR opcode

This opcode returns 0 if the function contains undefined references.

(SUBR DLD-FUNCTION-EXECUTABLE-P *string*)

String is the name of a function.

Opcode is #o255

The dld-list-undefined-sym SUBR opcode

This opcode returns returns a list of the undefined symbols in the system.

(SUBR DLD-LIST-UNDEFINED-SYM)

Opcode is #o256

The dld-create-reference SUBR opcode

This opcode forces the system to load the library containing this symbol.

(SUBR DLD-CREATE-REFERENCE *string*)

```

0      Successful call and return.
-1     DLD-ENOFILE      cannot open file.
-2     DLD-EBADMAGIC   bad magic number.
-3     DLD-EBADHEADER  failiure reading header.
-4     DLD-ENOTEXT     premature end of file in text section.
-5     DLD-ENOSYMBOLS   premature end of file in symbol section.
-6     DLD-ENOSTRINGS  bad string table.
-7     DLD-ENOTEXTRELOC premature end of file in text relocation.
-8     DLD-ENODATA      premature end of file in data section.
-9     DLD-ENODATARELOC premature end of file in data relocation.
-10    DLD-EMULTDEFS    multiple definitions of symbol.
-11    DLD-EBADLIBRARY  malformed library archive.
-12    DLD-EBADCOMMON  common block not supported.
-13    DLD-EBADOBJECT  malformed input file.
-14    DLD-EBADRELOC   bad relocation info.
-15    DLD-ENOMEMORY    vmem exhausted.
-16    DLD-EUNDEFSYM   undefined symbol.

```

Fasl format change log.

Version 3 (10-Nov-86)

Added opcode 149, FASL-INTERLISP-SYMBOL pname(V).

Upward compatible with version 2.

INTRODUCTION

You are reading the Introduction. First comes a section on the low level 1108 floppy implementation. This includes discussion of FLOPPYIOCBs, DISKADDRESSES, and FLOPPYRESULTs. The low level floppy command functions are described in this section.

Next comes a section on Pilot floppy basics. This section describes file device \PFLOPPYFDEV and the functions installed on \PFLOPPYFDEV which are called by generic FILEIO functions. We describe the peculiar marker pages (PMPAGES)...

LOW LEVEL FLOPPY

This section describes the lowest level code of the 1108 FLOPPY implementation.

Miscellaneous Global Variables

\FLOPPY.CYLINDERS	(Variable)
\FLOPPY.TRACKSPERCYLINDER	(Variable)
\FLOPPY.SECTORSPERTRACK	(Variable)

The number of cylinders on the floppy, the number of tracks per cylinder on the floppy, and the number of sectors per track in the DATA part of the floppy. On the 1108, \FLOPPY.CYLINDERS=77, \FLOPPY.TRACKSPERCYLINDER=2, and \FLOPPY.SECTORSPERTRACK=15. Some of these values are different for the 1186.

\FLOPPYIOCB and 1108 Input Output Control Blocks

\FLOPPYIOCB	(Variable)

Points to the 1108 floppy input output control block. Lisp is run by the CP Central Processor and must communicate via the input output control block to the IOP Input Output Processor the various floppy commands that need to be executed.

It will appear that \FLOPPYIOCB is bound to CURSORBITMAP.EM. This is just a peculiarity of how the Lisp virtual memory works.

An input output control block looks like

FLOPPYIOCB	(Datatype)

```
(DATATYPE FLOPPYIOCB
  ((\BUFFERLOLOC WORD)
   (\BUFFERHILOC WORD)
   (NIL WORD)
   (SECTORLENGTHDIV2 WORD)
   (TROYORIBM BITS 12)
   (DENSITY BITS 4)
   (DISKADDRESS FIXP)
   (SECTORCOUNT WORD)
   (FLOPPYRESULT WORD)
   (SAMEPAGE FLAG)
   (COMMAND BITS 15)
   (SUBCOMMAND WORD)
   (SECTORLENGTHDIV4 BITS 8)
   (ENCODEDSECTORLENGTH BITS 8)
   (SECTORSPEPTRACK BITS 8)
   (GAP3 BITS 8)
   (NIL 3 WORD)))
```

The Lisp code sets up and the IOP reads all fields except for FLOPPYRESULT. The IOP sets and the Lisp code reads the FLOPPYRESULT.

\BUFFERHILOC and \BUFFERLOLOC are the high and low part of the pointer to the page that is being read from or being written to if any. We are not able to put (BUFFER POINTER) instead of (\BUFFERLOLOC WORD) and (\BUFFERHILOC WORD) because \BUFFERHILOC and \BUFFERLOLOC occur in the wrong order. The reason for this is that Mesa pointers are swapped and Lisp pointers are not.

SECTORLENGTHDIV2 is the length in bytes of the sectors being read or written divided by 2.

TROYORIBM is always IBM.

DENSITY is usually DOUBLE. DENSITY is SINGLE for track CYLINDER=0, HEAD=0 on Pilot floppies.

DISKADDRESS is a FIXP encoding a CYLINDER+HEAD+SECTOR combination. Each DISKADDRESS points to a page (aka sector) on the floppy. The particular format of DISKADDRESSES is described elsewhere in this document.

SECTORCOUNT is looked at by the IOP only when the COMMAND is to format the floppy. SECTORCOUNT indicates how many tracks to format.

FLOPPYRESULT is the status result word which is set by the IOP. The FLOPPYRESULT is a set of flags which can be usefully inspected with the FLOPPYRESULT BLOCKRECORD described elsewhere in this document.

SAMEPAGE is always NIL.

COMMAND can be any of the following constants

C.NOP	(Constant)
C.READSECTOR	(Constant)
C.WRITESECTOR	(Constant)
C.FORMATTRACK	(Constant)
C.RECALIBRATE	(Constant)
C.INITIALIZE	(Constant)

SUBCOMMAND is always SC.NOP.

GAP3, SECTORLENGTHDIV4, ENCODEDSECTORLENGTH, and SECTORS PER TRACK are obscure numbers that need to be set up for the IOP when COMMAND is to format.

1108 Floppy DISKADDRESSES

Locations of pages on a floppy are referred to by the hardware by cylinder+head+sector number combinations called DISKADDRESSES. DISKADDRESSES are represented by FIXPs and are accessed and created with the help of the record declaration

DISKADDRESS (Accessfns)

```
(ACCESSFNS DISKADDRESS
  ((CYLINDER (LRSH DATUM 16))
   (HEAD (LRSH (LOGAND DATUM 65535) 8))
   (SECTOR (LOGAND DATUM 255)))
  (CREATE (IPLUS (LLSH CYLINDER 16)
              (LLSH HEAD 8)
              SECTOR)))
```

CYLINDER can be between 0 and \FLOPPY.CYLINDERS.

HEAD can be 0 or 1 to indicate which side of the floppy should be read.

The part of the floppy pointed to by a CYLINDER+HEAD combination is known as a track. Each track contains a certain amount of redundant operation for self checking purposes conducted by the IOP assembly language code plus actual data stored in pages called sectors.

SECTOR can be between 1 and 15 for tracks that are formatted IBM double density 512 bytes per sector. The upper limit for SECTOR is determined according to the formatting of the track as given by the following table

Format	Number of Sectors
IBMS128	26
IBMS256	15
IBMS512	8
IBMS1024	4
IBMD128	36
IBMD256	26
IBMD512	15
IBMD1024	8

The table above can be found coded into the function \FLOPPY.SECTORSPERTRACK.

Pilot converts between Pilot page numbers and DISKADDRESSES by using the functions \PFLOPPY.PAGENOTODISKADDRESS and \PFLOPPY.DISKADDRESSTOPAGENO.

\FLOPPYRESULT and 1108 Result Words

\FLOPPYRESULT

(Variable)

Points to the 1108 floppy result word offsetted into the \FLOPPYIOCB input output control block. Lisp is run by the CP Central Processor and must communicate via the input output control block to the IOP Input Output Processor the various floppy commands that need to be executed. After a command is executed, the Lisp code looks at the status word to see if any error conditions were signalled indicating a failed operation.

It will appear that \FLOPPYRESULT is bound to MOUSEX.EM. This is just a peculiarity of how the Lisp virtual memory works.

An 1108 result word looks like

FLOPPYRESULT (Blockrecord)

```
(BLOCKRECORD FLOPPYRESULT
  ( (DOOROPENED FLAG)
    (MPERROR FLAG)
    (TWO SIDED FLAG)
    (DISKID FLAG)
    (ERROR FLAG)
    (NIL FLAG)
    (RECALIBRATE ERROR FLAG)
    (DATA LOST FLAG)
    (NOT READY FLAG)
    (WRITE PROTECT FLAG)
    (DELETED DATA FLAG)
    (RECORD NOT FOUND FLAG)
    (CRC ERROR FLAG)
    (TRACK 0 FLAG)
    (NIL FLAG)
    (BUSY FLAG) ) )
```

It should be pointed out that some of the error bits in the \FLOPPYRESULT are a bit noisy. Certain errors turn out to be transients that can be overcome simply by reissuing the last command. The bits MPERROR and CRCERROR are notably noisy. The Lisp code takes the first occurrence of certain errors as an indication that perhaps the floppy drive has lost its way and needs to be recalibrated. In that case, a recalibrate command is issued and then the command that failed is tried again. After sufficient retrying of a command that fails (the amount of retrying depending on the kind of command and the particular settings of the error bits), the bad news is announced to the user. Most of this retrying and so on has been determined empirically and is coded into the function \FLOPPY.RUN.

DOOROPENED is T if the floppy drive door has opened since the last floppy command. This is an error condition which has to be cleared. (When DOOROPENED gets set, ERROR also gets set.) To clear this bit, an INITIALIZE command must be issued.

MPERROR is T if the IOP floppy handler tried to crash the machine. We suspect this condition gets set from time to time because of bugs in the IOP assembly language code written by OSD. The MPERROR feature is a patch on top of the OSD assembly language code installed by Mitch Lichtenberg. Instead of crashing the machine the flag MPERROR now gets set and Lisp deals with that. It has turned out to be the case that reissuing the last command to the IOP without change invariably works. When MPERROR is set to T, the remainder of the FLOPPYRESULT word is treated as an error code instead of the flags DOOROPENED, TWO SIDED, DISKID, ... BUSY. The MPCODEs stored in the remainder of the FLOPPYRESULT word can be

```
580 Domino NoValidCommand Error
581 Domino UnImplFloppyCmd Error
582 Domino InvalidEscapeCmd Error
```



```

583 Domino CommandTrack Error
584 Domino TrackToBig Error
585 Domino BadDmaChannel Error
586 Domino NoDmaEndCount1 Error
587 Domino NoDmaEndCount2 Error
597 Domino Error In NOOP Patch
598 Domino Error in Reset Patch

```

TWOSIDED is T if the floppy is two sided.

DISKID = T if the floppy drive is Schogart Associated model SA850 and NIL if the floppy drive is Schogart Associated model SA800.

ERROR = T if there was an error in trying to perform the last command issued to the IOP. To clear this bit, an INITIALIZE command must be issued.

RECALIBRATEERROR = T if there was an error while recalibrating I suppose. I don't think I've ever seen this flag set in practice except perhaps if you issue a recalibrate command when there is no floppy in the floppy drive. The Lisp code in that situation would tend to error out on a command preceding any recalibration.

DATALOST = T problem reading or writing. Maybe the track of the floppy which is being read from or written into isn't formatted in the way that was expected.

NOTREADY = T implies the IOP is not ready I guess. I'm not sure I've ever seen this.

WRITEPROTECT = T means the floppy is writeprotected. Any attempt to write on the floppy will fail so the Lisp code checks whether this flag is on and signals an error if need be before attempting to open an output stream or format a floppy.

DELETEDDATA = I have no idea.

RECORDNOTFOUND = T problem reading or writing. Maybe the track of the floppy which is being read from or written into isn't formatted in the way that was expected.

CRCERROR = Cyclic Redundancy Check. There is redundant information on the floppy which serves as a check to the IOP assembly language code that the floppy drive head is where it is supposed to be. This error bit is a bit noisy and spurious CRCERRORs can be overcome by reissuing the last command to the IOP. If after several retries the CRCERROR has not gone away, then the CRCERROR is treated as real and the user is hit with the bad news. A hard CRCERROR is caused by the track of the floppy which is being read from or written into not being formatted in the way that was expected.

TRACK0 = T after a recalibrate command. Says that the recalibrate successfully found CYLINDER=0 I guess. Not really paid attention to by the Lisp code.

BUSY = T implies the IOP is busy I guess. I'm not sure if this would mean that the IOP is still preprocessing the most recently issued command or not. The official way to wait for the IOP to finish the most recently issued command is with the loop

```

(while (NOT (ZEROP (fetch (IOPAGE DLFLOPPYCMD) of
\IOPAGE))) do (BLOCK))

```

Low Level 1108 Floppy Command Functions

(FLOPPY.RUN FLOPPYIOCB NOERROR)

This function makes the IOP really go.

FLOPPYIOCB should already be prepared by other functions like \FLOPPY.NOP, \FLOPPY.READSECTOR, \FLOPPY.WRITESECTOR, \FLOPPY.FORMATTRACKS, \FLOPPY.RECALIBRATE, or \FLOPPY.INITIALIZE described below.

First, if there is a buffer involved (the command is to read or to write), the buffer is locked down by calling the function \FLOPPY.LOCK.BUFFER.

Next, the FLOPPYIOCB passed in is \BLT'ed on to \FLOPPYIOCB. \FLOPPYIOCB points to the beginning of the 16 words in real memory that the IOP will look at and interpret as an input output control block to be processed.

Next, the IOP is notified that there is an input output control block in need of processing via

(replace (IOPAGE DLFLOPPYCMD) of \IOPAGE with \FLOPPYIOCBADDR)

The IOP periodically looks at the \IOPAGE for things to do and acts when it sees a nonzero DLFLOPPYCMD field in the \IOPAGE. After replacing the DLFLOPPYCMD field in the \IOPAGE, the Lisp code must wait until the IOP finishes via the loop

(while (NOT (ZEROP (fetch (IOPAGE DLFLOPPYCMD) of \IOPAGE))) do (BLOCK))

After this loop finishes, \FLOPPY.RUN looks at the \FLOPPYRESULT result word to see if any error flags have been set by the IOP. Supposing things have gone well, \FLOPPY.UNLOCK.BUFFER is called to unlock the buffer (if any) pointed to by the FLOPPYIOCB and \FLOPPY.RUN returns T indicating success.

If error bits in the \FLOPPYRESULT result word have been set, then \FLOPPY.RUN may try to recover in certain ways. This may involve reissuing a command and/or some intervening recalibration commands.

If an error persists, then \FLOPPY.RUN returns NIL if NOERROR = T and otherwise a break and an error message happen to the user.

(FLOPPY.LOCK.BUFFER FLOPPYIOCB)

Calls \LOCKPAGES on the buffer pointed to by the FLOPPYIOCB \BUFFERLOLOC and \BUFFERHILOC fields. It is necessary that the buffer be locked down before a command can be issued to the IOP. This function also does \GETBASEing and \PUTBASEing into the BUFFER to make the BUFFER look dirty to the IOP. This is known as "touching pages". A reasonable person would think that this touching ought not to be forced on the Lisp floppy handler, but that's just the way it is. If the Lisp floppy handler doesn't touch the buffer, the IOP will sometimes cause a fatal 510 crash.

\FLOPPY.LOCK.BUFFER gets called by \FLOPPY.RUN before a command to the IOP is allowed to begin executing.

(\FLOPPY.UNLOCK.BUFFER FLOPPYIOCB)

Calls \UNLOCKPAGES on the buffer pointed to by the FLOPPYIOCB. \FLOPPY.UNLOCK.BUFFER gets called by \FLOPPY.RUN after a command to the IOP has finished executing.

(\FLOPPY.NOP NOERROR)

Calls \FLOPPY.RUN to perform a NOP command. This command can be used to get the state of the DOOROPENED and WRITEPROTECT flags of the \FLOPPYRESULT without doing actual operations.

(\FLOPPY.READSECTOR FLOPPYIOCB DISKADDRESS PAGE NOERROR)

Calls \FLOPPY.RUN to perform a read. PAGE should be a virtual memory page VMEMPAGEP. Typically PAGE is a virtual memory page that has worked its way down from FILEIO functions to \FLOPPY.READSECTOR via the functions \PFLOPPY.READPAGES and \PFLOPPY.READPAGENO. FLOPPYIOCB will normally be \FLOPPY.IBMD512.FLOPPYIOCB. Data at the page of the floppy located at DISKADDRESS is read into PAGE.

(\FLOPPY.WRITESECTOR FLOPPYIOCB DISKADDRESS PAGE NOERROR)

Calls \FLOPPY.RUN to perform a write. PAGE should be a virtual memory page VMEMPAGEP. Typically PAGE is a virtual memory page that has worked its way down from FILEIO functions to \FLOPPY.WRITESECTOR via the functions \PFLOPPY.WRITEPAGES and \PFLOPPY.WRITEPAGENO. FLOPPYIOCB will normally be \FLOPPY.IBMD512.FLOPPYIOCB. The contents of PAGE are written into the page of the floppy located at DISKADDRESS.

(\FLOPPY.FORMATTRACKS FLOPPYIOCB DISKADDRESS KOUNT NOERROR)

Calls \FLOPPY.RUN to perform formatting. This function will format KOUNT tracks on the

(fetch (DISKADDRESS HEAD) of DISKADDRESS)

side of the floppy beginning with cylinder

(fetch (DISKADDRESS CYLINDER) of DISKADDRESS)

The quantity (fetch (DISKADDRESS SECTOR) of DISKADDRESS) is ignored for purposes of formatting.

KOUNT is spelled with a K just because COUNT is a CLISP word and CLISP gets in your way if you spell the word with a C.

The kind of formatting that takes place depends on the kind of FLOPPYIOCB. If FLOPPYIOCB is \FLOPPY.IBMD512.FLOPPYIOCB, then tracks are formatted IBM double density 512 bytes per sector. If FLOPPYIOCB is \FLOPPY.IBMD256.FLOPPYIOCB, then tracks are formatted IBM

double density 256 bytes per sector. If FLOPPYIOCB is \FLOPPY.IBMS128.FLOPPYIOCB, then tracks are formatted IBM single density 128 bytes per sector.

Pilot floppies are formatted this way:

CYLINDER=0, HEAD=0 => IBMS128 format

CYLINDER=0, HEAD=1 => IBMD256 format

CYLINDER >0, HEAD=0 or 1 => IBMD512 format

(FLOPPY.RECALIBRATE NOERROR)

This function is called to ask the floppy drive hardware to recalibrate itself. The floppy drive head positions itself over tracks on a floppy with the aid of a stepping motor which steps in from CYLINDER=0 which is specially recognizable. The stepping motor can slip some and it is occasionally necessary to ask the floppy drive hardware to recalibrate itself which means refind CYLINDER=0. CYLINDER = 0 is kind of a light pole on a very dark street for the floppy hardware. Once the hardware is back in the light of CYLINDER = 0, it knows where it is again.

\FLOPPY.RECALIBRATE is sometimes called by the Lisp implementation after certain errors are detected while performing normal commands like reading and writing. The assumption in those situations is that the stepping motor has slipped some and that recalibrating will make things well enough again that the command which failed will succeed if it is reissued after the recalibration. (Of course, after enough times of trying this strategy, the error just has to be announced to the user.)

\FLOPPY.RECALIBRATE is also called at the beginning of certain major operations. For example, \FLOPPY.RECALIBRATE is called several times by the formatting function \PFLOPPY.FORMAT. After all, if we're going to format a floppy which involves writing on to the floppy, we want to do the best job we can.

(FLOPPY.INITIALIZE NOERROR)

\FLOPPY.INITIALIZE initializes the IOP assembly language code floppy handler which Lisp must talk with. This function has to be the first function that is called when FLOPPY is started up. Thus, \FLOPPY.EVENTFN calls \FLOPPY.INITIALIZE after coming back from a LOGOUT, SYSOUT, MAKESYS, or SAVEVM.

\FLOPPY.INITIALIZE also has to be called to clear error conditions as they arise. This might happen, for example, if the user tries to read from the floppy drive but the floppy drive door is open or there is no floppy. This would set the DOOROPENED error bit in the \FLOPPYRESULT word. To clear these error conditions like DOOROPENED, \FLOPPY.INITIALIZE needs to be called.

PILOT FLOPPY

This section describes the connection between FLOPPY and FILEIO. FILEIO is the system source file that makes device

independent part of operations like OPENSTREAM, READ, PRINT, CLOSEF, RENAMEFILE, DELFILE, and DIRECTORY possible. FILEIO ultimately winds up calling functions stored in the fields of file device records.

\PFLOPPYFDEV

\PFLOPPYFDEV

(Variable)

The major FDEV record for floppy is bound to \PFLOPPYFDEV. At the time of this writing, \PFLOPPYFDEV looks like the following image

{FDEV}#66,3400 Inspector

```

-----
DEVICENAME          FLOPPY
RESETABLE           T
RANDOMACCESSP        T
NODIRECTORIES        T
PAGEMAPPED           T
FDBINABLE            T
FDBOUTABLE           T
FDEXTENDABLE         T
BUFFERED             T
REMOTEP              NIL
SUBDIRECTORIES        NIL
CLOSEFILE            \PFLOPPY.CLOSEFILE
DELETEFILE           \PFLOPPY.DELETEFILE
DIRECTORYNAMEP        TRUE
EVENTFN              \FLOPPY.EVENTFN
GENERATEFILES         \PFLOPPY.GENERATEFILES

GETFILEINFO          \PFLOPPY.GETFILEINFO
GETFILENAME           \PFLOPPY.GETFILENAME
HOSTNAMEP            \FLOPPY.HOSTNAMEP
OPENFILE             \PFLOPPY.OPENFILE
READPAGES            \PFLOPPY.READPAGES
REOPENFILE           \PFLOPPY.OPENFILE
SETFILEINFO           \PFLOPPY.SETFILEINFO
TRUNCATEFILE          \PFLOPPY.TRUNCATEFILE
WRITEPAGES           \PFLOPPY.WRITEPAGES
BIN                  \BUFFERED.BIN
BOUT                 \BUFFERED.BOUT
PEEKBIN              \BUFFERED.PEEKBIN
READP                \PAGEDREADP
BACKFILEPTR          \PAGEDBACKFILEPTR
DEVICEINFO            {PINFO}#65,4764
FORCEOUTPUT          \PAGED.FORCEOUTPUT
LASTC                NIL
SETFILEPTR           \PAGEDSETFILEPTR
GETFILEPTR           \PAGEDGETFILEPTR
GETEOFPTR            \PAGEDGETEOFPTR
EOFP                 \PAGEDEOFP
BLOCKIN              \BUFFERED.BINS
BLOCKOUT             \BUFFERED.BOUTS
RENAMEFILE           \PFLOPPY.RENAMEFILE
RELEASEBUFFER         NIL
GETNEXTBUFFER        \PAGED.GETNEXTBUFFER
SETEOFPTR            \PAGED.SETEOFPTR
FREEPAGECOUNT        NIL
MAKEDIRECTORY         NIL
WINDOWOPS            NIL
WINDOWDATA           NIL

```

CHECKFILENAME	NIL
HOSTALIVEP	NIL
OPENP	\GENERIC.OPENP
OPENFILELST	NIL
REGISTERFILE	\ADD-OPEN-STREAM

Pilot Floppy FDEV Functions

The following FLOPPY functions can be called by FILEIO

(\PFLOPPY.OPENFILE FILE ACCESS RECOG OTHERINFO FDEV OLDSTREAM)

Gets called when FILEIO opens a stream for input or output. If input, then \PFLOPPY.OPENOLDFILE eventually gets called. If output, then \PFLOPPY.OPENNEWFILE eventually gets called.

\PFLOPPY.OPENFILE returns a stream datatype. The DEVICE of the stream will be \PFLOPPYFDEV. Two other fields on the stream, F1 and F2, point to the allocation record (PFALLOC) and leader page (PLPAGE) for the stream. The PFALLOC and PLPAGE can be conveniently accessed by using the FLOPPYSTREAM ACCESSFNS. When other floppy functions are passed the stream to work with, FLOPPY looks at the arguments passed, the fields of the stream, and the fields of the PFALLOC and PLPAGE to determine how to act.

\PFLOPPY.DIR.GET is called to search for the allocation record of an old FILE. \PFLOPPY.DIR.PUT is called to store the allocation record of a newly created file.

(\PFLOPPY.READPAGES STREAM FIRSTPAGE# BUFFERS)

Reads sectors off floppy into virtual memory pages BUFFERS. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \PFLOPPY.READPAGES therefore fills BUFFERS with data read from the floppy beginning with the FIRSTPAGE# of STREAM.

FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file. FILEIO page numbers are converted into Pilot page numbers lying somewhere between \PFLOPPYFIRSTDATAPAGE and \PFLOPPYLASTDATAPAGE. Pilot page numbers are in turn converted into DISKADDRESSES which record head, sector, and cylinder of a page on a floppy.

\PFLOPPY.READPAGES calls \FLOPPY.READPAGE which computes a Pilot page number from the stream's PFALLOC and the FILEIO page number passed in as an argument.

\FLOPPY.READPAGE calls \PFLOPPY.PAGENOTODISKADDRESS to convert the Pilot page number into a DISKADDRESS and then calls \FLOPPY.READSECTOR to read the sector at the computed disk address.

(\PFLOPPY.WRITEPAGES STREAM FIRSTPAGE# BUFFERS)

Other than that writing is taking place instead of reading, \PFLOPPY.WRITEPAGES is similar to \PFLOPPY.READPAGES.

Writes contents of virtual memory pages BUFFERS on to sectors of floppyfills BUFFERS with data . FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \PFLOPPY.WRITEPAGES therefore writes to the floppy beginning with the location corresponding to the FIRSTPAGE# of STREAM.

FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file. FILEIO page numbers are converted into Pilot page numbers lying somewhere between \PFLOPPY.FIRSTDATAPAGE and \PFLOPPY.LASTDATAPAGE. Pilot page numbers are in turn converted into DISKADDRESSES which record head, sector, and cylinder of a page on a floppy.

\PFLOPPY.WRITEPAGES calls \FLOPPY.WRITEPAGE which computes a Pilot page number from the stream's PFALLOC and the FILEIO page number passed in as an argument.

\FLOPPY.WRITEPAGE calls \PFLOPPY.PAGENOTODISKADDRESS to convert the Pilot page number into a DISKADDRESS and then calls \FLOPPY.WRITESECTOR to write the sector at the computed disk address.

(\PFLOPPY.TRUNCATEFILE FILE LASTPAGE LASTOFFSET)

Called just before closing an output file. The effect as far as FLOPPY is concerned is to take the allocation record (PFALLOC) for FILE and split the allocation record into two records if necessary. The first PFALLOC created this way is just big enough to store the truncated file. The second PFALLOC created this way becomes a free block. Since FLOPPY does not know how big an output file will turn out to be when it is first opened, FLOPPY must go through the process of allocating a reasonable size block, growing the block on occasion when \PFLOPPY.WRITEPAGES is about to cause the block to overflow, and finally truncate--i.e. split--the block into actual file and free block when \PFLOPPY.TRUNCATEFILE gets called.

\PFLOPPY.TRUNCATEFILE calls \PFLOPPY.TRUNCATE which does the actual splitting of a PFALLOC . \PFLOPPY.TRUNCATE splits a PFALLOC into two PFALLOCs equal to file and free block. The list of PFALLOCs cached on the floppy file device is updated and sufficiently many \PFLOPPY.WRITEPAGENOs of new or updated Pilot marker pages are written out to the floppy.

(\PFLOPPY.CLOSEFILE FILE)

Called to close a file. Writes out the leader page (PLPAGE) of FILE and the two marker pages that go around the leader page+ file.

(\PFLOPPY.DELETEFILE FILE FDEV)

Called to delete a file. \PFLOPPY.DIR.GET gets called to search for the allocation record for FILE. \PFLOPPY.DIR.REMOVE gets called to remove the allocation record from the cached directory alist. \PFLOPPY.DEALLOCATE gets called to turn the removed PFALLOC into a free block. The two marker pages surrounding FILE are updated to indicate that there is a free block between the marker pages and then they are written out. (Otherwise, at least until the free block gets used for other purposes, the contents of the file are still there, but have become inaccessible.)

**(\PFLOPPY.RENAMEFILE OLDDEVICE OLDFILE NEWDEVICE
NEWFILE OLDRECOG NEWRECOG)**

Called to rename a file. \PFLOPPY.DIR.GET gets called to search for the allocation record for FILE. The PFALLOC found in the directory alist is removed with \PFLOPPY.DIR.REMOVE and then reentered with \PFLOPPY.DIR.PUT under the new file name which \PFLOPPY.RENAMEFILE computes. Finally, the leader page for the PFALLOC is changed to have the new file name and then is written out to the floppy.

(\PFLOPPY.GETFILEINFO FILE ATTRIBUTE FDEV)

Called by GETFILEINFO. All file attributes like WRITEDATE, CREATIONDATE, LENGTH, and TYPE are stored on the leader page of a file. \PFLOPPY.GETFILEINFO returns values, sometimes suitably converted, out of the leader page.

(\PFLOPPY.SETFILEINFO FILE ATTRIBUTE VALUE)

Called by SETFILEINFO. All file attributes like WRITEDATE, CREATIONDATE, LENGTH, and TYPE are stored on the leader page of a file. The VALUE for ATTRIBUTE, sometimes suitably converted, is stored into the leader page. The updated leader page is then written out to the floppy if FILE is not open. (If FILE is open then the leader page will be written out to the floppy when FILE is closed.)

(\PFLOPPY.GETFILENAME FILE RECOG FDEV)

Called by FINDFILE and INFILEP. This function can get called if {FLOPPY} (or a directory name like {FLOPPY}<MYDIR>) is on the user's DIRECTORIES search path. When a function like OPENSTREAM fails to find a file on the user's connected directory, \PFLOPPY.GETFILENAME may get asked about FILE to see if FILE is on {FLOPPY}.

\PFLOPPY.GETFILENAME is coded very similar to \PFLOPPY.DIR.GET which searches the floppy directory alist stored in (fetch (PFLOPPYFDEV DIR) of \FLOPPYFDEV).

\PFLOPPY.GETFILENAME returns NIL if no file is found. NIL is also returned if there is no floppy in the floppy drive or if there is no floppy drive on the machine that the user is using.

**(\PFLOPPY.GENERATEFILES FDEV PATTERN
DESIREDPROPS OPTIONS)**

Called by DIRECTORY and the DIR LISPXMACRO. Like all corresponding GENERATEFILES functions installed on other file devices, \PFLOPPY.GENERATEFILES returns a file generator FILEGENOBJ which is a record that contains a GENFILESTATE plus two function names--in FLOPPY's case, \PFLOPPY.NEXTFILEFN and \PFLOPPY.FILEINFOFN.

The GENFILESTATE is a record that amounts to a stack of all the desired FLOPPY files according to the PATTERN, DESIREDPROPS, and OPTIONS that have been supplied. The files that go into the GENFILESTATE are collected by walking along the floppy directory alist which is stored on (fetch (PFLOPPYFDEV DIR) of \FLOPPYFDEV). A lot of the work is done by calling the function DIRECTORY.MATCH.

\PFLOPPY.NEXTFILEFN takes a GENFILESTATE, pops a file name out of the GENFILESTATE which also side effects the GENFILESTATE, and then returns the file name. If the GENFILESTATE has gone empty, then \PFLOPPY.NEXTFILEFN just returns NIL.

\PFLOPPY.FILEINFOFN is asked to supply information about the file that is at the top of the GENFILESTATE stack. This is done quite easily by calling the same function (\PFLOPPY.GETFILEINFO1) that \PFLOPPY.GETFILEINFO calls.

(\FLOPPY.HOSTNAMEP NAME FDEV)

This kind of function is intended for file devices that have nicknames. The HOSTNAMEP functions get called when the user specifies a file name containing a nickname as the host part of the file name. Since {FLOPPY} doesn't have any nicknames, \FLOPPY.HOSTNAMEP returns T iff NAME is FLOPPY.

(\FLOPPY.EVENTFN FDEV EVENT)

This function gets called before and after any LOGOUT, SYSOUT, MAKESYS, or SAVEVM.

Pilot Page Numbers

The hardware refers to locations of pages on a floppy by cylinder, head, and sector numbers. This way of enumerating pages on a floppy is cumbersome and the OSD Pilot design makes the somewhat sensible choice of making up its own numbering system for the pages on a floppy. We refer to these numbers as pilot page numbers.

\PFLOPPY.FIRSTDATAPAGE (Variable)

\PFLOPPY.LASTDATAPAGE (Variable)

Each cylinder+head+sector combination can be encoded as a single FIXP called a DISKADDRESS. The functions to convert between Pilot page numbers and DISKADDRESSES are

(\PFLOPPY.PAGENOTODISKADDRESS PAGENO)

(\PFLOPPY.DISKADDRESSTOPAGENO DISKADDRESS)

The functions to read and write pages to particular pilot page numbers are

(\PFLOPPY.READPAGENO PAGENO PAGE NOERROR)

(\PFLOPPY.WRITEPAGENO PAGENO PAGE NOERROR)

Pilot Marker Pages

Marker pages delimit the files and free blocks found on a Pilot floppy. The pattern of the data stored on a floppy is

DATA = MP ALLOC MP ALLOC MP ALLOC MP ... MP ALLOC MP

Generally a ALLOC is a free block or a leader page + file combination. One ALLOC is the filelist.

The declaration for Pilot floppy marker pages is

PMPAGE **(Datatype)**

```
(DATATYPE PMPAGE
  ((SEAL WORD)
   (VERSION WORD)
   (* Previous marker page entry *)
   (PLENGTH SWAPPEDFIXP)
   (PTYPE WORD)
   (PFILEID SWAPPEDFIXP)
   (PFILETYPE WORD)
   (NIL 121 WORD)
   (* Next marker page entry *)
   (NLENGTH SWAPPEDFIXP)
   (NTYPE WORD)
   (NFILEID SWAPPEDFIXP)
   (NFILETYPE WORD)
   (NIL 121 WORD)))
```

SEAL.PMPAGE **(Constant)**

VERSION.PMPAGE **(Constant)**

The SEAL and VERSION fields of a PMPAGE are the same for all marker pages. The magic constants are SEAL.PMPAGE and VERSION.PMPAGE. The SEAL.PMPAGE magic constant is arbitrary enough that there is only a slight chance that a non marker page would begin with this particular word. Hence, the scavenger can search for marker pages by looking for pages that begin with SEAL.PMPAGE.

The fields PLENGTH, PTYPE, PFILEID, PFILETYPE describe the ALLOC preceding the marker page. The fields NLENGTH, NTYPE, NFILEID, NFILETYPE describe the ALLOC following the marker page.

PLENGTH, NLENGTH = Length of ALLOC in pages.

PTYPE, NTYPE = Free, file, or filelist.

PFILEID, NFILEID = Pretty worthless. Used by Mesa. Each ALLOC has its own fileid number in the Mesa floppy handler. Not used by the Xerox Lisp software.

PFILETYPE, NFILETYPE =Free, file, or filelist. At first glance you might think that there isn't much point in having both PTYPE, NTYPE and PFILETYPE, NFILETYPE. If you think that, you are right. Please remember before you throw the rotten oranges that this is not my design. Blame it on OSD.

Pilot Leader Pages

The pattern of the data stored on a floppy is

DATA = MP ALLOC MP ALLOC MP ALLOC MP ... MP ALLOC MP

Generally an ALLOC is a free block or a leader page + file combination. One ALLOC is the filelist. Thus the possible patterns for an ALLOC look like

ALLOC = FREE
 ALLOC = LP FILE
 ALLOC = FILELIST

We note that all leader pages that occur on a Pilot floppy are immediately preceded by a marker page. All files on a Pilot floppy are preceded by a leader page. Each leader page + file combination is surrounded by two marker pages.

An exception to the rule: Microcode files on boot floppies do not have leader pages. They look like ALLOC = FILE. You may wonder why a microcode file's ALLOC shouldn't also be ALLOC = LP FILE. There isn't any reason for the exception. This is just another feature of the OSD design.

The declaration for Pilot floppy leader pages is

PLPAGE **(Datatype)**

```
(DATATYPE PLPAGE
  ((SEAL WORD)
   (VERSION WORD)
   (MESATYPE WORD)
   (* Offset 6 *)
   (\CREATIONDATE SWAPPEDFIXP)
   (\WRITEDATE SWAPPEDFIXP)
   (PAGELENGTH SWAPPEDFIXP)
   (HUGEPAGESTART SWAPPEDFIXP)
   (HUGEPAGELENGTH SWAPPEDFIXP)
   (HUGELLENGTH SWAPPEDFIXP)
   (\NAMELENGTH WORD)
   (NAMEMAXLENGTH WORD)
   (* Offset 17 *)
   (\NAME 50 WORD)
   (* Offset 67 *)
   (UFO1 WORD)
   (UFO2 WORD)
   (DATAVERSION WORD)
   (\TYPE WORD)
   (NIL 183 WORD)
   (\BYTESIZE WORD))
```

The SEAL and VERSION fields of a PLPAGE are the same for all marker pages. The magic constants are SEAL.PLPAGE and VERSION.PLPAGE. The SEAL.PLPAGE magic constant is arbitrary enough that there is only a slight chance that a non marker page would begin with this particular word. Hence, the scavenger can search for marker pages by looking for pages that begin with SEAL.PLPAGE.

MESATYPE is fairly obscure. I think it marks the file according to what kind of application produced the file. Each application can register itself with OSD who doles out the numbers.

\CREATIONDATE is the obvious. However, the numeric encryption is according to Mesa's standards.

\WRITEDATE is the obvious. However, the numeric encryption is according to Mesa's standards.

PAGELength is the length of the file after the leader page in pages.

HUGEPAGESTART, HUGEPAGELength, HUGELength are pretty arcane. For ordinary files, HUGEPAGESTART = 0 and HUGEPAGELength = PAGELength. For all files, HUGELength = the length of the file in bytes. For sysout and huge pilot files, it is possible that HUGEPAGESTART > 0 and HUGEPAGELength < PAGELength. Or some funny business like this.

\NAMELength, NAMEMAXLength, \NAME indicate the name of the file following the leader page. Oddly enough, NAMEMAXLength is just a constant field for all leader pages.

UFO1, UFO2. Flying saucers. Treated as constant fields by Lisp.

DATAVERSION. Another constant field.

\TYPE = binary or text.

Pilot Sector 9

The name of the floppy is stored on the page at location CYLINDER = 0, HEAD = 0, SECTOR = 9. All of CYLINDER = 0 is wasted except for sector 9. There is nothing valuable stored on the 8 sectors before sector 9, or any of the sectors on CYLINDER = 0.

All the tracks from CYLINDER=1 to \FLOPPY.CYLINDERS are formatted IBM double density 512 bytes per sector. Strangely, the two sides of CYLINDER = 0 are formatted differently. As part of the OSD design, CYLINDER = 0, HEAD = 0 is formatted IBM single density 128 bytes per sector and CYLINDER = 0, HEAD = 1 is formatted IBM double density 256 bytes per sector.

The pattern of the data stored in sector 9 is conveniently represented by a datatype called PSECTOR9.

PSECTOR9

(Datatype)

(DATATYPE PSECTOR9

((SEAL WORD)

(VERSION WORD)

(CYLINDERS WORD)

(TRACKSPERCYLINDER WORD)

(SECTORSPEPTRACK WORD)

(PFILELISTSTART WORD)

(PFILELISTFILEID SWAPPEDFIXP)

(PFILELISTLENGTH WORD)

(ROOTFILEID SWAPPEDFIXP)

(NIL WORD)

(PILOTMICROCODE WORD)

(DIAGNOSTICMICROCODE WORD)

(GERM WORD)

(PILOTBOOTFILE WORD)

(FIRSTALTERNATESECTOR WORD)

(COUNTBADSECTORS WORD)

(NEXTUNUSEDFILEID SWAPPEDFIXP)

(CHANGING FLAG)

```
(NIL BITS 15)
(\LABELLENGTH WORD)
(\LABEL 106 WORD)))
```

Lisp verifies that the SEAL of the PSECTOR9 that it reads in is equal to magic constant SEAL.PSECTOR9. This action serves to check that the floppy that is being read is in fact a Pilot floppy.

A floppy's name is stored in \LABELLENGTH and \LABEL. The floppy name is accessed by the user through the function FLOPPY.NAME.

Everything else about PSECTOR9 is not useful to Lisp and is not used by any of the Lisp code. However Mesa does use some of this cruft and Lisp has to keep the cruft consistent with what Mesa would like to see.

Occasionally, the filelist (also useless) moves around on the floppy. When this happens, PFILELISTSTART, PFILELISTFILEID, PFILELISTLENGTH also have to be updated.

NEXTUNUSEDFILEID has to do with the Mesa floppy handler's notion of what a fileid is. This notion is irrelevant to the Lisp implementation of FLOPPY, but the Lisp implementation of FLOPPY does have to keep NEXTUNUSEDFILEID equal to one plus the number of files currently stored on the floppy.

Every time a new file is created and closed, NEXTUNUSEDFILEID has to be increased by one, the filelist has to change, and both the PSECTOR9 and the filelist have to be written out to the floppy.

(IPFLOPPY.SAVE.PSECTOR9 NOERROR)

Saves the cached PSECTOR9 out to the floppy.

Pilot Filelist

The Pilot filelist is not used by the Lisp FLOPPY but must be maintained by the Lisp FLOPPY code to be compatible with the Mesa floppy handler. The filelist has the form

```
FILELIST = SEAL VERSION NENTRIES MAXENTRIES ENTRY
          ENTRY ENTRY ... ENTRY
```

Every time a new file is created and closed, an ENTRY has to be added to FILELIST and FILELIST has to be written out. Every time a file is deleted, an ENTRY has to be deleted from FILELIST and FILELIST has to be written out. Other than to maintain the FILELIST in this way for Mesa's benefit, Lisp does not use the FILELIST. Lisp uses the marker pages on a floppy to find where the files are and could care less about the filelist.

The filelist and filelist entry record declarations are

```
(BLOCKRECORD PFILELIST
 ((SEAL WORD)
  (VERSION WORD)
  (NENTRIES WORD)
  (MAXENTRIES WORD)))
```

```
(DATATYPE PFLE
  ((FILEID SWAPPEDFIXP)
   (TYPE WORD)
   (START WORD)
   (LENGTH WORD)))
```

(IPFLOPPY.ADD.TO.PFILELIST PFALLOC)

Create a PFLE file list entry on the filelist for this PFALLOC.

(IPFLOPPY.DELETE.FROM.PFILELIST PFALLOC)

Deletes the PFLE file list entry on the filelist corresponding to this PFALLOC.

(IPFLOPPY.SAVE.PFILELIST NOERROR)

Saves the filelist out to the floppy.

Pilot Floppy Format

The overall architecture of a Pilot floppy looks like this

FLOPPY = CYLINDER0 DATA

CYLINDER0 = CYLINDER0HEAD0 CYLINDER0HEAD1

CYLINDER0HEAD0 = GARBAGE SECTOR9 GARBAGE

CYLINDER0HEAD1 = GARBAGE

DATA = DATA = MP ALLOC MP ALLOC MP ... MP ALLOC MP

ALLOC = FREE
 ALLOC = LP FILE
 ALLOC = FILELIST

Exactly one ALLOC is the FILELIST. SECTOR9 and the FILELIST are for the most part useless. However, the Lisp implementation has to maintain SECTOR9 and FILELIST in a way that will make the Mesa floppy handler happy.

The Lisp implementation finds out what files are present on a floppy by following the marker pages (MPs) in the DATA which always begins at \PFLOPPYFIRSTDATAPAGE and ends at \PFLOPPYLASTDATAPAGE. There are fields on the marker pages telling how long and what kind of ALLOC appears on either side of the marker page.

The formatting function for Pilot floppies is

(IPFLOPPY.FORMAT NAME AUTOCONFIRMFLG SLOWFLG)

\PFLOPPY.FORMAT formats the tracks of the floppy according to the peculiar OSD design and then makes the floppy look like it is a Pilot floppy with an empty directory. The DATA on a freshly formatted floppy has the form

DATA = MP FILELIST MP FREE MP

Cached Pilot Floppy Information

Duplicate directory information about a floppy is cached in core. This is held on to by a PFINFO datatype.

PFINFO (Datatype)

(DATATYPE PFINFO (OPEN PFILELIST PFALLOCS DIR PSECTOR9))

We may describe the fields of the PFINFO slightly out of order.

OPEN is a flag saying whether directory information for the current floppy has been cached or not in the remaining fields of the PFINFO.

PFILELIST is the Pilot filelist. The filelist is unimportant except that it must be maintained for Mesa's benefit. An updated PFILELIST has to be written out each time a file is written etc.

PSECTOR9 is the Pilot sector 9 record. This is where the floppy name lives. Otherwise, this record is also pretty worthless as far as Lisp is concerned. But there are fields on the PSECTOR9 that have to be maintained for Mesa's benefit. An updated PFILELIST has to be written out each time a file is written etc.

DIR is the cached floppy directory alist which is an alist of alists of alists.

PFALLOCS is a list of PFALLOC records which describe successive allocations on the floppy.

Cached Pilot Floppy DIR Alist

The PFINFO DIR is the cached floppy directory alist which is an alist of alists of alists. It has the form

```
((name (extension (version . PFALLOC) ...
              (version . PFALLOC)) ...
  (extension (version . PFALLOC) ...
              (version . PFALLOC))) ...
 (name (extension (version . PFALLOC) ...
              (version . PFALLOC)) ...
  (extension (version . PFALLOC) ...
              (version . PFALLOC))))
```

Say for example that we have 6 versions of the file {FLOPPY}FLOPPY.TEDIT stored on a floppy. Then the cached DIR alist could look like

```
((FLOPPY (TEDIT (1 . {PFALLOC}#55,72740)
                 (2 . {PFALLOC}#55,72704)
                 (3 . {PFALLOC}#55,72650)
                 (4 . {PFALLOC}#55,72434)
                 (5 . {PFALLOC}#55,72400)
```

(6 . {PFALLOC}#55,72340))))

Functions to manage and access the DIR alist are

(IPFLOPPY.DIR.GET FILENAME RECOG)

(IPFLOPPY.DIR.PUT FILENAME RECOG PFALLOC)

(IPFLOPPY.DIR.REMOVE PFALLOC)

(IPFLOPPY.DIR.VERSION VERSION RECOG VALIST
FILENAME)

Cached Pilot Floppy PFALLOCS

The PFINFO PFALLOCS is a list of PFALLOC records which describe successive allocations on the floppy. Some of the PFALLOCS are leader page+file combinations. Some are free blocks. One of the PFALLOCS is the allocation record for the filelist (this PFALLOC tells us where the contents of PFILELIST should be stored).

Each PFALLOC record has the form

PFALLOC **(Datatype)**

(DATATYPE PFALLOC
(FILENAME
(PREV FULLXPOINTER)
NEXT
START
PMPAGE
PLPAGE
PFLE
(WRITEFLG FLAG)
(DELETEFLG FLAG)))

FILENAME is the name of the file that the PFALLOC corresponds to. If the PFALLOC does not correspond to a file, then FILENAME will be a list like (FREE) or (PFILELIST).

PREV points back to the preceding PFALLOC in the PFALLOCS list.

NEXT points to the next PFALLOC in the PFALLOCS list.

START is the pilot page number of the first page of storage on the floppy corresponding to the PFALLOC (the page after the marker page preceding the allocation).

PMPAGE is a cached copy of the marker page preceding the allocation corresponding to the PFALLOC.

PLPAGE is a cached copy of the leader page, if any, that begins the allocation. (Only file allocations will have leader pages. If the PFALLOC does not correspond to a file allocation then the PLPAGE field is left NIL.)

PFLE is the file list entry in the Pilot filelist corresponding to this PFALLOC.

WRITEFLG = T if there is a stream writing to the allocation corresponding to this PFALLOC. That is, this PFALLOC corresponds to a file, and a stream is writing to that file.

DELETEFLG = T if the PFALLOC corresponds to a file whose deletion is pending. This may happen if some process is reading a file that a second process DEFILES. The PFALLOC is deleted for real (i.e. made into a free block) when the first process gives up control of the stream that is reading the file with CLOSEF.

PILOT FLOPPY STORAGE ALLOCATION

This section describes the Pilot floppy storage allocation strategy.

(\PFLOPPY.ALLOCATE LENGTH)

This function is called to generate the initial allocation PFALLOC assigned to a stream. The PFALLOC must be of a definite length and so there are other functions like \PFLOPPY.TRUNCATE and \PFLOPPY.EXTEND which know how to split an incompletely used up PFALLOC or a PFALLOC that is about to overflow.

\PFLOPPY.ALLOCATE returns a PFALLOC pointing to a free block of storage that is at least LENGTH pages long. If LENGTH=NIL is supplied, then the length is defaulted to somewhere between DEFAULT.ALLOCATION and MINIMUM.ALLOCATION with preference towards DEFAULT.ALLOCATION.

The strategy is to first select the largest free block available. Not being able to find a free block causes \PFLOPPY.GAINSPACE to be called and then \PFLOPPY.ALLOCATE retries.

Having obtained the PFALLOC pointing to the largest free block available, \PFLOPPY.ALLOCATE determines if the free block is big enough. If the free block is not big enough then \PFLOPPY.GAINSPACE is called and \PFLOPPY.ALLOCATE retries.

Now being in possession of a PFALLOC pointing to a sufficiently large free block, the question is whether or not to split the PFALLOC. If the length of PFALLOC exceeds LENGTH by MINIMUM.ALLOCATION then PFALLOC is split into two free blocks one of which will have length LENGTH and will be returned. Otherwise, the PFALLOC is returned as is.

Splitting excessively large PFALLOCs is done in order that no stream hog the floppy and thereby be unfair to other streams. It may happen, for example in a freshly formatted floppy, that there is just one or very few but large free blocks. It would be unfair to hand out the only and very large free block to a stream and thereby prevent a second stream from opening.

On the other hand, splitting a PFALLOC whose length exceeds LENGTH by just a little bit is also undesirable because it leads to fragmentation and consumption of storage by the marker page overhead that must occur for each allocation. Thus, the length of

PFALLOC not exceeding LENGTH by MINIMUM.ALLOCATION is considered to be close enough.

\PFLOPPY.ICHECK is called at the end of each \PFLOPPY.ALLOCATE to do an integrity check of the cached incore description of the floppy.

(\PFLOPPY.DEALLOCATE PFALLOC)

Deallocation is fairly easy. The two marker pages surrounding the allocation pointed to by PFALLOC are made to say that the type of the allocation between them is a free block.

\PFLOPPY.ICHECK is called at the end of each \PFLOPPY.DEALLOCATE to do an integrity check of the cached incore description of the floppy.

(\PFLOPPY.TRUNCATE PFALLOC LENGTH)

Truncation is also fairly easy. If PFALLOC is already of length LENGTH or smaller then nothing needs to be done.

Otherwise \PFLOPPY.TRUNCATE changes the local situation on the floppy from one of

MP ALLOC MP

to one of

MP ALLOC MP FREE MP

by setting the length of PFALLOC to LENGTH, creating another PFALLOC to take up the slack and to be considered as a free block, and creating and updating and writing out three marker pages.

\PFLOPPY.ICHECK is called at the end of each \PFLOPPY.TRUNCATE to do an integrity check of the cached incore description of the floppy.

(\PFLOPPY.EXTEND PFALLOC)

This function gets called when an output stream is about to overflow its initial allocation. \PFLOPPY.EXTEND is charged with extending the length of PFALLOC's storage allocation.

If PFALLOC is followed by a free block, then it suffices to let PFALLOC cannibalize the following free block. The situation

MP ALLOC MP FREE MP

is changed to one of

MP ALLOC MP

by changing the length of PFALLOC to the sum of the length of PFALLOC's length, one page from the marker page between PFALLOC and the free block which is eliminated, and the length of the free block.

If PFALLOC is not followed by a free block, but instead is bumping up against valuable data, then a different strategy is used.

\PFLOPPY.ALLOCATE is called to find a new and larger storage area NEW for the data stored at the allocation pointed to by PFALLOC. The data stored in the allocation pointed to by PFALLOC is then copied into the allocation pointed to by NEW. This is slightly time consuming, so the Lisp code prints the message "Reallocating" in the PROMPTWINDOW when this kind of activity is about to happen. After copying the data pointed to by PFALLOC into the area pointed to by NEW, NEW and PFALLOC are made to exchange places. NEW is changed to point to where the data used to live and PFALLOC is changed to point to where the data has moved to. The marker pages around NEW and around PFALLOC have to be updated and written out to the floppy. The result, therefore, has the appearance of data crawling out of its restricted cavity and kicking out the free block living from the bigger cavity and then the free block moves over to where the data used to live.

\PFLOPPY.ICHECK is called at the end of each \PFLOPPY.EXTEND to do an integrity check of the cached incore description of the floppy.

(\PFLOPPY.GAINSPACE LENGTH)

This function is charged with going out and hunting up space on a floppy. \PFLOPPY.GAINSPACE returns after a free block of length LENGTH has been made available.

The first thing to be done is to merge adjacent free blocks into larger free blocks. This is done by calling function \PFLOPPY.GAINSPACE.MERGE. \PFLOPPY.ICHECK is called at the end of each \PFLOPPY.GAINSPACE.MERGE to do an integrity check of the cached incore description of the floppy.

At this point \PFLOPPY.GAINSPACE checks to see if a free block of length LENGTH has been made available, and if so, returns.

Otherwise, \PFLOPPY.GAINSPACE calls FLOPPY.FREE.PAGES to determine whether a sufficiently large free block can be gained just by compacting the floppy. If the number of free pages available is greater than LENGTH, then compacting the floppy will have as one of its effects the collection of all free space into a single free block. Therefore, if the number of free pages available is greater than LENGTH, then \PFLOPPY.GAINSPACE calls FLOPPY.COMPACT to compact the floppy and then returns the single free block that is created by the compactor.

If calling FLOPPY.FREE.PAGES tells \PFLOPPY.GAINSPACE that there wouldn't be a large enough free block created just by compacting the floppy, then \PFLOPPY.GAINSPACE generates the usual controllable FILE SYSTEM RESOURCES EXCEEDED error break. If the user responds by deleting some files and typing OK, then \PFLOPPY.GAINSPACE continues onward by going back to the top of its list of things to do and retrying its attempt to find a free block of length LENGTH.

(\PFLOPPY.FREE.PAGES)

Just sums up the lengths of all the PFALLOCs pointing to free blocks. This function gets called by the user function FLOPPY.FREE.PAGES.

(\PFLOPPY.ICHECK)

Does an integrity check on the Lisp implementation's incore cached directory information.

The Lisp floppy implementation does not blithely trust itself to always be doing the right thing, at least as far as the incore description of how the floppy is arranged. It is somewhat important that the in core description be absolutely legal and in agreement with reality out on the floppy, because incorrect cached directory structures may cause floppy operations to scramble the user's floppy confusing contents of files or real directory structure on the floppy.

\PFLOPPY.ICHECK gets called near the end of each major function in the Pilot floppy allocation code.

SYSOUT

Sysouting to floppy is kind of a hack. The sysout file that gets written has to be broken into smaller files that will fit on individual floppies. The small files are not ordinary files but are Huge Pilot files. The HUGEPAGESTART and HUGEPAGELENGTH fields in the leader pages of these files become important.

\SFLOPPYFDEV

(Variable)

The usual FLOPPY file device is \PFLOPPYFDEV. The sysout FLOPPY file device is \SFLOPPYFDEV. (FLOPPY.MODE 'SYSOUT) by the user makes the switch between the two file devices. The function SYSOUT seems to do (FLOPPY.MODE 'SYSOUT) automatically for the user, but in fact, floppy functions at a lower level detect SYSOUT on the stack of function calls by a certain amount of gruneyness and make the switch between file devices when necessary.

(\SFLOPPY.OPENHUGEFILE FILE ACCESS RECOG OTHERINFO FDEV OLDSTREAM)

Installed on \SFLOPPYFDEV and gets called when FILEIO opens a stream for input or output when FLOPPY is in SYSOUT mode. \SFLOPPY.OPENHUGEFILE returns a stream datatype. The stream can be either input or output.

If the stream is to be an input stream, \SFLOPPY.INPUTFLOPPY is called. If the stream is to be an output stream, \SFLOPPY.OUTPUTFLOPPY is called.

The stream returned is in every way like a Pilot stream returned by \PFLOPPY.OPENFILE with the exception that the DEVICE of the stream is \SFLOPPYFDEV. Two other fields on the stream, F1 and F2, point to the allocation record (PFALLOC) and leader page (PLPAGE) for the stream. The PFALLOC and PLPAGE can be conveniently accessed by using the FLOPPYSTREAM ACCESSFNS.

(\SFLOPPY.READPAGES STREAM FIRSTPAGE# BUFFERS)

Installed on \SFLOPPYFDEV and called by FILEIO when FLOPPY is in SYSOUT mode. Reads sectors off floppies into virtual memory pages BUFFERS. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a

file. \SFLOPPY.READPAGES therefore fills BUFFERS with data read from floppies beginning with the FIRSTPAGE# of STREAM.

\SFLOPPY.READPAGES is implemented by calling \PFLOPPY.READPAGE. When \SFLOPPY.READPAGES is about to run off the end of a floppy, \SFLOPPY.CLOSEFLOPPY and \SFLOPPY.INPUTFLOPPY are called to bring in the next floppy.

(\SFLOPPY.WRITEPAGES STREAM FIRSTPAGE# BUFFERS)

Installed on \SFLOPPYFDEV and called by FILEIO when FLOPPY is in SYSOUT mode. Writes contents of virtual memory pages BUFFERS on to sectors of floppies. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \SFLOPPY.WRITEPAGES therefore writes to the floppy beginning with the location corresponding to the FIRSTPAGE# of STREAM.

\SFLOPPY.WRITEPAGES is implemented by calling \PFLOPPY.WRITEPAGE. When \SFLOPPY.WRITEPAGES is about to run off the end of a floppy, \SFLOPPY.CLOSEFLOPPY and \SFLOPPY.OUTPUTFLOPPY are called to bring in the next floppy.

(\SFLOPPY.CLOSEHUGEFILE STREAM)

Installed on \SFLOPPYFDEV and called by FILEIO to close a sysout file when FLOPPY is in SYSOUT mode.

For output streams, does the usual \CLEARMAP plus a \SFLOPPY.CLOSEFLOPPY on the last floppy being written.

There is some grungeyness to switch back to an old FLOPPY mode if the FLOPPY mode was reset by calling function SYSOUT.

(\SFLOPPY.INPUTFLOPPY FLOPPYNAME FILENAME OTHERINFO OLDSTREAM)

Called by \SFLOPPY.READPAGES when \SFLOPPY.READPAGES needs to go to the next floppy. \SFLOPPY.INPUTFLOPPY prompts the user to "Insert floppy" and does a (FLOPPY.WAIT.FOR.FLOPPY T) until the user does so.

Once the user has inputted the next floppy and FLOPPY.WAIT.FOR.FLOPPY has returned, \SFLOPPY.INPUTFLOPPY calls \PFLOPPY.OPENFILE to open a regular Pilot floppy stream to the piece of sysout file stored on the floppy inserted.

If no OLDSTREAM is supplied, then the Pilot floppy stream created by the call to \PFLOPPY.OPENFILE is returned. Otherwise, the PFALLOC and PLPAGE cached in the F1 and F2 fields of the Pilot floppy stream are pulled out and stuck into OLDSTREAM, after which OLDSTREAM is returned.

(\SFLOPPY.OUTPUTFLOPPY FLOPPYNAME FILENAME OTHERINFO OLDSTREAM)

Called by \SFLOPPY.WRITEPAGES when \SFLOPPY.WRITEPAGES needs to go to the next floppy. \SFLOPPY.OUTPUTFLOPPY prompts the user to "Insert floppy" and does a (FLOPPY.WAIT.FOR.FLOPPY T) until the user does so.

Once the user has inputted the next floppy and FLOPPY.WAIT.FOR.FLOPPY has returned, \SFLOPPY.OUTPUTFLOPPY tries to format the floppy. If there is any problem with formatting the floppy, such as the floppy being writeprotected, then the user is again asked to input a floppy.

Once the new floppy is formatted, \SFLOPPY.OUTPUTFLOPPY calls \PFLOPPY.OPENFILE to open a regular Pilot floppy stream which will be used to create a piece of the sysout file being stored on floppies.

If no OLDSTREAM is supplied, then the Pilot floppy stream created by the call to \PFLOPPY.OPENFILE is returned. Otherwise, the PFALLOC and PLPAGE cached in the F1 and F2 fields of the Pilot floppy stream are pulled out and stuck into OLDSTREAM, after which OLDSTREAM is returned.

(\SFLOPPY.CLOSEFLOPPY STREAM LASTFLOPPYFLG)

If the STREAM is an input stream, then \SFLOPPY.CLOSEFLOPPY just returns without doing anything.

If the STREAM is an output stream, then the leader pages and marker pages for the piece of sysout file stored on this floppy are written out. And as usual with Pilot streams, the updated filelist and PSECTOR9 also have to be written out.

HUGEPILOT

HUGEPILOT mode is implemented in a way similar to SYSOUT mode. The Huge Pilot file that gets written has to be broken into smaller files that will fit on individual floppies. The small files are not ordinary files but are Huge Pilot files. The HUGEPAGESTART and HUGEPAGELENGTH fields in the leader pages of these files become important.

\HFLOPPYFDEV

(Variable)

The usual FLOPPY file device is \PFLOPPYFDEV. The Huge Pilot FLOPPY file device is \HFLOPPYFDEV. (FLOPPY.MODE 'HUGEPILOT) by the user makes the switch between the two file devices.

(\HFLOPPY.OPENHUGEFILE FILE ACCESS RECOG OTHERINFO FDEV OLDSTREAM)

Installed on \HFLOPPYFDEV and gets called when FILEIO opens a stream for input or output when FLOPPY is in HUGEPILOT mode. \HFLOPPY.OPENHUGEFILE returns a stream datatype. The stream can be either input or output.

If the stream is to be an input stream, \HFLOPPY.INPUTFLOPPY is called. If the stream is to be an output stream, \HFLOPPY.OUTPUTFLOPPY is called.

The stream returned is in every way like a Pilot stream returned by \PFLOPPY.OPENFILE with the exception that the DEVICE of the stream is \HFLOPPYFDEV. Two other fields on the stream, F1 and F2, point to the allocation record (PFALLOC) and leader page (PLPAGE) for the stream. The PFALLOC and PLPAGE can be

conveniently accessed by using the FLOPPYSTREAM ACCESSFNS.

(\HFLOPPY.READPAGES STREAM FIRSTPAGE# BUFFERS)

Installed on \HFLOPPYFDEV and called by FILEIO when FLOPPY is in HUGEPILOT mode. Reads sectors off floppies into virtual memory pages BUFFERS. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \HFLOPPY.READPAGES therefore fills BUFFERS with data read from floppies beginning with the FIRSTPAGE# of STREAM.

\HFLOPPY.READPAGES is implemented by calling \PFLOPPY.READPAGE. When \HFLOPPY.READPAGES is about to run off the end of a floppy, \HFLOPPY.CLOSEFLOPPY and \HFLOPPY.INPUTFLOPPY are called to bring in the next floppy.

(\HFLOPPY.WRITEPAGES STREAM FIRSTPAGE# BUFFERS)

Installed on \HFLOPPYFDEV and called by FILEIO when FLOPPY is in HUGEPILOT mode. Writes contents of virtual memory pages BUFFERS on to sectors of floppies. FIRSTPAGE# is in FILEIO's scheme of counting the pages of a file, beginning with 0 for the first page of a file. \HFLOPPY.WRITEPAGES therefore writes to the floppy beginning with the location corresponding to the FIRSTPAGE# of STREAM.

\HFLOPPY.WRITEPAGES is implemented by calling \PFLOPPY.WRITEPAGE. When \HFLOPPY.WRITEPAGES is about to run off the end of a floppy, \HFLOPPY.CLOSEFLOPPY and \HFLOPPY.OUTPUTFLOPPY are called to bring in the next floppy.

(\HFLOPPY.CLOSEHUGEFILE STREAM)

Installed on \HFLOPPYFDEV and called by FILEIO to close a Huge Pilot file when FLOPPY is in HUGEPILOT mode.

For output streams, does the usual \CLEARMAP plus a \HFLOPPY.CLOSEFLOPPY on the last floppy being written.

(\HFLOPPY.INPUTFLOPPY FLOPPYNAME FILENAME OTHERINFO OLDSTREAM)

Called by \HFLOPPY.READPAGES when \HFLOPPY.READPAGES needs to go to the next floppy. \HFLOPPY.INPUTFLOPPY prompts the user to "Insert floppy" and does a (FLOPPY.WAIT.FOR.FLOPPY T) until the user does so.

Once the user has inputted the next floppy and FLOPPY.WAIT.FOR.FLOPPY has returned, \HFLOPPY.INPUTFLOPPY calls \PFLOPPY.OPENFILE to open a regular Pilot floppy stream to the piece of Huge Pilot file stored on the floppy inserted.

If no OLDSTREAM is supplied, then the Pilot floppy stream created by the call to \PFLOPPY.OPENFILE is returned. Otherwise, the PFALLOC and PLPAGE cached in the F1 and F2 fields of the Pilot floppy stream are pulled out and stuck into OLDSTREAM, after which OLDSTREAM is returned.

(\HFLOPPY.OUTPUTFLOPPY FLOPPYNAME FILENAME OTHERINFO OLDSTREAM)

Called by `\HFLOPPY.WRITEPAGES` when `\HFLOPPY.WRITEPAGES` needs to go to the next floppy. `\HFLOPPY.OUTPUTFLOPPY` prompts the user to "Insert floppy" and does a `(FLOPPY.WAIT.FOR.FLOPPY T)` until the user does so.

Once the user has inputted the next floppy and `FLOPPY.WAIT.FOR.FLOPPY` has returned, `\HFLOPPY.OUTPUTFLOPPY` tries to format the floppy. If there is any problem with formatting the floppy, such as the floppy being writeprotected, then the user is again asked to input a floppy.

Once the new floppy is formatted, `\HFLOPPY.OUTPUTFLOPPY` calls `\PFLOPPY.OPENFILE` to open a regular Pilot floppy stream which will be used to create a piece of the Huge Pilot file being stored on floppies.

If no `OLDSTREAM` is supplied, then the Pilot floppy stream created by the call to `\PFLOPPY.OPENFILE` is returned. Otherwise, the `PFALLOC` and `PLPAGE` cached in the `F1` and `F2` fields of the Pilot floppy stream are pulled out and stuck into `OLDSTREAM`, after which `OLDSTREAM` is returned.

(\HFLOPPY.CLOSEFLOPPY STREAM LASTFLOPPYFLG)

If the `STREAM` is an input stream, then `\HFLOPPY.CLOSEFLOPPY` just returns without doing anything.

If the `STREAM` is an output stream, then the leader pages and marker pages for the piece of Huge Pilot file stored on this floppy are written out. And as usual with Pilot streams, the updated filelist and `PSECTOR9` also have to be written out.

Font/Character Documentation

Greg Nuyens

filed as: {eris}<lispcore>internal>doc>font&chars.tedit
last edited: March 24, 1986

Notice:

this is a draft made available for comments. Please do not forward copies.

**Comments are encouraged. Please send them to
Nuyens.pa@Xerox.com**

This note provides information about font and character facilities in Interlisp-D. It is organised in two parts:

- 1) a user level view of the changes involved in including NS characters in Interlisp-D (adapted from the Koto release notes ({Erinyes}<doc>koto>releasenotes>*))
- 2) a description of the underlying data-structures and facilities. (exported macros and fns, etc)

User-level view

Interlisp-D now supports the Xerox corporate character code standard, commonly referred to as the NS (Network Systems) encoding, described in the document *Character Code Standard* [Xerox System Integration Standards, XSIS 058404, April 1984]. Previous to the Koto release, Interlisp-D used the ASCII (American Standard Code for Information Interchange) encoding. While the extended-ASCII encoding provided for 8-bit (256 available) characters (primarily Latin alphabet and computer-specific symbols), the NS encoding supports 16-bit (65536 available) characters comprising many foreign alphabets and special symbols.

The benefit of having this large character set, in contrast to approaches that use a small set of character codes and a multiplicity of fonts (e.g., a Greek font, a math font), is that each semantically distinct character is represented by its own character code, completely independent of the character's appearance (font). Thus, the Greek character upper-case Beta is always character code 9794, independent of whether it appears in printed form in a serif style, sans-serif style, italic, etc., and it is unrelated to the Roman letter B (character code 66).

NS characters can be used in strings, litatom print names, symbolic files, or anywhere else that characters can be used. All of the standard string and print name functions (RPLSTRING, GNC, NCHARS, STRPOS, etc.) accept litatoms and strings containing NS characters. For example:

```
_(STRPOS "char" "this is an 8-bit character string")
```

18

```
_(STRPOS "char" "celui-ci comporte des caractères NS")
```

23

Characters are organized into 256-member *character sets*, each of which generally consists of semantically related characters. For example, character set 38 is the Greek character set and contains the Greek alphabet and punctuation characters needed to print Greek text. A 16-bit character code thus consists of an 8-bit character set and an 8-bit character number within that set. The ASCII character set is contained in NS character set zero; thus, ASCII characters are still represented by the same 8-bit character codes as previously (i.e., 16-bit character codes whose high 8 bits are zero). Most strings and atoms still consist entirely of characters from character set zero and are represented just as space-efficiently in memory and on files as in earlier releases of Interlisp-D that used only ASCII characters.

In almost all cases, a program does not need to know that it is dealing with 16-bit characters rather than 8-bit characters—the higher level system functions all treat them transparently. The exception is in character-level input/output, where the important fact to be aware of is that *characters are not bytes*. The file pointer of a random-access file still counts bytes, and the function NCHARS still counts characters, but the two are no longer directly related. This is discussed in more detail below.

Character-level Input/Output

Incompatible Change:

BIN and BOUT are no longer appropriate for character input/output.

A character is no longer generally representable in 8 bits. Therefore, characters can no longer, in general, be read or written with the functions BIN and BOUT, which read and write 8-bit quantities. The change is mostly transparent to user programs, especially if those programs use only the higher level functions, such as READ and PRINT. However, it is likely that user programs that manipulated a file character by character using BIN and BOUT should now use the following functions, which may produce or consume more than a single byte:

(READCCODE *STREAM*) [Function]

(PEEKCCODE *STREAM*) [Function]

(PRINTCCODE *CODE STREAM*) [Function]

These functions are documented in the new Interlisp-D Reference Manual. The functions BIN and BOUT are still appropriate for use when reading and writing strictly binary (rather than character) data.

Interlisp-D supports two ways of writing NS characters on files. One way is to write the full 16-bits (two bytes) every time a character is output. The other way, which is the system default, is to use "run-encoding," in which a run of characters in the same character set is written as a sequence of 8-bit character numbers within the character set, preceded by a "change character set" command. The byte 255 (illegal as either a character set number or a character number) followed by a character set number is used to signal a change to a given character set; the following bytes, up until the next change-character set sequence, are all interpreted as coming from the specified character set. Run-encoding can reduce the number of bytes required to encode a string of NS characters, as long as there are long sequences of characters from the same character set, which is usually the case.

Most characters in common use, including those in the ASCII character set, are in character set zero; a file containing only these characters is thus in exactly the same format as in previous releases, viz., one byte per character. However, this should not be relied on.

The fact that the file representation of a character may be more than a single byte has important consequences for any program that uses random access on text files whose characters are run-encoded. First, and most obviously, you cannot count the characters in a string being printed and use that number to derive the file pointer of where the string ends—you must use GETFILEPTR. Second, programs that use SETFILEPTR need to be aware of possible character set changes. At any point when a file is being read or written, it has a "current character set," viz., the character set specified in the most recent "change character set" command written on the file. If the file pointer is changed with SETFILEPTR to a part of the file with a different character set, any characters read or written may have the wrong character set. Programs that use COPYBYTES to copy blocks of characters must ensure both that they are copying on character boundaries and copying to a place that is in the correct character set.

(Internal Note: PRINTCCODE is the user entry to the OUTCHARFN of the stream. It is bounds checked.)

The current character set can be accessed with the following function:

(CHARSET *STREAM CHARACTERSET*) [Function]

Returns the current character set of the stream *STREAM*, or T if *STREAM* is not run-encoded. If *CHARACTERSET* is non-NIL, the current character set for *STREAM* is set. For output streams this causes bytes to be written to the stream if *CHARACTERSET* is different from the current character set; for input streams it merely changes the reader's belief about the current character set. If *CHARACTERSET* is T, run encoding for *STREAM* is disabled—henceforth each character printed to the stream is printed as exactly two bytes (the character set and the character number).

Programs that wish to count characters or avoid worrying about character set changes can thus disable run encoding for a particular stream and count each character as two bytes. There is, however, a cost in file space.

Extensions to CHARCODE

CHARCODE has been extended to allow specifying NS Characters.

CHARCODE has been extended to allow the specification of 16-bit NS characters in multiple character sets. It also uses two new variables, CHARACTERNAMES and CHARACTERSETNAMES, so characters and character sets can be specified symbolically. The new definition is the following:

(CHARCODE *CHAR*)

[NLambda Function]

Returns the character code specified by *CHAR* (unevaluated). If *CHAR* is a one-character atom or string, the corresponding character code is simply returned. Thus, (CHARCODE A) is 65, (CHARCODE 0) is 48. If *CHAR* is a multi-character litatom or string, it specifies a character code as described below. If *CHAR* is NIL, CHARCODE simply returns NIL. Finally, if *CHAR* is a list structure, the value is a copy of *CHAR* with all the leaves replaced by the corresponding character codes. For instance, (CHARCODE (A (B C))) => (65 (66 67)).

If a character is specified by a multi-character litatom or string, CHARCODE interprets it as follows:

CR, SPACE, etc. The variable CHARACTERNAMES contains an association list mapping special litatoms to character codes. Among the characters defined this way are CR (13), LF (10), SPACE or SP (32), ESCAPE or ESC (27), BELL (7), BS (8), TAB (9), NULL (0), and DEL (127). Examples: (CHARCODE SPACE) returns 32, and (CHARCODE CR) returns 13.

CHARSET,CHARNUM, CHARSET-CHARNUM If the character specification is a litatom or string of the form CHARSET,CHARNUM or CHARSET-CHARNUM, the character code for the character number CHARNUM in the character set CHARSET is returned. CHARSET is either an octal number, or a litatom in the association list CHARACTERSETNAMES (which defines GREEK, CYRILLIC, etc.). CHARNUM is either an octal number, a single-character litatom, or a litatom from the association list CHARACTERNAMES. Examples: (CHARCODE 12,6), (CHARCODE 12,SPACE), (CHARCODE GREEK,A) and (CHARCODE ^GREEK,A)

Note that if CHARNUM is a single-digit number, it is interpreted as an octal character code, *not* as a character. Thus (CHARCODE GREEK,3) denotes the fourth character in the Greek character set, not the character "3" in that character set.

^CHARSPEC If the character specification is a litatom or string of one of the forms above, preceded by the character "^", this indicates a "control character," derived from the normal character code by clearing the seventh bit (100Q) of the character code (normally set in alphabetic characters). Example: (CHARCODE ^A)

#CHARSPEC (8-bit character codes) If the character specification is a litatom or string of one of the forms above, preceded by the character "#", the eighth bit (200Q), normally zero for 7-bit ASCII characters, is set. This is the way to get character numbers greater than 127. ^ and # can both be set at once. Examples: (CHARCODE #A), (CHARCODE #^GREEK,A)

Note: In Intermezzo (and in some other operating systems), characters with the eighth bit set were considered "meta" characters. In the Koto release, however, "meta" means character set 1, and the meta key produces characters with the 400Q bit set, not 200Q.

Internals

Note: The information in this section is advisory only. There is no guarantee of back-compatibility in future changes.

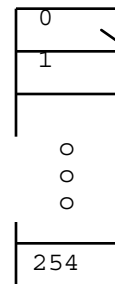
Structure of Font Descriptors

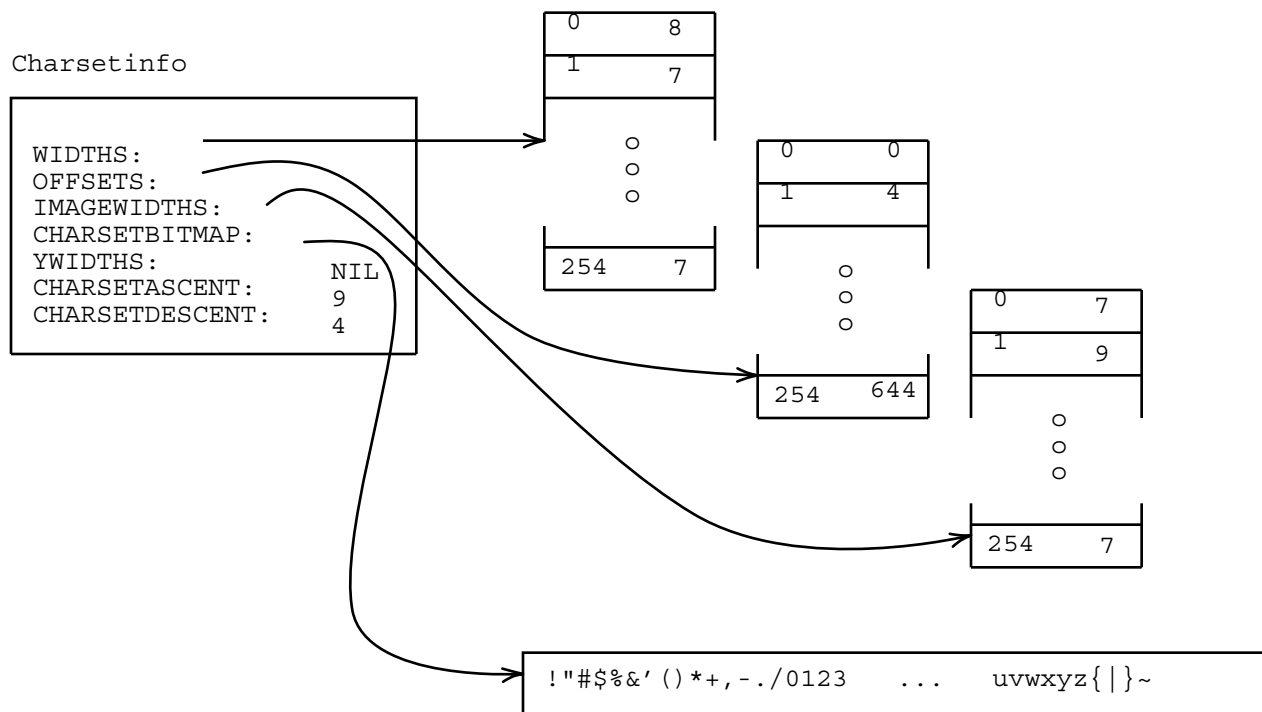
To incorporate the increase in information contained in font descriptors due to NS characters, the structure of font descriptors has changed. The current structure is as follows.

Font Descriptor

```
FONTFAMILY:      Modern
FONTSIZE:        12
FONTFACE:        (Medium Regular Regular)
ROTATION:        0
FONTDEVICE:      RandomDIGdevice
\SFAscent:       10
\SFDescent:      4
\SFHeight:       14
FONTCHARSETVECTOR:
```

CharsetVector





Description:

Font Descriptors:

The figure represents the logical structure of a font descriptor. Clarification of field values:

- Face: a list containing the weight, slope and expansion. (expansion is always regular)
- Rotation: degrees of rotation
- Device: name of the device for which this font is created
- Ascent: the maximum distance above the baseline for any character in this font.
- Descent: the maximum distance below the baseline for any character in this font. (always positive)
- Height: maximum total height of any character in this font. Equals Descent+Ascent.
- FontDeviceSpec: this is the font specification actually used to create this font after coercions. Thus, if the fontcreate method substituted something other than the original arguments to fontcreate, then this field shows the real contents of the font descriptor, while the family (etc.) fields contain the ostensible (pre-coercion) contents. For instance, display font substitutions occur in \CREATECHARSET.DISPLAY according to the variable MISSINGDISPLAYFONTCOERCIONS.
- OtherDeviceFontProps: available to the implementation of each stream. Unexamined by the system.
- Charsetvector: this is a ptr block pointing to the individual charsetinfo's for each of 255 charsets. These are either NIL or a ptr to the charsetinfo. In the diagram, only the charsetinfo for charset 0 is present. When a fontdescriptor is created the charsetvector will already be present.

Charsetinfo:

A charsetinfo contains all the metrics for a single character set (255 characters) of the font.

Widths: a block of SMALLP's which gives the width in device units for each character (how much to advance the stream xpos after printing this character). This field must be present no matter what the stream type.

Offsets: integer offsets into the bitmap field, showing the xposition of the beginning of the bitmap for this character. Many streams (especially hardcopy streams) have no need of a bitmap or offsets.

Imagewidths: This is the width of the image of this character. Often this will be the same as the width, but it can be less or greater. The field must always be valid.

Bitmap: The bitmap for the characters in this charset. [optional]

YWidths: currently unused, but for forward compatibility, make this be a block giving the y distance to move for each character (i.e. carriage return should move the y position the height of the font.) [**?* is this a positive distance]

Ascent: max ascent in the character set.

Descent: max descent in the character set.

Widths, Offsets, YWidths and Imagewidths are instances of the result of CREATECSINFOELEMENT (see below).

Exported Macros and Functions

(\CHARSET *CHARCODE*)
returns the character set (upper 8 bits) of *CHARCODE*.

(\CHAR8CODE *CHARCODE*)
returns the offset of *CHARCODE* within the character set (the low 8 bits of the character)

(\CREATECHARSET *CHARSET FONT NOSLUG?*)
the createcharset method (determined by the value of the variable IMAGESTREAMTYPES) is called. *NOSLUG?* determines the result if the createcharset method returns NIL. If *NOSLUG?* is NIL, then a "slug" charsetinfo (all characters have a slug (black rectangle) as their image) is returned, otherwise NIL is returned. [This needs improvement, to control how a slug is build. Currently \BUILDSLUGCSINFO is presumed to know how to build a slug for all imagestreams.]

(\GETCHARSETINFO *CHARSET FONTDESC NOSLUG?*)
returns the charsetinfo for *CHARSET* (0..254) from *FONTDESC*. Calls \CREATECHARSET if the charsetinfo wasn't cached already.

(\SETCHARSETINFO *CHARSETVECTOR CHARSET CSINFO*)
will install *CSINFO* as the charsetinfo of (smallp) *CHARSET* in *CHARSETVECTOR*. Since \CREATECHARSET calls \SETCHARSETINFO directly, it usually need not be called.

(\CREATECSINFOELEMENT)
creates a word block for installing as a widths (imagewidths, offsets) field in a csinfo.

(\FGETWIDTH *WIDTHSBLOCK CHAR8CODE*)
returns the smallp width at index *CHAR8CODE*. e.g. (\FGETWIDTH (FETCH (CHARSETINFO WIDTHS) OF csinfo) 55)

(\FSETWIDTH *WIDTHSBLOCK CHAR8CODE WIDTH*)
sets the smallp width at index *CHAR8CODE*.

(\FGETCHARWIDTH *FONTDESC CHARCODE*)
returns the width of any character without having to explicitly fetch the correct charsetinfo for the character set of the character.

(\FGETIMAGEWIDTH *FONT CHARCODE*)
analogous to \FGETWIDTH but for imagewidths (the width of the character image rather than the amount the xposition should be incremented when printing this character.)

(\FGETCHARIMAGEWIDTH *FONT CHARCODE*)
analogous to \FGETCHARWIDTH but for imagewidths.

(\FGETOFFSET *OFFSETBLOCK CHAR8CODE*)

analogous to \FGETWIDTH but for offsets (the position in the bitmap for this character set where the image for this character begins.)

(\FSETOFFSET *OFFSETSBLOCK CHAR8CODE OFFSET*)

sets the smallp offset at index *CHAR8CODE*.

1109 vs 1188 Floating Point Benchmark Results

Jan Pedersen
28 Aug. 1986

The timing results below compare the performance of an 1109 vs an 1188 on a suite of floating point benchmarks. The desire was to measure as closely as possible, using TIMEALL, the relative speeds of various arithmetic opcodes. No attempt was made to benchmark a "real" (e.g. linear algebra) application.

The 1109 was running a lispcore sysout(Makesysdate "21-Aug-86"), a real memory size of 7167 pages, and a set of Weitek floating point chips.

The 1188 was running a lispcore sysout(Makesysdate "25-Aug-86"), a real memory size of 7424 pages, and no floating point hardware, but microcode support for several boxed and unboxed floating point opcodes.

Both boxed and unboxed opcodes were benchmarked. Most benchmarks were a tight loop with the opcode evaluated 10,000 times. The block floating point opcodes were evaluated 1,000 times on arrays of size 100 (for a total of 100,000 arithmetic operations). Some of the boxed opcodes produced no garbage since they returned one of their inputs as an output, or returned T or NIL.

The 1188 had no microcode support for the block opcodes (they ran in lisp using scalar unboxed opcodes).

Cpu time and GC time are recorded separately for the boxed opcodes. Cpu time and CPU less the CPU time for an empty loop are recorded separately for the unboxed opcodes.

NA stands for Not Applicable.

Boxed Float Results (Time in seconds)

Opcode	1109		1188		Ratio	
	Cpu	Gc	Cpu	Gc	Cpu	Gc
-----	---	--	---	--	---	--
FPLUS	.98	(2.35)	1.02	(2.05)	.96	(1.15)
FDIFF	.98	(2.35)	1.03	(2.05)	.96	(1.15)
FTIMES	.99	(2.35)	1.17	(2.05)	.85	(1.15)
FQUOT	1.36	(2.34)	1.19	(2.05)	1.14	(1.15)
FGREATP	.304	(0.0)	.267	(0.0)	1.14	(NA)
Function	1109		1188		Ratio	
	Cpu	Gc	Cpu	Gc	Cpu	Gc
-----	---	--	---	--	---	--
FABS	2.1	(2.33)	2.14	(2.03)	.98	(1.15)
FMINUS	1.13	(2.33)	1.11	(2.04)	1.02	(1.14)
FIX	6.56	(0.0)	5.85	(0.0)	1.12	(NA)
FMAX	1.15	(0.0)	1.04	(0.0)	1.10	(NA)
FMIN	1.14	(0.0)	1.02	(0.0)	1.12	(NA)

Unboxed Float Results (Time in seconds)

Opcode	1109		1188		Ratio	
	Cpu	(- empty)	Cpu	(- empty)	Cpu	(- empty)
-----	---	---	---	---	---	---
Empty lp	.109	(NA)	.097	(NA)	1.12	(NA)
UFPLUS	.244	(.135)	.363	(.266)	.67	(.508)
UFDIFF	.244	(.135)	.362	(.265)	.67	(.509)
UFTIMES	.26	(.151)	.515	(.418)	.50	(.361)
UFQUOT	.616	(.507)	.533	(.436)	1.16	(1.16)
UFGREATP	.235	(.126)	.206	(.109)	1.14	(1.16)
UFABS	.178	(.069)	.161	(.064)	1.11	(1.08)
UFMINUS	.179	(.07)	.161	(.064)	1.11	(1.09)
UFIX	.213	(.104)	.205	(.108)	1.04	(.963)
UFMAX	.235	(.126)	.206	(.109)	1.14	(1.16)
UFMIN	.231	(.122)	.206	(.109)	1.12	(1.12)
BLKPLUS	.39	(.281)	6.73	(6.63)	.058	(.042)
BLKDIFF	.384	(.275)	5.71	(5.61)	.067	(.049)
BLKTIMES	.39	(.281)	7.63	(7.53)	.051	(.037)
POLY	.45	(.341)	4.92	(4.82)	.091	(.071)

Summary

- a.) Unboxed operations are a factor of ten faster than boxed operations across the board.
- b.) On an 1109 the block opcodes yield another factor of five to ten.
- c.) For scalar operations, the 1188 is never worse than .36% of the 1109, and never better than 1.16% of the 1109.
- d.) The 1188 was actually faster than the 1109 for several unboxed opcodes -- and generally faster for the boxed opcodes.
- e.) The 1109's floating point hardware really comes to the fore in the block opcodes. Unfortunately, with the exception of polynomial opcode, these opcodes are rarely used.

J.P.

Freemenu Internal Documentation

All names are in the Freemenu package (except old names, which begin with \fm.)

The Freemenu Description Language:

A freemenu description is defined by:

[] optional
{ } group of things for some other operator
* 0 or more
+ 1 or more
| or
literals in bold

menu-desc --> menu-element ; can be passed to **freemenu**

menu-element --> item-desc | group-desc

item-desc --> ({item-prop value}* **:label** item-label {item-prop value}*)

item-prop --> one of the prop keywords described in the user doc

item-label --> string | bitmap | imageobj

group-desc --> (**[group]** [prop-spec] {menu-element}*)

prop-spec --> (**:prop** {group-prop value}*)

group-prop --> one of the group property keywords described in the user doc

Group Formats:

The possible formats are:

:column - the elements in this group are layed out vertically, top to bottom, flush left

:row - the elements in this group are layed out horizontally, left to right, along the same baseline

:table - the elements in this group are layed out vertically, top to bottom, flush left. additionally, the second element of each direct subgroup is positioned at the same horizontal location. the third element of each direct subgroup is positioned at the same horizontal location. and so on, such that each element of each subgroup is both in a row and a column.

:explicit - each element of this group has a :left and :bottom property specifying its position. if the group property :coordinates is :group, then the values of the :left and :bottom property are relative to the lower left corner of the group, and if :menu (the default) they are relative to the lower left corner of the topmost group.

Group Format Defaults:

The format of a group of menu-elements can be specified in the prop-spec for that group. If it is not, the following rules apply:

- The default format for a group-desc passed directly to **freemenu** is :column.
- When formatting a group in :column format, the default format of each sub group is :row.
- When formatting a group in :row format, the default format of each sub group is :column.
- When formatting a group in :explicit format, the default format of each sub group is :explicit
- When formatting a group in :table format, the default format of each sub group is :table-element, which signals the formatter to horizontally align each element in the sub group with the other elements in the table. If the format is specified for a sub group, the elements of that sub group will not be aligned with the other elements in the table.

The Formatter:

Entry point:

The entry point to the freemenu formatter is `\\fm.format (fm::format)`. It takes a description of the menu to be formatted and returns a group hierarchy structure. In the current version of freemenu, the group structure is just an alist of group id's and properties, with the topmost group first. The formatter takes the following arguments:

description : a menu-element as defined above

format : the format to be used to lay out this group

font : the default font for each item in this group, must be a fontdescriptor

left, bottom : the lower left corner of the group. format everything relative to this position.

row-space : the number of pixels to leave between rows, that is, the space to leave between elements in a column

column-space : the number of pixels to leave between columns, that is, the space to leave between elements in a row

mother : the mother group of the one being passed for formatting. in the current version, this is the ID of the mother group, not the group itself.

The remaining arguments are optional. In the current version they are not specified, but they are SET by the guy who processes the group prop-spec. They might want to be specified in later versions:

id - the ID to use for this group

props - the prop-spec to use for this group

Group prop-spec:

The macro `\fm.setupprops` processes the group prop-spec in the description and fills in the slots in group accordingly (currently plist format). At the same time, it sets the arguments above from the values specified in the prop-spec, thus overriding the passed-in default. When the formatter is done with the current group, the function return, and thus all of the arguments are popped off the stack, and the previous state is now dynamically visible. This is how the formatter keeps track of format prop state, and pops back to the previous state when done formatting a group.

`\fm.setupprops` takes a group-spec and a list of group props to set. It generates code that will fill in the group props information, and then sets the format state arguments for the props that are in the list of group props to set.

The `:left` and `:bottom` group props in the prop-spec specify offsets for the entire group from where the formatter would otherwise position the group. This is similar to the way `:left` and `:bottom` item props specify offsets for an item that is automatically formatted (as opposed to explicitly positioned in the menu description).

Group boxing:

The macro `\fm.checkforbox` looks in the props to see if the group is boxed, and if it is, it adjusts the left,bottom position of the group to allow for the width of the box and the boxspace ($\text{boxoffset} = \text{box-width} + \text{box-space}$).

Then the elements in the group are layed out normally.

Finally, the macro `\fm.updateforbox`, if the group is boxed, does the following: save the calculated extent of the group as the interior region of the group, and then adjusts the region to include the box, and saves this region as the region property of the group.

Layout routines:

The formatter calls one of the layout routines (`\fm.layout-column` `\fm.layout-row` `\fm.layout-table` `\fm.layout-explicit`) to lay out the elements in the group. The layout routines provide the real guts of the formatting. These functions return multiple values:

1. list of items in all the elements layed out
2. list of groups in all the elements layed out
3. extent region of all the elements layed out
4. list of id's of all the subgroups layed out (this one would go away if had real group structure, instead of flat alist)

Layout algorithm:

Here is a description of the algorithm used in laying out a column of elements. The other routines follow the same procedures, but operate in a different dimension or have other overhead (like layout-table).

```

bind extent ; the region the elements in this group occupy
    itemlist ; a flattened list of items in this group
    grouplist ; a flattened list of groups in this group
    subgroupids ; a list of id's of groups in this group
    element-position ; the position of the next element formatted
    element ; temp for hanging onto the element newly formatted

iterate through each element in the group description
    if the element is a group description:
        set element to call format on the group description
        extend: extent, itemlist, grouplist, subgroupids with the results
    otherwise:
        set element to create an item from the item description
        extend extent, itemlist with the item

    increment element-position by the size of element and extra space

return extent, itemlist, grouplist, subgroupids

```

Putting it all together:

With the information returned by the layout routine in the hands of the formatter, create a new group structure from the group prop-spec and extent, itemlist, and subgroupids. Add this group to the front of the list of groups layed out, and you get a list of groups for the description just formatted.

Freemenu Data Structures:

A Freemenu is currently a window, with all of the necessary properties set to make it behave as a menu when it is open. Eventually a Freemenu probably wants to be an independent structure, which can be enclosed in different display mechanisms, like windows, image-objects, pop up managers, etc.

There are three main data structures composing a Freemenu:

ITEM - Instance of the datatype freemenuitem, one for each item in the menu. The macro itemprop provides access to fetching and replacing fields in the datatype. The macro %itemprop is an internal version of the same macro which doesn't type check the item and requires the field (property) name to be provided explicitly (not bound to some variable) in the call.

GROUP - A list structure describing a group of items in the menu. List format is (<group-id> <group-type-identifier> {<prop> <value>}*). The <group-id> is used for ASSOC purposes on the list of all groups. The <group-type-identifier> is checked by the macro

group-p to ensure that this list is a valid freemenu group. The macro groupprop provides access to getting and setting the props in the cddr of the list.

NWAY - A list structure describing an nway collection in the menu. The list format is the same as that for groups. nway-p and nwayprop are analogous to group-p and groupprop.

A list of all the items in the menu is stored on the ITEMS window property.

A list of all the groups in the menu is stored on the GROUPS window property.

A list of all the nway collections in the menu is stored on the NWAYS window property.

The functions get-item, get-group, and get-nway take an id and search the appropriate list for a matching freemenu item, group, or nway, respectively.

Additionally, many of the freemenu functions, like redisplay-menu, depend on being able to use a flat list of objects in the menu, either items, groups, or nway collections.

GC Overview:
February 6, 1986
Jan Pedersen & Greg Nuyens

Reference counts are not contiguous with their objects, but rather are kept in three tables: \HTMAIN (the hashed main ref count table), \HTCOLL (the collision table), and \HTBIGCOUNT (the big ref count table). \HTMAIN is a hash table, where the hash function is based on the address of the object, while \HTCOLL and \HTBIGCOUNT handle two aspects of hash table overflow.

Since most (?) objects have refcount 1 (e.g. cons cells of a list), they are not explicitly represented; if an object (of a refcountable type) is not present in the tables, its refcount is 1.

If several objects hash to the same entry in \HTMAIN, then the entries are kept in a linked list of ref count entries in \HTCOLL.

If an object has a refcount equal to \MAXHTCNT (63), its refcount is stored in a "big" refcount table (\HTBIGCOUNT).

The Lisp function \HTFIND can handle all cases and is the punt function for the opcode (GCREF), which handles only the simplest case of no collision and no big refcount. Many opcodes call GCREF as a microcode subroutine and if GCREF punts, an entry is made in the table \HTOVERFLOW. Before opcode completion, the microcode calls the Lisp function \GC.HANDLEOVERFLOW, which processes \HTOVERFLOW by explicitly calling \HTFIND on each entry.

\HTMAIN is a locked down table of 32K word sized entries (or 64K bytes).

\HTCOLL is a paged table of 32K double word sized entries (or 128K bytes), where the first word in a pair is a ref count entry and the second is a 16 bit offset to the next entry in the chain.

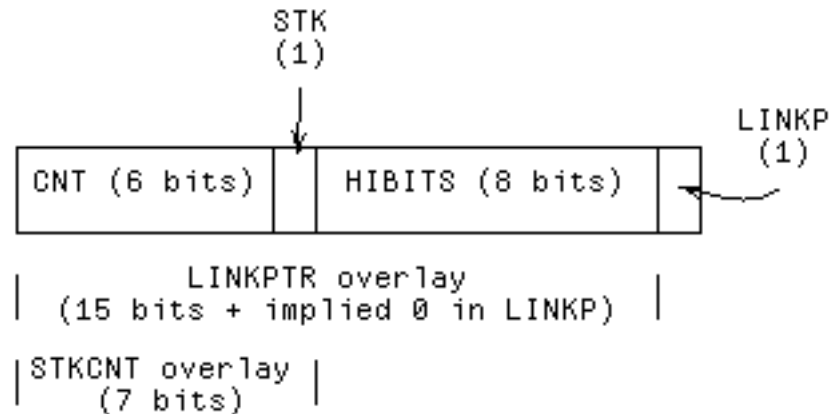
\HTBIGCOUNT is a linearly searched table of big ref count entries; new pages are allocated as needed.

Hashing:

Given an address, the hash function computes a 15 bit offset into \HTMAIN by logically shifting the low 16 bits of the address right one bit.

GC record structure:

Each entry in \HTMAIN is a word (16 bits) long, and is a record of type GC. Its structure is the following:



REFCOUNT TABLE ENTRY

CNT is the refcount (0 to 63, excluding 1)

STK is on if this object is referenced from the stack.

HIBITS contains the top 8 bits of the PTR address of the object (the offset at which you found this entry gives you the lower 15 bits (only even objects are refcounted)).

LINKP is on if this entry is a pointer to the collision table.

Organization of \HTCOLL:

\HTCOLL points to a double word entry, the first word of which is a 16 bit offset from \HTCOLL where the first entry on the linked free list may be found. The second word is a 16 bit offset from \HTCOLL where the first free double word in sequential storage may be found.

At startup the first word is zero, indicating an empty free list and the second word is four, indicating that the first free double word in sequential storage is at offset four from the top of \HTCOLL.

A new entry is allocated by first looking at the free list. If the free list is empty, a new entry is allocated from sequential storage.

It is an error to allocate double words from sequential storage at offsets greater than or equal to \HTCOLLTHRESHOLD (65528 or 177770Q) -- if this occurs the garbage collector is turned off.

The double word entries in \HTCOLL are similar to those in \HTMAIN in that the first word in a pair is an instance of record type GC. The adjacent word is a 16 bit link offset (field name NXPTR) from the top of \HTCOLL to the next double word entry in this collision chain. A NXPTR value of zero indicates end-of-chain.

Algorithm:

\HTFIND is called with two arguments, PTR (a pointer to the object) and CASE (one of \ADDREFCASE, \DELREFCASE, \SCANREFCASE or \UNSCANREFCASE). The \SCANREFCASE means mark the object as referenced on the stack, likewise \UNSCANREFCASE. \HTFIND returns PTR if the result is zero ref count, else NIL.

First, \HTFIND hashes the address into a 15 bit offset from the top of \HTMAIN, pointing at a refcount table entry. If the resulting entry is empty (all bits 0) then the macro .NEWENTRY. handles it.

.NEWENTRY. does the following:
updates the HIBITS.
and by case does:

- \ADDREFCASE sets CNT to 2 (since its absence from the table implied previous refcount 1)
- \DELREFCASE leaves the CNT at 0 and returns the PTR
- \SCANREFCASE Sets count to 1 and sets the STK bit.

If the entry from the hash has the same HIBITS then this main table entry is the entry for PTR, so the macro .MODENTRY. handles it.

.MODENTRY. does the following:

if CNT is \MAXHTCNT, then this entry is superceded by an entry in the big refcount table (\HTBIGCOUNT), so call \GC.MODIFY.BIGREFCNT.

Otherwise, by refcount case:

- \ADDREFCASE if incrementing CNT makes it equal to \MAXHTCNT, then call \GC.ENTER.BIGREFCNT, otherwise just increment CNT.
- \DELREFCASE decrements CNT
- \SCANREFCASE sets STK.
- \UNSCANREFCASE clears STK.

If resulting refcount is 1 and no STK (STKCNT = 2), return T signalling removal.

If the HIBITS don't match, then a new collision has occurred. The collision resolution is basically to construct a linked list of refcount table entries out of cells from \HTCOLL

New collision:

- Get two double word entries (LINK and PREV) from \HTCOLL by calling the .GETLINK. macro.
- Make the ref count contents of PREV equivalent to the ref count contents of the entry in \HTMAIN and link PREV to the next double word entry, LINK.
- Smash the \HTMAIN entry so that the LINKPTR overlay is an offset to PREV.
- Mark LINK as empty, and the end of a chain.
- proceed with LINK as an empty entry (.NEWENTRY.)

where .GETLINK. does the following

- fetch the FREEPTR field of \HTCOLL (should be an offset to the front of the free list).
- if FREEPTR is non-zero, fetch the first free cell and update the FREEPTR field of \HTCOLL
- else fetch the NEXTFREE of \HTCOLL (should be an offset to the first free double word in sequential storage). If NEXTFREE is not GEQ to \HTCOLLTHRESHOLD, allocate that double word and increment NEXTFREE by 2.

If the entry from the hash has LINKP set, then a collision has already occurred and the entry is a pointer to a chain in the collision table. The LINKPTR overlay is used as a (16 bit) offset into \HTCOLL (the first 15 bits of the entry are used as is, with the last bit (LINKP) masked to zero). Note that the LINKP test should occur before testing HIBITS, but we place the explanation here since this case is more complex than a new collision.

So LINKPTR is an offset to the beginning of a chain of double word entries in \HTCOLL. This chain is searched sequentially (by following NXPTR offsets down the chain), resulting in either finding an entry for PTR, or hitting the end of the chain (NXPTR offset equal to zero).

If an entry is found, then .MODENTRY. handles it as before except that if the refcount goes to zero the link is deleted from the chain (by .DELLINK.).

.DELLINK. takes three args, (LINK -- the link to be deleted, PREV the previous link in the chain (can be NIL), and ENTRY -- the \HTMAIN table entry)
 -- If PREV, then update PREV to point to the link following LINK, else update ENTRY to point to the link following LINK.
 -- Place LINK back on the free list (.FREELINK.)
 -- If ENTRY now points to a chain one entry long, then update ENTRY to have the refcount table entry contents of the remaining link, and free the remaining link.

.FREELINK. takes one arg, the LINK cell to be freed
 -- zero LINK
 -- place LINK on free list of \HTCOLL by update the first word (field name FREEPTR) of \HTCOLL

If no entry in the chain is found for PTR, then allocate an entry (.GETLINK.) and put it on the end of the chain. Treat the new LINK as an empty entry (.NEWENTRY.).

Stay tuned for further updates...

February, 1989

Movement of Guaranteed Type Numbers, Addition of hashing MISCN subops.

Summary

The type numbers of several Lisp datatypes were moved down into the range that is "known to the microcode," to allow me to write C support for hashing.

Overview

Certain lisp type numbers must be known to the underlying implementation (microcode on 1186's, the C emulator on Suns). One obvious example is that the emulator must be able to detect SMALLPs, so it can do arithmetic quickly.

There is a range of type numbers that are allocated very early in the load-up process, so that they are assigned known numbers. After those "well-known" types, the type numbers for hunked storage are allocated. After that, type numbers are allocated to types as the loadup progresses, in whatever order they are defined.

The New Requirement

I implemented MISCN sub-opcodes for CL: SXHASH, CL::EQLHASHBITSFN, IL: STRINGHASHBITS, and IL: STRING-EQUAL-HASHBITS. CL: SXHASH has special-case code for several data types that were not in the "well-known" range: RATIONAL, COMPLEX, PATHNAME, and BIGNUM. I needed to move the type numbers for those types down.

The New "Well-Known" Type Numbers

The well-known type numbers are defined from the list \BUILD-IN-SYSTEM-TYPES, which is defined in the file LLDATATYPE. Listed below are the old and new type number assignments:

Pre-existing	Newly-Added
1 SMALLP	
2 FIXP	
3 FLOATP	
4 LITATOM	
5 LISTP	
6 ARRAYP	
7 STRINGP	
8 STACKP	
9 CHARACTER	
10 VMEMPAGEP	
11 STREAM	
12 BITMAP	
13 COMPILED-CLOSURE	
14 ONED-ARRAY	
15 TWOD-ARRAY	
16 GENERAL-ARRAY	
	17 BIGNUM
	18 RATIO
	19 COMPLEX
	20 PATHNAME

Changes to the C emulator

Moving those four types into the well-known range had the effect of moving the array hunk type numbers up. The C coded garbage collector had several hunk type numbers hard-coded into it (as hex constants!!). I added definitions for all the type numbers in use (both well-known and hunk) to LISPTYPES.H, and changed the GC code (in GCRECLAIMCELL.C, as I recall) to accomodate it.

The new hashing MISCN sub-opcodes

The source for the new opcodes is in sxhash.c. The document {Eris}<Lispcore>Internal>Doc>Opcodes.TEdit has been updated to reflect their addition. I have also reserved opcodes for CL:VALUES and CL:VALUES-LIST.

The Lisp definitions for the functions CL: SXHASH, CL::EQLHASHBITSFN, IL:STRINGHASHBITS, AND IL:STRING-EQUAL-HASHBITS are now written to use the MISCN sub-opcode appropriate; the UFNs are defined in the same files as the original functions, CMLHASH and LLARRAYELT.

Hello 11.0

program: [eris]<lisp>Harmony>mesa>HelloDlion.boot
documentation: [eris]<lisp>Harmony>Doc>Hello.tedit

Hello 11.0 is a modified version of the Othello 11.0 utility, which is used for managing Pilot disk volumes. It offers most of the Othello commands (documented in the Mesa Users Guide), and adds a few commands which are useful when using Interlisp-D on a Dlion.

Hello is a .boot file, which can be loaded onto a Dlion disk using the Fetch Boot File command of Othello. It can also be booted off of a floppy disk.

Loading Interlisp From a File Server to a Logical Volume:

When Hello starts up, it prints out some information about the machine it is running on, including the Dlion's host number and memory size, and then prints the prompt ">", to indicate it is ready to receive a command. The "Online" command, which is automatically printed, tells Hello to bring the physical disk on-line. (Note: all user input is underlined, including confirming carriage-returns)

```
Hello 11.0 of 6-Sep-84 10:14:03
Processor = ...
Memory Size=1536 bytes
>Online
Drive Name: RD0
```

Before fetching a lisp sysout from a file server, it is necessary to open a connection to a file server, and login. Note: Hello currently cannot communicate with NS file servers (ones with colons in their names, such as "Phylex:").

```
>Open
Open Connection to ERIS
>Login
User: Sannella
Password: *****
```

In order for a lisp sysout to run, it needs to have a special "initial microcode" file installed. This microcode only needs to be installed once for each Dlion.

```
>Initial Microcode Fetch
Drive Name: RD0
File Name: [eris]<lisp>Harmony>Basics>Lisp11SAx000Initial.db
Formatting...Fetching...Installing...done
```

Now, fetch the Interlisp sysout file [eris]<lispcore>next>Full.sysout and store it on the logical volume named "Lisp". Depending on the size of the sysout file, and the load on the ethernet, this can take 5-10 minutes.

```
>Lisp Sysout Fetch
Logical volume name: Lisp
Lisp sysout file name: [eris]<lisp>Harmony>Basics>Full.sysout
Fetching....
```

Before running a lisp sysout, it is necessary to "expand" the file containing the sysout to the full size of the logical volume. This will allow Interlisp virtual memory to grow as Interlisp needs more space. If this is NOT done, there can be problems with Interlisp on large-memory Dliions. Eventually, the low-level Interlisp virtual memory management system will be improved, so this will not be necessary.

```
>Expand Vmem file
volume to expand: Lisp
```

Finally, the Interlisp image on volume "Lisp" can be started with the boot command.

```
>Boot
```

```
Logical volume name: Lisp
Boot Lisp from this volume? YES
```

Copying Interlisp From One Logical Volume to Another:

A very useful facility that Hello offers is the capability of copying an Interlisp sysout from one logical volume to another. This is much faster than retrieving a sysout over the ethernet. Many people like to keep a "virgin" sysout on their DLion, and reload from that. Here is how it is done: Load a sysout onto a volume named "BootLisp", as specified above. This sysout will be the "virgin" sysout used to re-initialize other Interlisp volumes. This volume should not be expanded with the "Expand Vmem file" command, and should never be booted.

```
>Open
Open Connection to ERIS
>Login
User: Sannella
Password: *****
>Initial Microcode Fetch
Drive Name: RD0
File Name: [eris]<lisp>Harmony>Basics>Lisp11SAx000Initial.db
Formatting...Fetching...Installing...done
>Lisp Sysout Fetch
Logical volume name: BootLisp
Lisp sysout file name: [eris]<lisp>Harmony>Basics>Full.sysout
Fetching....
```

To re-initialize the Interlisp volume "Lisp", copy the sysout file from the volume "BootLisp". This takes about a minute.

```
>Copy Lisp from Another Volume
Volume to copy from: BootLisp
Volume to copy to: Lisp
```

Now, expand and boot the "Lisp" volume as if the sysout had been loaded from a file server.

```
>Expand Vmem file
volume to expand: Lisp
>Boot
Logical volume name: Lisp
Boot Lisp from this volume? YES
```

Determining What is on Your Disk:

It is easy to forget exactly what sysouts are on what volumes of the DLion disk, or which Interlisp sysouts have been "expanded", etc. The "Describe" command prints out useful information about every volume on the disk.

```
>Describe
.
.
Volume Lisp (type=normal) 700 of 16200 pages free
  starting at physical address 4129
  Lisp sysout: [eris]<lispcore>next>Full.sysout (3-Aug-84 10:35:17)
.
.
```

In this example, we can tell what lisp sysout, with which creation date, is stored on the volume "Lisp". The fact that 700 of 16200 pages are free strongly indicates that the sysout has been expanded (expansion doesn't use ALL of the space).

Additional information on "How-To" can be found on:

{Eris}<Sybalsky>How-to>

such as:

Installing New 1108 ucode
Running AR cleanup
Seding info to Sales
Writing AR test cases

Subject: How to write the release notes
To: James
cc: Sannella.pa

How to write a release note for a fixed AR:

When an AR is fixed, it is important to write a release note for the AR, even if it is just a note saying "Release Note: none needed". There is no better time to do this than when the problem is still fresh in your mind. And there is no better person to do this than the implementer, who knows exactly how important the AR fix is.

Does this AR affect the next release? There are some ARs which are created and closed during the course of development. However, it is hard to tell just from the text of the AR. It is important to put this information in the AR, so the Release master can tell that it shouldn't be collected. The note "Release Note: Don't release -- development bug" is better than nothing.

Does this AR affect outside users? If an AR just affects internal users, put a release note "Release Note: Don't release -- internal info" in the AR.

If there is info in the AR that the outside users should know, compose a release note. It doesn't have to be perfect, but the implementer is in a better position to know what needs to be told than anyone else.

A single release note item should talk about one piece of information. If a single AR contains or refers to multiple subjects, compose multiple release notes.

A release note contains a title, telling the substance of the item. Like AR subjects, the release note title should be short and as informative as possible (think of newspaper headlines). Include as many keywords as possible. For example "* New window functions: FOOBAR, BAZ" is better than "* New window functionality". When documenting bug fixes, try to make the titles "positive", saying "* FOOBAR arguments interpreted correctly" rather than "* Bug fixed where FOOBAR ignored argument".

When in doubt about whether a bug fix or new feature is worth documenting in the release notes, please document it. It is a lot easier for the release master to throw away information than to write it from scratch.

How to assemble individual items into the "Release Notes":

If the individual release note items have been written, the primary job is to sort them by subject and importance, and assemble the big document.

For the Harmony release, the "topics" were ordered loosely on the importance of the different areas for that release (a lot of changes were made to I/O, fonts, printing, so they came first). This order is not sacred.

Within each topic area, the items are sorted by importance to the user, a subjective decision. Major new functionality, and incompatible changes obviously want to come first.

Characteristics of the Xerox 1100 Machines upon which the Gabriel Benchmarks Were Performed

All three members of the Xerox 1100 family are custom microcoded processors. The Interlisp-D virtual machine is built around a compact 8-bit "bytecode" instruction set, the opcodes of which are implemented by a combination of microcode and macrocode. Not all bytecodes are supported directly in each member by microcode; the alternative is a trap out to a standard Lisp function. Above the level of the instruction set, all three members of the family appear identical to the Interlisp-D programmer. The implementation is such that a memory image can be compatibly run on any of the machines, without any change.

An Interlisp pointer is an address in a 24-bit virtual address space; a "quantum map" indexed by the high bits of the address provides information for type decoding. Additionally, litatoms (symbols) and immediate numbers (integers in the range of -2^{16} to $2^{16}-1$) live in a reserved portion of the address space; integers of larger magnitude (within the range -2^{31} to $2^{31}-1$) are "boxed"; floating-point numbers, which are in IEEE 32-bit format, are also boxed. All three machines have a 16-bit memory bus and 16-bit ALU; however, the bytecodes tend to hide the actual word size from the programmer. The virtual address space is broken down into units of 512-byte pages, and the three machines have different degrees of hardware assist for virtual memory management and instruction fetch.

Cons cells are cdr coded in a manner described in D. Bobrow and D. Clark, "Compact Encodings of List Structure", ACM Trans. on Prog. lang. and Systems, Vol 1 No 2, p266 October 1979. A cell of 32 bits is used to store a cons -- typically 24 bits for the car, and 8 bits for an encoding of the cdr. The encoding covers the four cases where (1) the cdr is NIL, or (2) the cdr is directly on the same page as the cons cell, or (3) the cdr is contained in another cell on the same page as the cons cell, or (4) the cons cell is itself a full indirect pointer, which can address an ordinary two-cell slot on any page (the space normally used for the car is used to address a 64-bit cell elsewhere; this is to allow for RPLACDs when there is no more free cells on the same page as the cell being updated). All cons cells are cdr-coded, independent of how they are created, and as a consequence the "average size" of such a cell is considerably less than 64 bits.

Strings and arrays are implemented as a fixed-length header, with one field pointing to a variable-length memory chunk taken from an area which is separately managed. To run some of the benchmarks, we used Interlisp's Common Lisp array utility package. Additionally, Interlisp permits the user to define new first-class fixed-length data types, with corresponding entries in the quantum map mentioned above; for example, a STREAM is implemented as a record structure with 19 pointer fields and assorted integer fields of 16 bits or less.

Garbage collection is patterned after Deutsch and Bobrow, "An Efficient, Incremental, Automatic Garbage Collector" CACM, July 1976. A reference count is maintained for every collectible pointer (in addition to immediate pointers, litatoms are not reclaimed in Interlisp-D). Updates to non-stack cells in data structures (i.e., the CAR slot of a CONS cell, or the value-cell of a global variable) require updates to the reference count. The reference counts are maintained separate from the objects in a hash table, which is generally very sparse; and the updating is normally done within the microcode that effects the update operations. Reclamations are

performed frequently, and involve scanning the stack area and augmenting the reference counts by a "stackp" bit; then scanning the reference count table reclaiming any entry which has a count of 0 and no reference from the stack (and possibly additional pointers whose reference count goes to zero as a result of such a reclamation); and finally re-scanning the table to clear the "stackp" bits. The scan through the reference count table looking for 0-count entries corresponds roughly to the scan of the marked-bits table in a Mark-and-Sweep collector; however, the scan of the stack is infinitesimal in time compared to a full "mark" phase, and thus a reclamation typically runs in well under a second.

The internal architecture of the stack is a variant of the "spaghetti stack" model described in Bobrow and Wegbreit "A Model and Stack Implementation of Multiple Environments, Comm. ACM, Vol. 16, No. 10, Oct. 1973, pp. 591-603. The stack area is currently limited to 128KB.

The particular configurations upon which the benchmarks were run are as follows:

Xerox 1100: (Dolphin). 4K words of 40-bit microstore; microinstruction time 180ns; hardware assist for macro-instruction fetch; hardware memory map for up to 8MB of virtual space; hardware stack (for stack tip); memory access is 1-to-4 words (64 bits) in about 2us. The particular unit used in the benchmarking runs had 1.8MB of real memory attached, but 2MB has been in standard delivery.

Xerox 1108: (Dandelion) 4K words of 48-bit microstore; microinstruction time 137ns; hardware assist for macro-instruction fetch; hardware assist for virtual memory management (memory map is kept in non-paged real memory); memory access is 1 non-mapped 16-bit word in 411ns, but a random 32-bit cell access in about 1.2us. The stack is held in real, non-mapped memory. The particular unit used in the benchmarking runs had 1.5MB of real memory attached.

Xerox 1132: (Dorado) 4K words of 34-bit high-speed ECL microstore; microinstruction time 64ns; hardware instruction fetch unit; hardware memory map for up to 32MB of virtual space; 4Kilowords of high-speed ECL memory cache permit memory access of one 16-bit word in 64ns, and a cache-reload of 256 bits takes about 1.8us (additional details on the cache and memory organization may be found in D. Clark, B. Lampson, and K. Pier: "The Memory System of a High-Performance Personal Computer", IEEE Transactions on Computers, vol C-30, no. 10, Oct 1981). The particular unit used in the benchmarking runs had 2MB of real memory attached.

Note that the benchmarks were not run on the 1108-111 (Dandelion), which has considerably more memory and control store than the basic 1108, and which also has a floating-point processor.

Interlisp-D Implementor's Reference

Filed as: {Eris}<LispCore>Internal>Doc>ImplManual.Tedit;22

Contents

- * Updating the InterLisp Reference Manual
- * Making a SYSOUT, the Loadup process
- * Writing Renamable Code
- * How Teleraid works
- * Low-level Storage Reference and Manipulation Functions
- * ABC, EXPORTS.ALL, etc.
- * SYSRECORDS
- * Installing new DLion microcode
- * DTigerness vs. DLionness
- * WHEREIS & the WHEREIS database
- * Installing New Op-Codes
- * Creating new DEVICES
- * VMEM.PURE.STATE
- * Using Dorado Mufflers & Manifolds (fold into opcodes.tedit?)

Chapters elsewhere:

- * The DLion Low Level Disk Drivers (see ImplManual-DlionDiskDriver.tedit)
- * XDE Tutorials (see ImplManual-XDE.tedit)

Updating the InterLisp Reference Manual [Sannella]:

Documentation files are kept on {eris}<LispManual>*.im. LispCore members are encouraged to modify these documentation files to reflect the changes made as the system is modified. Please be careful.

One important note: it is necessary to keep track of how the manual is changed, in order to provide a list of changes with the next revision of the manual. Therefore, I would strongly suggest that whenever anyone makes a significant change to the manual (adds/deletes a function definition, adds new arguments to a function, non-trivial rewording, etc.) that they send a short message to me (Sannella, not LispCore^).

The manual is stored in a large number of separate files, and it is difficult to know which file contains a particular function definition. Therefore, I have created a small package that will take an "IM Name" (a function, variable, property name, etc), create a TEdit window on the appropriate IM file, and position the TEdit cursor at the right place.

To use this, do

```
_(FILESLOAD (FROM LISPUSERS) IMNAME)
_(INSPECT.IM 'FOO)
```

INSPECT.IM uses the hash file package to search a hash file containing index information for the name FOO. If it is found, it will put up a pop-up menu listing references in different files. Selecting one of the references will move the cursor in the appropriate TEdit window (if there is an active TEdit window to the appropriate file), or create a new TEdit window to the appropriate file.

Sometimes, a particular name is defined as more than one "type" (function, variable, etc.). In this case, a pop-up menu will prompt you to declare which type you are interested in.

A somewhat more convenient way of using this facility, if you want to use it repeatedly, is to do

```
_(MAKE.IM.INSPECTOR)
```

This sets up an "IM Inspector Window", which contains a menu. Initially, this contains the single selection "Type an IM name", which prompts the user to type a name which will be looked up in the database. Below this window will appear type-selection and reference-selection menus, which do not disappear until another selection is made above them. This is hard to describe.... try it out. [It works great! -- LMM]

MAKING A LOADUP:

Loadups can currently only be made on Dorados. Command files, with extension "CM", are read by the Alto exec using the "@" command.

There are various command files on {Eris}<LispCore>CM>:

LoadFull.CM makes a LISP.SYSOUT and a FULL.SYSOUT from scratch.

LoadFullFromLisp.CM makes a FULL.SYSOUT directly from the LISP.SYSOUT.

LoadDemoFromFull.CM makes a DEMO.SYSOUT

[Note that the command files were modified to use standard cache partitions on Dorados (which are?). These are used to load the sysout that the renamed functions run in, and to save the sysouts. The old non-cached behavior can be had by using command files whose names begin with SLOW-. A complication this causes is that changes to the command files must now be made in two places, the SLOW- and caching versions.]

The first part of the command file runs MAKEINIT to create a file INIT.SYSOUT and then DLFIXINIT to make it dandelion bootable by merging in the dandelion microcode to create an INIT.DLINIT.

[FS: THE FOLLOWING INDENTED SECTION HAS BEEN CHANGED. THE CORRECT EVENTS FOLLOW BELOW:

The next part starts up the INIT.DLINIT (which will run on all machines), and calls LOADUP (HUGE). The function LOADUP (on the file APUTDQ which is merged in at MAKEINIT time) has directions on how to do various kinds of loadups. It determines what other files are in the default loadup.

If you have a special kind of loadup that you think should be supported for some applications, it is possible to add a separate clause to LOADUP and include that in the standard source.]

The next part starts up the INIT.DLINIT (which will run on all machines), and the CM script loads LOADUP.LISP and LOADFULL.LISP, which call the function LOADUP on the rest of the files in the LISP.SYSOUT and the FULL.SYSOUT, respectively. (LOADUP 'HUGE) is obsolete and should be deleted.

How MAKEINIT works:

Basically, all the storage-modifying functions are redefined so they make their changes to a file. After some initialization of the blank memory space in the SYSOUT file the normal LOAD code is run, but the effects take place in the new sysout file instead of in memory.

Modification of the low-level storage functions is done by the code in the file RENAMEFNS, based on information in the file FILESETS. The function DORENAME is called with the argument I (for INIT). DORENAME uses the RENAMETYPES variable to determine which files to get low level definitions from,

and how to rename them. `RENAMETYPES` also indicates how to create an `R` (for `READSYS`) type rename, which is used by `teleraid` to read definitions out of a `sysout` (or `sysout` file).

The end result of `DORENAME I` is a file `I-NEW`, containing the "remoted" definitions. This file is loaded and the "renamed" loaded code is run to load up the earliest parts of the system.

The basic goal of `MAKEINIT` is to make the `INIT.SYSOUT` capable of loading files over the ethernet and writing out (as with `LOGOUT` or `MAKESYS`) the resulting system. This minimal set of files to load is defined in `FILESETS` in the variables `0LISPSET` and `1LISPSET`.

Once these files are loaded, `INIT.SYSOUT` is written out. After having the dandelion microcode spliced into the memory image by `DLFIXINIT` the resulting `INIT.DLINIT` is run. The first thing done here is to run the "init expressions" of all the files which were loaded renamed. The expressions could not be evaluated remotely earlier in the init (indeed, the evaluator is not fully loaded at this point).

Things to watch out for: The data type `STREAM` must be the first datatype declared after `MAKEINIT` time. This means that no file loaded before `FILEIO` can declare a datatype. Packages are "turned off" in the early part of the init and symbols written into the `INIT.SYSOUT` are all package qualified. The file `PACKAGE-STARTUP` makes the switchover (located at the end of `1LISPSET`).

Now the rest of the files in `FILESETS` are loaded (those in `2LISPSET` and up). The greatest number of problems encountered after this point, aside from outright bugs, are dependencies of code on parts of the system which have not yet been loaded.

After this point the `LOADUP` function loads in the standard sets of files to make `LISP.SYSOUT` and `FULL.SYSOUT` (more detail?).

Writing Renamable Code:

Renamed code is used to build loadups and read the format of the resulting memory spaces (see sections above and below).

This is intended as a start at describing what you need to know to write renamable code.

`\COPY` is called to move something from the local memory space to the remote one.

`\UNCOPY` brings objects back.

`LOCAL` can be used to ensure that a function's effects occur in the local memory image. It inhibits renaming of forms inside of it.

`ALLOCAL` can be used to ensure that a function's effects occur in the local memory image. It inhibits renaming of forms inside of it.

`UNLESSRDSYS` takes two forms, the first to execute normally, and the second to be used when the function is renamed.

Since `DTEST` doesn't run renamed, code that is intended to run renamed should use `ffetch` and `freplace`

How Teleraid works:

Teleraid runs in two parts: a *server*, running in microcode on one machine (usually entered by pressing UNDO to a front panel error), and a *front-end*, running on another machine. The front end machine examines the other over the network using the server.

The teleraid server is actually a simple memory page server written in microcode. It transfers pages of memory (using PUP protocols) to the front-end machine. It is up to the front-end machine to understand the internal format of the other machine's memory. This happens through a reverse version of the init process. Rather than renaming functions to write onto a remote memory space (in a file), teleraid runs functions which are renamed to read a remote memory space (on another machine or in a file; you can teleraid a non-running sysout file).

A variation of the renaming scheme used to build the init is employed in Teleraid. A large number of functions in the system are renamed to call (at their lowest levels) the Teleraid page server on the other machine.

Since the lowest levels of the system can change between releases it is important to have the same sysout running on the two machines.

The actual file that contains teleraid's renamed functions is RDSYS. It is created automatically by DORENAME on the file RENAMEFNS and must be updated whenever low level representation is changed. Details of what to rename for teleraid are contained on the file FILESETS. The functions which get renamed are scattered all through the low level system files (but those are pointed to by FILESETS).

Internal Storage Reference Functions [Masinter, van Melle]:

There are a number of low-level functions for directly accessing memory as if it were an enormous array of 16-bit words, bytes, 32-bit cells, etc. In general, don't call any of these directly if you can help it. They are generally "unsafe" and can confuse your system in subtle ways if misused. Also, there are often alternatives that, if not completely safe, are at least less prone to error:

(A) Write functions or macros to do the accesses, and have them perform suitable type checking. See the `GETPUPWORD` and `PUTPUPWORD` macros, for example.

(B) Define a `BLOCKRECORD` to overlay a given data structure. This is much better than using `\GETBASE` et al if you are using fixed offsets—it is safer (less error-prone) and generally produces better code. With creative use of `LOCF` and `ACCESSFNS`, you can often avoid using explicit `\GETBASEs` altogether, and your code is much more readable. Also check out the `MESATYPES` package, written by Tayloe Stansbury, for producing such expressions from Mesa type declarations.

`(\ALLOCBLOCK NCELLS GCTYPE INITONPAGE ALIGN)`

The basic low level storage allocation function. *NCELLS* is the number of 32 bit cells to allocate. *GCTYPE* is an integer, usually stated as the value of either `UNBOXEDBLOCK.GCT` (`NIL` is an old style synonym), `PTRBLOCK.GCT` (`T` is an old style synonym) or `CODEBLOCK.GCT`. *INITONPAGE* is the number of cells at the beginning of the block which must be allocated on the same page. *ALIGN* is the alignment of the address of this block in memory space. The base address will be evenly divisible by this number.

The argument *BASE* in the following functions refers to an address, an Interlisp pointer. For example, if the value of *X* is an instance of a datatype, then *X* is actually a pointer to the first cell of that instance. There are essentially only two operations that perform "pointer arithmetic": `\ADDBASE` and `\VAG2`; these compile directly into the `ADDBASE` and `VAG2` opcodes.

`(\ADDBASE BASE OFFSET)`

Produces a new address that is *OFFSET* 16-bit words beyond *BASE*.

`(\VAG2 HI LO)`

Produces an address whose left 8 bits is *HI* and whose right 16 bits is *LO*.

There are, however, many other ways to produce addresses that ultimately perform `\VAG2` or `\ADDBASE`, and these are usually preferable. The record `POINTER` is useful for decomposing pointers into `page#` and `word-in-page` or `cell-in-page` quantities. `LOCF` is useful in conjunction with `BLOCKRECORDs` and `DATATYPEs`.

`(LOCF (fetch FIELDNAME of datastructure))`

[Macro]

Produces a pointer to the first word containing *FIELDNAME*. E.g., if *BAR* is declared as a `WORD` field in a record, then `(fetch BAR of X)` is equivalent to `(\GETBASE (LOCF (fetch BAR of X)) 0)`.

(INDEXF (fetch FIELDNAME of T))

[Macro]

Returns the word offset to the first word containing FIELDNAME. Since this is independent of the actual datum being operated on, the datum is often given as "T". E.g., if BAR is declared as a WORD field in a record, then (fetch BAR of X) is equivalent to (\GETBASE X (INDEXF (fetch BAR of T))). There is rarely any need for INDEXF.

Note that \ADDBASE, LOCF and other pointer-producing operations are *not* in general safe in Interlisp-D. The garbage collector can get very confused if you save away arbitrary pointers anywhere other than in local variables. This is because the reference count of an object is associated only with the pointer to its beginning, i.e., only with the address that the public traffics in (the pointer returned from **create**, for example). If you must store away internal pointers, be very careful that you continue to hold on to the pointer to the start of the object for as long as you maintain the internal pointer. This assures that the object will not get garbage-collected out from under you, the most common source of such confusion.

(\ADDBASE2 BASE N)

Equivalent to (\ADDBASE BASE 2*N).

(\GETBASE BASE OFFSET)

(\PUTBASE BASE OFFSET VALUE)

These fetch and store, respectively, the 16-bit word (as a Lisp small positive integer) located at OFFSET words beyond BASE. \PUTBASE is *really* dangerous. E.g., (\PUTBASE NIL n) for many small values of n will smash your system beyond repair. Not good for a residential environment where a smashed system can lose a lot of work.

(\GETBASEBYTE BASE OFFSET)

(\PUTBASEBYTE BASE OFFSET BYTE)

Fetch and store 8-bit quanta. BASE is a word address, and OFFSET is a byte offset—counting the high byte of the base word as offset zero.

(\GETBASEPTR BASE OFFSET)

Fetches a pointer at OFFSET from BASE. A pointer is a 24-bit quantity, which is stored right-justified in a 32-bit cell. Note, however, that BASE and OFFSET are both still in terms of 16-bit words.

(\PUTBASEPTR BASE OFFSET PTR)

Stores pointer PTR at OFFSET from BASE. This is not a direct inverse of \GETBASEPTR, because it stores a full 32 bits, never mind what used to be the high 8 bits originally stored there. \PUTBASEPTR does not do reference counting, so this can be especially dangerous if not used carefully. BASE is a word address, and OFFSET is in *words*, not cells! \RPLPTR is similar to \PUTBASEPTR, except that it *does* do reference counting.

(\RPLPTR BASE OFFSET PTR)

Stores a 24-bit pointer, similar to \PUTBASEPTR, except that (a) it stores only 24 bits, preserving whatever used to be in the high 8 bits; and (b) it does reference-counting operations; decrements the count of the pointer being smashed and increments the count of

the pointer being put into the location. This is the proper way to smash a pointer field if you must. However, there is almost never any need for you to call this directly; the usual way to smash a pointer field is to use records.

Implementation notes: `\GETBASEBYTE` and `\PUTBASEBYTE` compile directly into the corresponding opcodes, and execute entirely in microcode when the *OFFSET* and *VALUE* arguments are small positive integers. `\GETBASE`, `\PUTBASE`, `\GETBASEPTR`, `\PUTBASEPTR`, and `\RPLPTR` compile directly into the corresponding opcodes when the *OFFSET* argument is a constant less than 256; for other *OFFSET* arguments (variable quantities, larger integers), they require an *ADDBASE* in addition.

```
(\GETBASEFIXP BASE OFFSET)
```

```
(\PUTBASEFIXP BASE OFFSET VALUE)
```

These fetch and store 32-bit integers.

```
(\GETBASEFLOATP BASE OFFSET)
```

```
(\PUTBASEFLOATP BASE OFFSET VALUE)
```

These fetch and store 32-bit floatps.

```
(\GETBASESTRING BASE OFFSET NCHARS)
```

Creates a string *NCHARS* characters long whose characters consist of the bytes located starting at *OFFSET* (a byte offset) from *BASE*. Thus, the first character of the result is `(\GETBASEBYTE BASE OFFSET)`.

```
(\PUTBASESTRING BASE OFFSET STRING)
```

Stores the characters of *STRING* as consecutive bytes starting at *OFFSET* (a byte offset) from *BASE*.

```
(\BLT DBASE SBASE NWORDS)
```

Copies a sequence of *NWORDS* words starting at *SBASE* to corresponding words starting at *DBASE*. This compiles directly into the BLT opcode. In the case where the source and destination ranges overlap, the behavior is well-defined: words are copied from the end of the range backwards to the beginning. Thus, this is equivalent to (for *I* from *NWORDS*-1 to 0 by -1 do `(\PUTBASE DBASE (\GETBASE SBASE I))`). Very fine point: this operation is defined to be completely uninterruptable if *NWORDS* is less than 10; thus, you can use opcode to make small indivisible transfers.

```
(\MOVEWORDS SBASE SOFFSET DBASE DOFFSET NWORDS)
```

Obsolete predecessor of `\BLT`.

```
(\MOVEBYTES SBASE SBYTEOFFSET DBASE DBYTEOFFSET NBYTES)
```

Copies a sequence of *NBYTES* bytes starting at *SBYTEOFFSET* bytes beyond *SBASE* to *DBYTEOFFSET* bytes beyond *DBASE*. If the ranges overlap, the result is formally undefined.

```
(\ZEROBYTES BASE FIRST LAST)
```

Stores zeroes into the bytes at offsets *FIRST* thru *LAST*, inclusive, from *BASE*. Thus, a total of *LAST-FIRST+1* bytes are cleared.

```
(\ZEROWORDS BASE ENDBASE)
```

Stores zeroes into the words from *BASE* thru *ENDBASE*, inclusive. There are obscure reasons for the lack of symmetry among `\ZEROWORDS`, `\ZEROBYTES`, and `\MOVEBYTES`.

ABC, EXPORTS.ALL, &c:

Note that for Interlisp-D, `LOAD(ABC)` loads `<LispCore>Library>EXPORTS.ALL`.

`(LOAD 'MAKE-EXPORTS.ALL)` will connect to `<LISPCORE>SOURCES>` and gather all exports into the `EXPORTS.ALL` file.

“ABC” stands for “A Byte Compiler”—meaning the augmented environment required to compile Interlisp system code. The augmentation includes any of the definitions found under the `EXPORT FILEPKG` command. The variable `EXPORTFILES`, set up by loading the file `FILESETS`, contains the rootname of all system files which have any `EXPORT` commands. A file will generally export those items that other files need (e.g., records or macros) which are `DONTCOPY`, and thus not part of the user’s system.

SYSRECORDS [van Melle]:

In order for the inspector to be able to inspect an object of some user-declared datatype, it needs a declaration for it. The declarations for system datatypes are omitted from the loadup (by being marked `DONTCOPY` and being initialized with `INITRECORDS`). In order for their instances to be inspectable, they should be added to `SYSTEMRECLST` by the filepkg command `SYSRECORDS`, which is syntactically identical to `RECORDS`. The datatype declaration is actually stripped of comments, subrecords and initialization info before being put out.

Putting new DLion Microcode into a Sysout:

Load the file `SPLICE.DCOM` from `<Lispcore>Sources>`.

`(NCLIP MICROCODEFILE SYSOUTFILE)`

Copies the entire contents of *MICROCODEFILE*, a DLion .db file, into *SYSOUTFILE*, which must be a Lisp sysout. Both files must be random access.

DTigerness vs. DLionness

How programs can tell if they're running on a DTiger

Programs can tell if they are running on a Dandelion (1108 with CPE) by evaluating `(AND (EQ \MACHINETYPE \DANDELION) (ODDP (\DEVICE.INPUT 8)))` - this will return T when run on a DTiger, NIL when run on anything else. `(\DEVICE.INPUT 8)` returns the microcode version number when run on an 1108; if the version is odd, you're running on a DTiger.

WHEREIS, and the WHEREIS database

The WHEREIS package is used to find where (which source package) an atom has come from.

The expanded WHEREIS package comes preloaded in FULL.SYSOUT, or can be loaded from {ERIS}<LISPCORE>LIBRARY>WHEREIS.DCOM. Normally WHEREIS only searches loaded files. If the function WHEREIS is called with a FILES argument of T it searches the list of hashfiles given in the global WHEREIS.HASH. These hashfiles may have names such as WHEREIS.HASH, LIBRARY.WHEREISHASH, SYSTEM.WHEREISHASH, etc., and may live in places like {ERIS}<LISP>INTERMEZZO>LIBRARY. Usage:

```
_ (WHEREIS <foo> 'FNS T)
```

This definition also allows you to ask for MACROS, RECORDS, PROPS, and VARS in addition to FNS.

To make a new WHEREIS database use the function WHEREISNOTICE thus:

```
_ WHEREISNOTICE (
  (<LISPCORE>SOURCES <LISPCORE>LIBRARY> <LISPUSERS>)
  T
  <LISPCORE>SOURCES>SYSTEM.WHEREISHASH]
```

Note that it takes a long time to examine all the files. Best to leave this task to a lonely Dorado.

Installing New Opcodes [Masinter]:

[abstracted from messages to Jim desRivieres]

The various functions (CALLSCCODE, PRINTCODE, CHANGENAME) know about opcodes via the list \OPCODES. If you want to add some new opcodes, you can edit the list. The format of \OPCODES is documented, I think, in the function PRINTOPCODES, which is on the file ACODE. Note that if you install new opcodes on the fly you should then reset the variable \OPCODEARRAY to NIL.

You can tell if you have installed the opcodes by calling \FINDOP directly. (\FINDOP 'CAR) should return the opcode-description-record for CAR, while (\FINDOP 231Q) should look up opcode 231.

If you add opcodes, you should send a message to LispCore^ outlining what opcodes you want to reserve. The file OPCODES.TEDIT (on <LispCore>INTERNAL>DOC) I think has a listing of opcodes too, and if you are reserving a range, that reservation should be documented there too.

Subject: adding UFNs

The UFN mechanism hasn't really been extended for simple experimentation but is workable with a little effort. Normally, UFN entries get set up at MAKEINIT time by a renamed version of a function, I think it is called \SETUFNENTRY or some such. (LLCODE, LLBASIC, LLNEW or one of those). The entries in the OPCODES record is used to set up the ufn's. Now, it is currently the case that UFN's can't do anything like push N things on the stack -- all they can do is pop N arguments (N>=0) and push 1 result.

Writing ufn's that do something other than that, e.g., that don't follow the normal function call paradigm, are a lot more work. Basically I think you have to get into the level of stack-hacking that is found inside LLSTK. For example, a UFN that wanted to push a bunch of NIL's would have to do something awful, like steal space out of its own basic frame to give it back to the caller. This kind of code is tricky to write and debug, especially because you can't do things like insert BREAKs.

Popping N off of course is easy since that is what function calls do.

In order to do a jump operation, doing something like (add (fetch PC of (\MYALINK)) 10) would do a relative jump to byte +10.

It may actually be necessary to extend the UFN mechanism to allow some of the extensions that you want. Why don't you figure out what you can do with the current mechanism, and come back with the ones that you can't figure out how to implement.

Adding new opcodes to the Interlisp-D system

Written by: Herb Jellinek
Revised: 14 June 1984

The process of adding new opcodes to the Interlisp-D system has long been a mysterious one. This document is an attempt to shed some light on these mysteries. The document covers: Creating new opcodes, Writing UFNs, and The OpcodeTool. Enjoy.

Creating new opcodes

There are a number of global objects and properties that one must know about in order to install new opcodes/UFNs. Here's a list of them:

`\OPCODES` [List]

A list of the current opcodes, each of which is a record of type `OPCODE`.

`\OPCODEARRAY` [Array]

An array-ified version of `\OPCODES`. If set to `NIL` it will be reinitialized from the contents of `\OPCODES`. `\OPCODEARRAY` is recreated when needed by the function `\FINDOP`.

`DOPVAL` [Property]

Information on how to emit code for a given function. There are two formats:

1. $(nargs . opcode-sequence)$

If the number of arguments supplied matches *nargs*, compile into the sequence *opcode-sequence*.

2. $((nargs1 . opcode-sequence1) (nargs2 . opcode-sequence2) \dots (nargsN . opcode-sequenceN) . other-cases)$

If the number of arguments supplied matches *nargs1*, compile into *opcode-sequence1*, otherwise see if the number of arguments supplied matches *nargs2*, etc. One may also supply a function name as the tail of the list; the code generator will call that function if none of the other cases apply. The function `OPT.COMPILERERROR` is typically used for this purpose; it is equivalent to `HELP`.

`DOPCODE` [Property]

The `OPCODE` record for a given atom.

`OPCODE` [Record]

A record describing the structure of the `DOPCODE` property and the elements of the list `\OPCODES`; it has the following fields:

UFNFN	name of the ufn. Actually read by MAKEINIT
LEVADJ	stack effect (+/-n) or token. used by PRINTCODE. See code in PRINTCODE for details
OPPRINT	used only by PRINTCODE.
OPNARGS	number of extra bytes
OPCODENAME	name of opcode
OP#	number of opcode, or range for opcode sequences

Herb will document OPPRINT.

Writing UFNs

UFNs are Lisp functions that either run in the place of unimplemented instructions or when the microcode detects a situation that is too complex for it to handle. (This is termed *punting*.) There are two cases involved in writing UFNs: those for single-byte opcodes, and those for multi-byte opcodes.

UFNs for single-byte opcodes

For example, assume we have a single-byte opcode called `SQRT`, which takes a `FLOATP` as operand and returns its square root. The instruction is designed to punt out to its UFN (named `\SQRT`) when its operand is of the wrong type, at which time the UFN can either attempt to coerce the operand to a `FLOATP` or signal an error. `\SQRT` need be no more than a function of a single argument.

UFNs for multi-byte opcodes

These UFNs are slightly more complicated, but not much. The difference between single-byte UFNs and multi-byte ones is in the handling of the "extra" (alpha, beta, gamma) byte or bytes. To wit: all multi-byte opcodes that begin with the same byte have the same UFN, and the extra bytes get passed to this UFN in the form of extra arguments. We might have a group of three bit-vector operators (we'll call them `BITOP`), that all begin with a bytecode of `72Q` and vary from 0 to 2 in the second byte. The bytecodes each expect one argument on the stack. The UFN (`\BITOP`), would probably have the following form:

```
(DEFINEQ
  (\BITOP
    (LAMBDA (BITVECTOR OP)
      (SELECTQ OP
        (0 (\BITOP.MASK BITVECTOR))
        (1 (\BITOP.SHIFT BITVECTOR))
        (2 (\BITOP.ROTATE BITVECTOR))
        (HELP "\BITOP - illegal operation" OP))))))
```

The OpcodeTool

[Imm: I changed the opcode format; I don't know if this works]

[hdj: it doesn't]

[jop: fixed 5-2-86]

The OpcodeTool is a package that makes it easy to set up and test new opcodes in a running Lisp system. Do

```
(LOAD ' {Eris} <LispCore>Misc>OpcodeTool.dcom)
```

This package has one entry point, MAKEOPCODE, a function which takes 8 arguments:

OPNAME	a litatom
NUM	the opcode number
OPNARGS	the number of extra bytes (alpha, beta, etc.)
OPPRINT	usually T
LEVADJ	the stack level adjustment for this opcode
UFNFN	the UFN for this opcode
UNIMPL	a list describing which machines have no microcode for this op
DOPVAL	a (optional) DOPVAL prop for the litatom OPNAME

After you've run MAKEOPCODE, you can compile functions that use the new opcode and test them out.

Creating New Devices:

Date: 15 MAR 84 09:47 PST
From: MASINTER.PA
Subject: AR for Implementors Manual
To: LispSupport
cc: Kaplan, LispCore^

Section on making new file devices, How To.

VMEM.PURE.STATE [van Melle]

When preparing a demo, it is often nice to set things up in such a way that you can push the boot button at any time to instantly restart the sysout, rather than having to go back to some installation utility to reinstall the sysout. `VMEM.PURE.STATE` is a hack that lets you do this. Basically, while it is on, the page fault handler is altered to write dirty pages beyond the original end of the vmem, thus keeping the original vmem "pure".

`(VMEM.PURE.STATE FLG)`

[Function]

When *FLG* is true, enables "pure vmem" as of the next operation that writes out a consistent vmem, e.g., `LOGOUT`, `SAVEVM`, or `SYSOUT`. While in this state, as long as you do not perform another vmem-writing operation (`LOGOUT`, etc.), you can boot the machine (or slightly more cleanly, call `(LOGOUT T)`) and be back in the same state as the `LOGOUT` (or whatever) that initiated the pure image.

When *FLG* is `NIL`, returns to normal page fault operation. This is usually not too interesting, unless you really do want to `LOGOUT`, etc and forgo the "checkpoint" you set up. Note, however, that in either case, your virtual memory file is bloated by whatever pages had been written to the end of the vmem file instead of where they belonged.

There is a mode in which `LOGOUT` compresses the pages back to where they belong, but I never got it fully debugged.

There is a new MP error in this state: 9316. It means you wrote out so many dirty pages, you ran into the absolute end of the vmem file (8MB) even though you still have plenty of "virtual memory" left.

There are two typical modes of operation:

(1) Call `(VMEM.PURE.STATE T)` before calling `SYSOUT`, thus making a sysout that has the pure feature turned on for anyone running it.

(2) Start up a sysout not so made, and then call

```
(PROGN (CLEARW (TTYDISPLAYSTREAM))
      (VMEM.PURE.STATE T)
      (LOGOUT))
```

to turn the vmem into which this sysout was loaded one with the pure property.

Dorado Mufflers & Manifolds

Date: 23 Sep 84 19:29 PDT
 From: JonL.pa
 Subject: [JONL.PA: Mufflers and Manifolds]
 To: Jellinek

A couple weeks ago I promised to send you "all my knowledge" about the alpha bytes of the MISC opcodes. Not much knowledge; don't even know for sure why I picked "9", but possibly Bill will know of an "8" used for Dolphin-specific purposes.

-- JonL --

----- Begin Forwarded Messages -----

Date: 6 MAR 84 18:42 PST
 From: JONL.PA
 Subject: Mufflers and Manifolds
 To: vanMelle, Charnley
 cc: Purcell, JonL

Sorry I didn't mention this before -- after talking with Bill and Don I decided to use opcode MISC1 with alphabyte of 9 to do the equivalent of Mesa's ReadWriteMufflerManifold operation. No MISC opcodes at all were implemented on the Dorado (before now), and the set of alphabytes from 0 through 8 seem to be exhausted by the Dlion/Dolphin needs.

The interface to this operation is:

Input -- one 16-bit integer; low-order 12 bits are Muffler/Manifold address, bit 2[↑]15 non-zero means to write manifold, zero means to read muffler.

Output -- for write operation: NIL. for read operation: one 16-bit integer whose low-order 15 bits are garbage and bit 2[↑]15 is the 1-bit value of the muffler.

See the file <LispCore>Misc>MAKEDORADONSHOSTNUMBER for the code I'm about to install modulo the following: (1) SELECTQ on \MACHINETYPE, (2) don't do it if (MICROCODEVERSION) is less than 12004Q, (3) do "replaces" into (IFPAGE NSHost[i]) rather than cons up the NSHOSTNUMBER record.

----- End Forwarded Messages -----

The DLion Low Level Disk Drivers

Last revised: 26-Jun-84 23:13:59 by Mitch Lichtenberg

This file is an attempt to explain the operation of the Dandelion rigid disk interface, the microcode, and how Lisp constructs and uses disk IOCBs to perform disk operations.

DISK DRIVE INTERFACE

The Dandelion's central processor divides its time among the high speed I/O devices: the ethernet, the rigid disk, the I/O processor, and the display. The "I/O Page" is located in a well-known (to the microcode and Lisp) area of virtual memory, and it holds locations for communication between the different "micro-tasks."

Currently, the Dlion's Disk IOCB is the second word on the I/O page (that is, (`\ADDBASE \IOPAGE 1`)). Memory locations placed on this page must be up to 16 bits long, which constrains the address to be within the first 256 pages.

The IOCB page is used to store parameters and other information that is picked up by the microcode. When one wants to initiate an I/O operation, it can be done by depositing the parameter block onto the IOCB page (somewhere) and then placing the location (16 bits) of the parameter block onto the device's "mailbox" on the I/O page. When the device notices that something has been deposited onto its special I/O page location, it will read in the parameters, execute the operation, and reset the flag to zero to indicate that the operation is complete. (*Note: This is not completely true for the disk.). So, device I/O in Lisp usually looks like the following:

```
(\BLT (\ADDBASE \IOCBPAGE IOCBDisplacement) IOCB IOCBLen)
... or an alteration of an existing IOCB on the IOCB page ...
(\PUTBASE \IOPAGE DeviceCSBDisplacement
  (\LOLOC (\ADDBASE \IOCBPAGE DeviceIOCBDisplacement)))
(until (ZEROP (\GETBASE \IOPAGE DeviceCSBDisplacement)))
```

The `\PUTBASEs` and `\GETBASEs` usually come in the form of record package macros.

The reason why the disk does not follow the usual `\IOPAGE` convention of resetting its CSB to zero is because the drivers were meant to cause interrupts when disk I/O is finished. Lisp does not currently utilize this feature, so to poll the IOCB to detect when it has been completed, the IOCB status is set with some unused bits activated, and when the IOCB completes, the disk microcode will fill in the status and wipe out those bits as a side effect.

DISK IOCBS

To make life easier on the Dandelion's disk microcode, the implementors thought it to be a good idea to have the microcode emulate some kind of primitive "instruction set." So, when you want disk I/O, you have to write a little "program" in "IOCB Machine Language" to accomplish it.

Fortunately, these IOCBs are still around after booting, and Lisp leaves them in place but changes the fields to do more complicated operations.

The entire IOCB page is divided into three sections:

1. The Data Field:

This portion of the IOCB page contains mostly scratch space for the microcode and information for the programmer. Of particular interest here are the Header and Label "template" fields. For read operations, these fields are modified by the microcode, whereas on verify and write operations, they are just read.

The header field corresponds to the header records that were written on the disk when it was formatted. They contain identification information for the microcode - the sector's track, cylinder, and head numbers. These records are *never* written to. The usual operation on this field is verify, and it is primarily used to indicate to the microcode which sector is which on a given track, and to provide a security mechanism for the microcode. This template is also used to store the current cylinder number for the disk drive, and it is the place where the cylinder, track, and sector numbers are stored prior to a disk operation.

The label field is more general-purpose. In the pilot world, the ID number of a file and the page's relative location within the file are stored in the label field. For booting, a coded pointer is also stored here to lead the boot microcode from one sector to the next.

Note that the header and label fields *must* be in these locations on the IOCB page. (they may not be somewhere else in VM). The pointers to these fields from within the parameter blocks are only 16 bits long, so the headers and labels must be kept here.

2. The Parameter Area

The second portion of the IOCB page is the parameter areas for the IOCB programs. There are two of these IOCB parameter areas left over after booting, but Lisp only uses one of them. The information contained in the parameter block includes the run length, the type of operation to use (which operation, read/write/verify, to use on each field), the virtual page number of the disk buffer, and information on how to handle errors. They must be aligned on 16 word boundaries, due to the way that they are loaded into the micro-registers inside the CP.

3. The IOCBs

The third portion of the IOCB page is contains the actual IOCBs themselves. There are two basic types of IOCBs:

1. Seek IOCB:

The Seek IOCB is complete as it stands. (it has no parameter areas to read in like the transfer IOCB has). The Lisp code fills in the fields for the number of cylinders and the direction to seek, and the IOCB's code steps the drive head in the given direction for the given number of steps. There are no verify operations on seeks, so it is the programmer's responsibility to remember which track number the head is currently positioned at. If a disk drive gets lost, a recalibrate operation is necessary. This can be accomplished by setting up an IOCB to step

out one track, and continually running it until the `Track00` bit of the disk controller status register becomes T. This register can be read with the function (`\DEVICE.INPUT 3`) and the fields can be found in the record `DLDISK.STATUS` on `DISKDLION`.

2. Transfer IOCB:

The Transfer IOCB reads in a parameter area of 17 words, executes the transfer operation, and exits.

DISK IOCB MACHINE LANGUAGE

As was mentioned before, the disk microcode emulates a very small instruction set (to keep the code size down and increase flexibility). This instruction set is as follows:

Opcode	Operation
8000 xxxx	Send word xxxx to disk controller register <code>KCt1</code> .
0007 ssss	Set status bits from ssss
0000 aaaa	Increment number in location aaaa, and skip if zero.
0002 aaaa	Unconditional jump to location aaaa.
0006	Finish up IOCB.
0400 aaaa	Write status to location aaaa.
0005 aaaa	Load parameter table from locations starting at aaaa.
0800	Transfer a run of pages, skip of no error

Some interesting "8000 xxxx" commands follow:

Opcode	Operation
8000 0422	Wait for pending seeks to complete. (InsureSeekComplete)
8000 0420	Seek step IN (positive direction)
8000 04A0	
8000 0460	Seek step OUT (negative direction)
8000 04E0	

Inside the parameter areas, the following "code numbers" are important:

Code	Meaning
001E	Abort on NotReady, WriteFault, Overrun, or CRC errors
001C	Abort on NotReady. WriteFault, Overrun
001F	Abort on NotReady, WriteFault, Overrun, CRC, or Verify

OTHER NOTES, RESTRICTIONS, ETC.

To specify the length of the data field, "8100" is used instead of "0100". Setting the high bit of the data length field causes the microcode to increment the virtual page number after each page is transferred. This is used primarily for multiple page runs.

The length of the header and label fields must be decremented for verify operations.

It is impossible to follow a write operation with anything other than a write operation. That is, if you write the label field you *must* write the data field. Otherwise, the tail of the label write operation will trash the data field. (Something in the microcode or disk controller causes this, and it cannot be avoided!)

The files [Eris]<Lispcore>Dlion>DiskBootIOCBs.bravo and [Eris]<Lispcore>Dlion>DiskTest.dm contain many sample IOCBs. They are invaluable anyone tinkering with the Dlion disk system.

THE LISP DLION DISK HEAD

The heads for the DLion disk are stored on the file DISKDLION in the sources directory. The following is a description of the functions in this file and their purposes:

(\DL.DISKINIT) [Function]

Determines the shape of the disk drive and sets up variables as follows:

```
\DISKTYPE:    One of \SA4000, \SA1000, \Q2040, \Q2080
SEC/HD:       Sectors per head on this disk drive
SEC/CYL:      Sectors per cylinder on this disk drive
```

The data for each drive follows:

Drive	Sec/Hd	Sec/Cyl	Heads
SA4000	28	224	8
SA1000	16	64	4
Q2040	16	128	8
Q2080	16	112	7

(\DL.RECALIBRATE) [Function]

Attempts to find track zero of the disk drive by repeatedly stepping the drive out and checking the status word for Track00 indication. If more than 512 steps are made and Track00 still is not true, a call to RAID is made.

(\DL.DISKSEEK CYL) [Function]

Seeks disk drive to cylinder CYL, and updates information in the header template.

(\DL.TRANSFERPAGE DA BUFFER MODE LABEL
RUNLENGTH NORAIIDFLG) [Function]

"User" entry (that is, DLion file system entry) to the disk head. DA is the disk address, which may be a fixp. The remaining args are the same as those for \DL.XFERDISK, as described below:

(\DL.XFERDISK CYL HD SEC BUFFER MODE LABEL [Function]
 RUNLENGTH NORAIIDFLG)

Starts a Disk I/O operation. The argument format is meant to be compatible with the old \DL.XFERDISK. This minimizes the confusion with changing the swapper. New features include the ability to work with labels and an error recovery mechanism. If a disk error occurs, the \DL.XFERDISK function will retry the operation up to ten times. If it fails, it will do a (\DL.RECALIBRATE) first and try ten more times before finally calling RAID. Arguments are as follows:

CYL Cylinder number of disk address

HD Head number of disk address

SEC Sector number of disk address

Note: These numbers will be normalized automatically. For example, it is permissible to transfer Cylinder 0, Head 440, Sector 1215. \DL.XFERDISK will change that into a meaningful value. This is how the swapper works - see below.

BUFFER A pointer to the first page that will be used in the disk operation. **Note:** The page must be *locked down*, *touched* (referenced), and *dirty*, or else the swapper will not perform properly!

MODE One of the following:

NIL	Read pages , read labels (VRR operation)
T	Write pages, read labels (VRW operation)
VRR	Read pages, read labels
VVR	Read pages, verify labels
VVW	Write pages, verify labels
VWW	Write pages, write labels
VRW	Write pages, read labels (used by swapper)

LABEL A pointer to the label record (10 words), or NIL if you don't want to use a label record. The label must be locked down to prevent page faults inside the \DL.XFERDISK routine.

RUNLENGTH The number of consecutive pages to transfer, or NIL for one. There are restrictions on multiple page runs: To do a multiple page run, the virtual page numbers of the buffer pages must be sequential, and the run may not cross cylinder boundaries. (See function \CYLBOUNDCROSSP below).

NORAIIDFLG Normally, \DL.XFERDISK will bomb after failing to do an I/O operation ("failing" does not include verify errors). (It will call RAID). To suppress this, set NORAIIDFLG to T and disk errors will be returned as status to the caller.

(\CYLBOUNDCROSSP DA1 DA2) [Function]

Predicate returns T if DA1 and DA2 are on different cylinders, NIL otherwise. Note: This function is not locked down.

(\DL.DISKOP IOCB) [Function]

Passes IOCB to the disk microcode (which starts the I/O operation), waits for it to complete, and returns the status.

(\D2V CYL HDSEC) [Function]

Returns the disk address of the page on cylinder CYL and with encoded head and sector information in HDSEC (left byte is head number, right byte is sector number)

(\V2HDSEC DA) [Function]

Returns encoded head and sector information from disk address.

(\V2CYL DA) [Function]

Returns cylinder number from disk address.

(\DL.ACTONVMEMPAGE FILEPAGE BUFFER WRITEFLG) [Function]

Performs a file operation on the virtual memory file. FILEPAGE is a file relative page number to transfer, BUFFER is the page number for the transfer, and WRITEFLG is passed to \DL.XFERDISK as the MODE parameter. It is usually T or NIL. This function is implemented by figuring the starting address triple of the beginning of the VMEM file and computing the number of pages into the disk from there that the page is located (skipping bad pages), then supplying this information as the sector number to \DL.XFERDISK, which normalizes it internally to a real disk address.

(\DL.ACTONVMEMFILE FILEPG BUFFER NPGS WRITEFLG) [Function]

Performs multiple file operations on the virtual memory file. FILEPG is the starting file page number (relative to the start of the VMem file). BUFFER is a pointer to the first page in the group to be transferred. NPGS is the number of pages to transfer. WRITEFLG passed to \DL.ACTONVMEMPAGE as the WRITEFLG parameter. This function will transfer a run of pages to or from the virtual memory file.

(\DLDISK.GETSTATUS) [Macro]

Returns the status of the disk controller in a smallp. Use the record definition DLDISK.STATUS to understand its contents. This macro expands to (\DEVICE.INPUT 3)

DLDISK.STATUS

[Record Definition]

Record definition (access functions) for reading the result of (\DLDISK.GETSTATUS). Contains the following fields:

TRACK00	True if on track zero
HEADSELECT	Current head number
SA1000	True if controller is in SA1000 mode
DRIVENOTREADY	True if drive is not ready
WRITEFAULT	True if last operation caused a write fault
OVERRUN	True if last operation caused an overrun
CRCERR	True if last operation caused a CRC error
VERIFYERR	True if last operation caused a verify error

IOCBPAGE

[Record Definition]

This record contains the layout of the IOCB page. The fields are as follows:

LASTIOCBSTATUS	Last status reported while running IOCB
NEXTIOCB	Short pointer to next IOCB in chained IOCBs. This is not currently used.
SEEKIOCBLOC	Contains the location of the SEEK IOCB.
XFERIOCBLOC	Contains the location of the TRANSFER IOCB
VRRIOCBLOC	Contains the location of the VRR Parameter block
VVRIOCBLOC	Contains the location of the VVR Parameter block
HCYLINDER	Header Template: Contains current cylinder number. Changed in all operations.
HHEAD	Header Template: Contains current head number. Changed in all operations.
HSECTOR	Header Template: Contains current sector number. Changed in all operations.
LID	Label Template: 5 words of ID number for the label.
LPAGELO	Label Template: Low 16 bits of page-within-file information in the label.
LPAGEHI	Label Template: High 7 bits of page number within file.
LFLAGS	Label Template: Flag storage for boot code
LTYPE	Label Template: Type of page (type of file in which the page is a part) (16 bits)
LBOOTLINKCHAIN1	Label Template: Boot chain info
LBOOTLINKCHAIN2	Label Template: Boot chain info
PRUNLENGTH	Parameter Block: Run length (number of pages)
PLABELCMD	Parameter Block: Code for operation on label field
PLABELLEN	Parameter Block: Length of label field
PLABELABORT	Parameter Block: Conditions for aborting transfer & error codes to scan for
PDATA CMD	Parameter Block: Code for operation on data field
PDATALEN	Parameter Block: Length of data field
PVPAGE	Parameter Block: Virtual page number of memory buffer

PDATAABORT	Parameter Block: Conditions for aborting transfer & error codes to scan for
PTERMCOND1	Code to halt hardware after transfer
PTERMCOND2	Code to halt hardware after transfer
SCYLINDERDISPLACEMENT	Seek IOCB: Number of cylinders to move in seek operation.
SSEEKCMD1	Seek IOCB: First part of seek command
SSEEKCMD2	Seek IOCB: Second part of seek command

DISK IOCB PAGE

This section contains the contents of the IOCB page.

Displacements are relative to the start of the IOCB page.

Lines with asterisks following the opcode indicate fields for the user to fill in.

Address Op/Data Comment

0100:	000B		; Special Block Type
0101:	00FE		; Word count
0102:	0000		; Not used
0103:	0000		; IOCB Status (filled in by uCode)
0104:	0000	*	; Next IOCB address (or 0 for last one)
0105:	0120		; Address of seek IOCB
0106:	0132		; Address of transfer IOCB
0107:	0140		; Address of verify-read-read parameter area
0108:	0160		; Address of verify-verify-read parameter area
0109:	0180		; Address of verify-verify-write parameter area
010A:	01A0		; Address of verify-write-write parameter area
010B:	0000	*	; Header template: Cylinder number
010C:	0000	*	; Header Template: Head[0..7], Sector[0..7]
010D:	0000	*	; Label Template: Word 0 \
010E:	0000	*	; Label Template: Word 1 \
010F:	0000	*	; Label Template: Word 2 > ID Number for page
0110:	0000	*	; Label Template: Word 3 /
0111:	0000	*	; Label Template: Word 4 /
0112:	0000	*	; Label: Page # low [bits 7..22]
0113:	0000	*	; Label: [Pg# Hi 0..6, Pad 7..12, Flags 13..15]
0114:	0000	*	; Label: Type #
0115:	0000	*	; Label: Unused
0116:	0000	*	; Label: Unused
0117:	0000		; Filler for 16 wrd boundary lineup
0118:	0000		; Filler for 16 wrd boundary lineup
0119:	0000		; Filler for 16 wrd boundary lineup
011A:	0000		; Filler for 16 wrd boundary lineup
011B:	0000		; Filler for 16 wrd boundary lineup
011C:	0000		; Filler for 16 wrd boundary lineup
011D:	0000		; Filler for 16 wrd boundary lineup
011E:	0000		; Filler for 16 wrd boundary lineup
011F:	0000		; Filler for 16 wrd boundary lineup

Parameter Area for Verify-Read-Read IOCB

```

0140:      0000      *      ; Number of sectors to read
0141:      0031      ; Max #+1 of secs that may be skipped searching
0142:      0432      ; Verify header field
0143:      0001      ; word count-1 of header field
0144:      010B      ; address of header field
0145:      001C      ; Abort on NotReady. WriteFault, Overrun
0146:      0003      ; skip to next sector if CRC/Vrify err on header
0147:      0430      ; Read label field
0148:      000C      ; word count of label field
0149:      010D      ; Address of label field in IOCB
014A:      001E      ; Abort on NotReady, WriteFault, Overrun, CRC
014B:      0430      ; Read Data Field
014C:      8100      ; Length of data field (256 words)
014D:      0000      *      ; virtual page # of buffer
014E:      001E      ; Abort on NotReady, WriteFault, Overrun, CRC
014F:      0420      ; control word to halt hw after each field
0150:      0426      ; control word to find sector mark for header

```

Parameter Area for Verify-Verify-Read IOCB

```

0140:      0000      *      ; Number of sectors to read
0141:      0031      ; Max #+1 of secs that may be skipped searching
0142:      0432      ; Verify header field
0143:      0001      ; word count-1 of header field
0144:      010B      ; address of header field
0145:      001C      ; Abort on Not Ready. Write Fault, Overrun
0146:      0003      ; skip to next sector if CRC/Vrify err on header
0147:      0432      ; Verify label field
0148:      000B      ; word count of label field (-1 for verify)
0149:      010D      ; Address of label field in IOCB
014A:      001F      ; Quit on NotRdy, WrtFlt, Ovrn, CRC, Verif Err
014B:      0430      ; Read Data Field
014C:      8100      ; Length of data field (256 words)
014D:      0000      *      ; virtual page # of buffer
014E:      001E      ; Quit on NotReady, WriteFault, Overrun, or CRC
014F:      0420      ; control word to halt hw after each field
0150:      0426      ; control word to find sector mark for header

```

Parameter Area for Verify-Verify-Write IOCB

```

0140:      0000      *      ; Number of sectors to read
0141:      0031      ; Max #+1 of secs that may be skipped searching
0142:      0432      ; Verify header field
0143:      0001      ; word count-1 of header field
0144:      010B      ; address of header field
0145:      001C      ; Abort on Not Ready. Write Fault, Overrun
0146:      0003      ; go to next sector if CRC/Verify err on header
0147:      0432      ; Verify label field
0148:      000B      ; word count of label field (-1 for verify)
0149:      010D      ; Address of label field in IOCB
014A:      001F      ; Stop on NotRdy, WrtFlt, Ovrn, CRC, Verif Err
014B:      043B      ; Write Data Field
014C:      8100      ; Length of data field (256 words)
014D:      0000      *      ; virtual page # of buffer
014E:      001C      ; Abort on NotReady, WriteFault, Overrun
014F:      0420      ; control word to halt hw after each field
0150:      0426      ; control word to find sector mark for header

```

Parameter Area for Verify-Write-Write IOCB

```

0140:      0000      *      ; Number of sectors to read

```

```

0141:      0031      ; Max #+1 of secs that may be skipped searching
0142:      0432      ; Verify header field
0143:      0001      ; word count-1 of header field
0144:      010B      ; address of header field
0145:      001C      ; Abort on Not Ready. Write Fault, Overrun
0146:      0003      ; skip to next sector if CRC/Vrfy err on header
0147:      043B      ; Write label field
0148:      000C      ; word count of label field
0149:      010D      ; Address of label field in IOCB
014A:      001C      ; Abort on Not Ready. Write Fault, Overrun
014B:      043B      ; Write Data Field
014C:      8100      ; Length of data field (256 words)
014D:      0000      * ; virtual page # of buffer
014E:      001C      ; Abort on NotReady, WriteFault, Overrun
014F:      0420      ; control word to halt hw after each field
0150:      0426      ; control word to find sector mark for header

```

Parameter Area for Verify-Read-Write IOCB

```

0140:      0000      * ; Number of sectors to read
0141:      0031      ; Max #+1 of secs that may be skipped searching
0142:      0432      ; Verify header field
0143:      0001      ; word count-1 of header field
0144:      010B      ; address of header field
0145:      001C      ; Abort on Not Ready. Write Fault, Overrun
0146:      0003      ; skip to next sector if CRC/Vrfy err on header
0147:      0430      ; read label field
0148:      000C      ; word count of label field
0149:      010D      ; Address of label field in IOCB
014A:      001C      ; Abort on Not Ready. Write Fault, Overrun
014B:      043B      ; Write Data Field
014C:      8100      ; Length of data field (256 words)
014D:      0000      * ; virtual page # of buffer
014E:      001C      ; Abort on NotReady, WriteFault, Overrun
014F:      0420      ; control word to halt hw after each field
0150:      0426      ; control word to find header sector mark

```

Disk IOCB program for Seek

```

0152:      0000      * ; Number of cylinders to move (negative)
0151:      8000      ; Insure that the seek
0152:      0422      ; completed from before
0153:      0007      ; clear out the status bits that
0154:      0000      ; are not used in a seek operation
0155:      8000      ; send a step pulse
0156:      0000      * ; (direction is filled in)
0157:      8000      ; finish sending step pulse
0158:      0000      * ; (direction is filled in)
0159:      0000      ; increment remaining distance
015A:      0152      ; in IOCB field, and skip if zero
015B:      0002      ; Jump back to the step
015C:      0155      ; loop.
015D:      0006      ; Clean up and finish IOCB
015E:      0000
015F:      0000
0160:      0400      ; quit and write status back
0161:      0103      ; into status word

```

Disk IOCB program for Transfer

```

0162:      0005      ; Load parameters from
0163:      0000      * ; parameter table

```

```
0164:      8000      ; Wait for any pending seeks
0165:      0422      ; to complete.
0166:      0800      ; transfer run of pages
0167:      0002      ; if there was an error,
0168:      0169      ; finish up anyways (we're done!)
0169:      0006      ; Clean up and finish IOCB
016A:      0000
016B:      0000
016C:      0400      ; quit and write status back
016D:      0103      ; into status word.
```

End of file {Eris}<LispCore>Internal>Doc>DLionDiskDriver.TEdit.

Interlisp-D AR Fields

Interlisp-D software support and development uses an "Action Request" data base system for keeping track of bug reports and requests for features. We take feature requests as serious as bug reports -- don't hesitate to ask. Users are encouraged to submit ARs, either via mail, electronic mail, or (for internal Xerox users) directly using the AREDIT facility within Interlisp-D. The following documents the fields we use in ARs and what they mean:

AR identification

Number: Every AR has a number, which increases by one for each AR submitted. AR numbers are **never** recycled, and ARs are **never** deleted. The AR number is automatically generated.

Source: This is an arbitrary string which should contain the name and electronic address of original customer or internal Xerox users reporting problem. Customers should be identified by "Liason (Customer name)", e.g., Raim.pasa (Bill White @ Teknowledge). This field is used in sending mail back to find out more technical details, or to notify people about the change in status. If multiple people report the same bug, each one should be included in the Source field.

Problem Description

Subject: A terse summary of problem, enough to identify it uniquely. "FOO doesn't work" or "Floppy problem" is not good enough. Think of yourself as a newspaper headline writer: "Attempt to write file when Floppy door open causes awful noise". Implementors may change the Subject field as more details about the true nature of the problem becomes apparent. Feature requests generally start with "Want", e.g., "Want way to make windows triangular rather than square."

Problem Type: What kind of problem report or feature request:

Bug	The system does not work as documented.
Implementation	The system works, but the internal structure is wrong. (Generally submitted by other implementors or looking at the sources.)
Interface	The design of the user interface is wrong. Includes problems in the way in which things display, as well as program callable structures.
Feature Request	Request for a new feature or set of facilities
Documentation	The system works, but the documentation is wrong, unclear, or incomplete.
Performance	The system works, but it is too slow doing the described operation.

Description: This field should contain the complete description of problem or request, including any subsequent discussion. If bug reports come in via electronic mail, put the whole message in this field. Edit relevant info into the beginning of the Description, especially if it summarizes what the problem really is.

Frequency: How reproducible is the problem? Leave blank if irrelevant (e.g., feature request.) Generally only relevant for bug reports. One of:

Everytime	reproducible every time.
Intermittent	doesn't always happen.
Once	saw it happen once.

Impact: How seriously does it affect your ability to get work done, value of Interlisp-D, etc. The names apply to bug reports, but feature requests should be rated along analogous lines.

Fatal	causes system crash, loss of work, etc. requirement for project completion.
Serious	can be worked around but seriously interferes with work, requires substantial reimplementation
Moderate	tolerable, but clearly a problem, responsibility of Interlisp development

Annoying annoying problem, minor request for new feature that "would be nice"
 Minor may be some dispute about whether it is even a bug, very minor feature request.

Test Case: This field isn't used for what the name might imply: it should be a list of the files needed to recreate problem. Recipe for reproducing the problem (which is what you might think a Test Case was for) should be in the Description.

Lisp Version: This should be either the release name (Fugue.1, Carol, Harmony) and MAKESYSDATE in which the problem occurs (or the feature doesn't occur.) The Lisp AREEDIT package attempts to fill this in; if you submit from a different system that you are running in, please change it. If its a documentation problem, include the date of the documentation.

Machine: What machine does problem occur on (one of 1108, 1132, 1100). Leave blank if *all*.

Disk: What kind of disk is on the machine? (only fill in if relevant to AR.) Constrained to have the known disk types for Machine.

Microcode Version: (automatically generated, delete if known to be irrelevant, e.g., for documentation problems.)

Memory Size: (automatically generated. Delete if known to be irrelevant.)

File Server: If problem deals with communication, what server are you talking to? (This field should really have "Server" on it, rather than File Server.) One of VAX/VMS, VAX/UNIX, 8037, etc.

Server Software Version: As appropriate for the server you're talking to.

System: Subsystem: Category & sub-category of problem type. Subject to change. System includes: Lisp Software, System Software, Text and Graphics, Documentation, IO Architecture. Generally these are filled in by LispSupport, as it corresponds more to our own internal project structure.

Problem disposition

Workaround: If relevant, this field can contain a known procedure to work around problem until it is fixed; generally a short recipe belongs here.

Status: Status of AR as it moves thru the system:

New	All ARs start out as New.
Open	Has been looked at by LispSupport; all fields are filled in & has been assigned.
Fixed	problem fixed, in LispCore loadup. The In/By: field is set to the next release name. The Assigned-to: field is set.
Closed	System with fix in it has been tested & released.
Declined	Request for feature officially *never* going to be implemented (e.g., we think its a bad idea). Bug report considered spurious (we don't think it is a bug)
Superseded	Another AR includes the problem described in this one. In this case, the Subject of this AR should include the AR# of the one that supercedes it, and the superceding AR should contain the union of information in this one.
Obsolete	e.g., module in which problem reported is no longer supported.
Incomplete	The information submitted is not enough to take action -- not enough information to identify the bug, or the feature request doesn't spell out in enough detail what is wanted. This is different from Declined in that the request is considered valid, but open for more detail.

In/By: What version of Interlisp-D has/will have this problem fixed? (E.g., Harmony, Intermezzo, Jazz, Fugue.6, etc.)

Disposition: Brief notes explaining changes to status, plus automatically generated description of who changed status when.

Difficulty: Rough estimate filled in by developer on scope of problem.

Easy	< 1 week to fix
Moderate	< 1 month to fix
Hard	< 6 months to fix
Very Hard	> 6 months to fix
Impossible	can't be fixed
Design	requires a design

Priority: Development's estimate of whether it will be in the "next" release. (Changes from release to release).

Absolutely Release will be held if not completed.

Hopefully Major release goal. Schedule slip admitted, but likely to get completed.

Perhaps Will get implemented if other revisions in same area are completed.

Unlikely Unlikely to be included in the next release.

Assigned To: The developer who 'took care' of this AR, either by fixing it or declining it. Currently only after AR is fixed.

Attn: The developer(s) who should look at this AR. We're moving away from relying on this.

Submitter: The user name (and registry) of the person who actually did the Submit. Automatically generated by AR submit, not editable.

Date: The date the AR was originally submitted. Automatically generated by AR submit, not editable.

Edit-By: person who edit this AR last. Automatically generated by the Lisp AR edit tool. (The Disposition field contains more info about edits history.)

Edit-Date: date when this AR was last edited. Also automatically generated.

Maintenance Panel Error Code Summary for Xerox 1108 Interlisp-D

There are two types of maintenance panel codes: progress codes and error codes. Progress codes are placed in the Maintenance Panel at various stages of initialization. Error codes are traps which freeze or blink the error number in the maintenance panel. All errors except the 9000-range errors are fatal.

Summary of MP code ranges

0000-0499 boot-time diagnostics
0500-0699 IOP code
0700-0899 Pilot microcode
0900-0999 Pilot
1000-6999 tech-rep diagnostics
7000-8887 Star
8888-8888 MP lamp test
9000-9999 Lisp

Boot-time errors

0096 Insufficient real memory (<1MByte) for lisp
0149 Usually right after power-on. Disk not ready. Safe and effective to 0-boot from this state.

0200-0299 Booting phase 2 (Initial microcode)

0200 normal booting phase 2
0201 CP error in reading from boot device
0202 null Mesa germ installed in physical volume
0203 broken rigid disk boot chain (possibly intermittent)
0204 Illegal IOP port command
0205 CP Trap (CS parity or double-bit memory error)
0206 null diagnostic microcode in physical volume
0207 null Pilot/Mesa emulator microcode in physical volume
0208 null Mesa germ installed in physical volume
0217 Inconsistent Virtual Memory. Requires re-installation or try another partition.

0500-0502 Domino progress codes

0500	StartDomino	Domino has started
0501	InitReadTOD	Domino starting to read the TOD clock
0502	InitReadTODdone	Reading of TOD clock completed (next MP number from Lisp)

0505-0599 Domino error codes

0505	CSParity	CS parity error detected
0506	BurdockCPDDisabled	Burdock attempted to use EtherKludge
0507	CPBurdockDisabled	CP attempted to use EtherKludge
0508	IOPBreak	An IOP break with no IOP kernel
0509	IllegalIOPIntr	Illegal IOP interrupt
0510	BadMapEntry	Incorrect vm Map entry in IOP access.
0511	NoCPDmaComplete	CP Dma operation failed to complete
0512	NoCPDmaChannel	CP Dma channel not specified
0513	ReadCPPortDead	CP not responding to Read CPPort
0514	WriteCPPortDead	CP not responding to Write CPPort
0520	StackOverflow	A task's stack has overflowed
0565	InvToneCmd	Invalid keyboard tone generator comnd
0570	InvProcCmd	Invalid cmd value in Processor CSB
0571	UnImplCmd	Unimplemented cmd in Processor CSB
0572	SetTODError	The Time-Of-Day could not be set
0576	LSEPctlOVR	LSEP Control CSB overrun
0580	NoValidCommand	Invalid floppy IOCB command
0581	UnImplFloppyCmd	Unimplemented floppy IOCB cmd
0582	InvalidEscapeCmd	Invalid Escape floppy cmd
0583	CommandTrack	Floppy track register is not correct

0584	TrackToBig	Floppy track number is too large
0585	BadDmaChannel	Couldn't program Floppy Dma
0586	NoDmaEndCount1	External Dma End Count not set
0587	NoDmaEndCount2	Internal Dma End Count not set

0900-0999 Pilot codes

0915	Pilot breakpoint	
0937	Trying to find out the time and date. Will hang in this state if no time server is responding, and the time has not been set on the machine since power-up.	
0981	Trying to discover Ethernet pup host number. Will hang in this state if non-Lisp code tries to perform Pup operations and no Pup ID Server responds.	

9000-9299 DLion Interlisp-D microcode error detected

Most of these errors are indicative of some serious problem, probably hardware, and usually fatal (but try ↑D if you can't TeleRaid). The main exception is 9004—see description of code 9304.

9001	CSParErr	Control store parity error
9002	StackErr	hardware stack overflow
9003	IBEmptyErr	instruction fetch unit empty error
9004	VirtAddrErr	Attempt to reference virtual address >22 bits
9005	EmuMemErr	double bit memory error or non-existent memory
9009	CAR/CDR	bad pointer
9013	NegPcError	inconsistent PC at FnCall
9014	applyUfn	arg to apply not integer
9016	notFreeTrap	stack allocation error
9024	Page fault in the page fault handler.	
9051	BadUfnTable	
9120	MiscErr Output	opcode no such register
9121	MiscErr	opcode bad 2nd byte
9122	MiscErr Input	opcode no such register
9126	PcNegError	inconsistent PC at Punt

9300-9399 Lisp system code error (call to \MP.ERROR)

These codes generally indicate an error state in Lisp system code that cannot be handled in the break package. Most are "should never happen" cases that indicate a serious error; but some (in particular, 9305 and 9318) may be much less serious. If possible, use TeleRaid to find out more information (press the Undo key to enter the TeleRaid server (cursor changes into "TeleRaid"), and run the TeleRaid user from another machine). Even if you can't TeleRaid from another machine, several of these codes you can convert into a Lisp break if the world is still mostly consistent and the error occurred under user code (rather than, say, the garbage collector): type ↑B to the TeleRaid server. Summary of TeleRaid server commands:

↑B	attempt to enter Break. If error is in a special system context, will change cursor to "CANT", indicating refusal to enter break. Warning: even if the system is willing to try to enter a break, it may fail, leaving your system unrestartable. When in doubt, use ↑D.
↑D	perform Hard Reset—clear stack, flush all non-restartable processes.
↑N	continue from error. Warning: You should not use this command except for the following errors: 9318 (when you believe it be be continuable, see below); 9915 error when caused by typing the Raid interrupt; 9325; 9326; 9329.
↑P	display Pup host number (in decimal) in maintenance panel.

9302	Invalid Vmem: attempt to boot an image that is not a valid Lisp sysout, or which is inconsistent from having some, but not all, of its dirty pages written. Can happen if you boot instead of calling LOGOUT. Usually caught sooner as code 217.
9303	"No place for IOCB page at startup"—this usually only happens if your machine has insufficient memory.
9304	Obsolete [Map out of bounds].
9305	Invalid address: attempt to use a pointer that does not refer to an existing (allocated) part of virtual memory. Usually means garbage was fetched from somewhere that should have contained a pointer; a

- common source is code with type checking turned off attempting to fetch a datatype field from an object that is not a datatype, such as NIL or a small integer. This error can often be converted to a break with the ↑B TeleRaid command if the Lisp image is otherwise in a good state.
- 9306 Obsolete [Invalid virtual page].
 - 9307 "Unavailable page on real page chain"—inconsistent state in page fault handler.
 - 9308 "Loop in \SELECTREALPAGE"—inconsistent state in page fault handler.
 - 9309 Attempt to allocate already existing page (from call to \NEWPAGE).
 - 9310 A 9309 error recursively inside the new page allocator.
 - 9311 "Locked page occupies a file page needed to lock another"—bad state in virtual memory system.
 - 9312 Arg to CLOCK0 not an integer box.
 - 9313 Fault on resident page: processor took a page fault for a page that appears to be resident.
 - 9314 PageFault on stack: shouldn't happen, as stack is resident.
 - 9316 Obsolete [Attempt to extend vmem beyond 8MB].
 - 9317 "Attempt to write a locked page when not under \FLUSHVM"—bad state in virtual memory system.
 - 9318 Error in uninterruptable system code: an error that ordinarily would enter a break (e.g., a type test failure), but in a piece of code that should not be user-interruptable. This is generally a sign that some datum used by system code has been smashed, but this is not always fatal. Should you not be in a position to diagnose the error with TeleRaid, you can type ↑N after entering the TeleRaid server; Lisp will proceed from the MP halt and attempt to enter a break anyway, from which (if it succeeds) you might be able to glean more information about the problem. Warning: continuing with ↑N can be fatal if the error really was in a place where a break would not succeed.
 - 9319 Stack full: hard stack overflow. A soft stack overflow (Lisp break "STACK FULL") occurs when the stack is mostly used up; if you proceed beyond that point without resetting you can completely fill the stack and get this MP code. Press STOP to perform a HARDRESET to clear the stack, or run TeleRaid to find out who was guilty of overflowing the stack.
 - 9320 Storage is completely full. A continuable Lisp break "STORAGE FULL" occurs when the allocation space is nearly full.
 - 9321 Unknown UFN: attempt to execute an unimplemented opcode. This usually means that the processor is trying to execute random memory, or took a wild jump somewhere. Often a microcode bug.
 - 9322 Atoms full: the limit on number of litatoms (2^{16}) has been reached.
 - 9323 Obsolete [Pnames full].
 - 9324 Stack frame use count overflow: the program has attempted to create more than 200 references to the same stack frame.
 - 9325 Storage nearly full: this is a warning that comes later than the "STORAGE FULL" break but before you completely run out (and get a 9320). You can continue from this error with ↑N from TeleRaid.
 - 9326 Bad MDS free list: the free list of recycled MDS pages got trashed. You can continue from this error with ↑N from TeleRaid.
 - 9327 Bad array block. The array allocator found a bad array block in its free list. Generally means some unsafe code trashed one or more locations in array space.
 - 9328 A variation on 9327.
 - 9329 The garbage collector attempted to reclaim an array block, but the block's header was trashed. You can continue from this error with ↑N from TeleRaid, but it is symptomatic of array trashing, and you should save your state as soon as possible and restart in a good sysout.
 - 9330 Reference counting problem: an object marked as having a overflowed reference count (greater than 62) is not found in the overflow table.
 - 9331 Reference counting problem: an object whose reference count just now overflowed was already in the overflow table.
 - 9332 Reference counting problem: an attempt was made to decrement the reference count of an object whose reference count was already zero.
 - 9333 One of a number of consistency checks in the process manager failed.
 - 9334 The process manager needed to build a function frame for some operation, but failed. This normally should never happen, but could conceivably if you are about to completely fill up the stack (and thus would otherwise get a 9319 error).

- 9335 Occurs at boot time when the sysout you are trying to run uses the full 32MB virtual address space, but you are trying to run it on a machine that can only address 8MB. The function 32MBADDRESSABLE reports whether a machine has the hardware required to address 32MB.
- 9336 Somehow control was transferred to the T frame at the top of the world (effectively a (RETTO T), except that RETTO turns that case into a RESET), thereby evading the process world. This leaves the stack in an unresumable state.
- 9337 The process that is being scheduled to run next has had its stack released—inconsistent state in the process scheduler.
- 9393 (Koto Only) See 9341
—*Post-Koto Error Codes*—
- 9338-40 Error in locked page logic—not currently used.
- 9341 Hard disk error in the swapper—the swapper has tried several times to access a page of the virtual memory backing file and failed; page fault cannot proceed.
- 9342 Disk run table for the virtual memory backing file is malformed.

9400-9899 unassigned

9900-9924 Attempt to call Raid or 1132 Subr.

The only code normally seen in this range is 9915:

- 9915 Call to RAID. Note that if you have the Raid interrupt enabled (typically on ↑C), you will get a 9915 error by typing that interrupt character, which you can continue by typing ↑N from TeleRaid. Any other occurrence of 9915 generally signifies an error in system code that has not been explicitly assigned a code in the 9300 range.

LocalFile

1108 Local Hard Disk File System

Intermezzo Release

Stored as [eris]<lisp>intermezzo>doc>localFile.tedit

Introduction

The 1108 hard disk file system is designed to provide Interlisp-D users with a flexible mechanism for storing and accessing files. Like the file systems for the 1100 and 1132, the 1108 file system supports features like random access and version numbers on files. In addition the 1108 local file system supports a hierarchical naming structures for files.

Partitioning the Disk

The hard disk used with an 1108 may be partitioned into up to ten regions called logical volumes. Logical volumes are like directories on the disk device: they may be used to hold Interlisp virtual memories, Interlisp files, Mesa files, or Star files. You can partition the disk with the *Installation Utility* floppy or with *Othello*. Since partitioning the hard disk erases all its contents, you are advised to partition the disk appropriately before storing anything on it; otherwise, you will have to offload all files from the disk, repartition, and then copy the files back to the disk.

Although an Interlisp virtual memory file could coexist on a logical volume with other files, it is generally advisable to give each virtual memory file a logical volume which it does not share with anything else. Otherwise the resulting fragmentation would adversely affect swapping performance. A logical volume intended to contain an Interlisp virtual memory should be between 16,000 and 64,000 disk pages long.

The Intermezzo file system (unlike its predecessors) allows Interlisp user files to coexist on a logical volume that contains Mesa or Star files. So it is no longer necessary to have a special logical volume given over to Lisp user files, though you still can have one if you like. Note that to store Interlisp files on a logical volume, you must create a Lisp directory on that volume (see below for instructions).

File System Utility Functions

So long as there is a logical volume with a Lisp directory on it, you will have a local disk device called {DSK}. This device can be used from within Interlisp-D just like the {DSK} device on the 1100 and 1132, except that it supports a hierarchical naming structure for files.

If you do not have a logical volume with a Lisp directory on it, Interlisp will emulate the {DSK} device by a coredevice, which is fine except: (a) The coredevice provides limited scratch space for some system programs; (b) when running GREET, Interlisp will fail to find {DSK}INIT.LISP and will have to prompt the user for an init file; and (c) since the coredevice is contained in virtual memory, it (and the files stored on it) can last only as long as you keep your virtual memory image.

To create a Lisp user file directory on a logical volume, call the function

(CREATEDSKDIRECTORY volumeName) [function]

CREATEDSKDIRECTORY affects only the specified volume, and (unlike previous versions of the file system) does not affect Mesa or Star files in the specified volume. CREATEDSKDIRECTORY returns the name of the directory created. Installing an Interlisp directory is something that should have to be done only the first time the logical volume is used. After that, the system will automatically recognize and open access to the logical volumes that have Interlisp directories on them.

Should you ever want to get rid of a Lisp directory (and all the files in it), call the function

(PURGEDSKDIRECTORY volumeName) [function]

PURGEDSKDIRECTORY affects only the Lisp files on the specified volume; it will not tamper with Mesa or Star files on the same volume. An alternative but cruder way to get rid of a Lisp directory is to use *Othello* or the *Installation Utility* to Erase the entire logical volume.

To find out if a particular logical volume already has a Lisp directory on it, call

(LISPDIRECTORYP volumeName) [function]

To find out what logical volumes you have on your local disk, call the function

(VOLUMES) [function]

To find out the total size of a logical volume in disk pages, call

(VOLUMESIZE volumeName) [function]

To find out the number of free pages left on a volume, call

(DISKFREEPAGES volumeName recompute) [function]

And to find out which logical volume contains the virtual memory you are currently running in, call the function

(DISKPARTITION) [function]

Finally, once an Interlisp directory has been installed on a logical volume, any program running in Lisp has access to the Lisp files on the the volume. Access is provided through the usual device independent file interface: CONN (to connect to any directory or subdirectory on the local disk), OPENSTREAM, CLOSEF, DELFILE, GETFILEINFO, SETFILEINFO, BIN, BOUT, LOAD, etc.

File Name Conventions

Each logical volume with a Lisp directory on it serves as a directory of the device {DSK}. Files are referred to as

`{DSK}<logical-volume-name>file-name`

Thus the file `Init.lisp` on the volume `LispFiles` would be called `{DSK}<LISPPFILES>INIT.LISP`.

In addition, you can indicate subdirectories using the `>` character in file names to delimit subdirectory names. Subdirectories allow users to group files to a finer degree of granularity. Files with subdirectories are written

`{DSK}<logical-volume-name>subdir1>...>subdirN>file-name`

For example, suppose you had a file `LRdesign.tedit` on the subdirectory `ParserGenerator` on the subdirectory `Compiler` on the directory (logical volume) `LispFiles` of the hard disk device; its name would be written `{DSK}<LISPPFILES>COMPILER>PARSERGENERATOR>LRDESIGN.TEDIT`.

You can default directory names for the 1108 hard disk in an unusual but simple way. That is: *if the file does not have a subdirectory and you leave out the directory (logical volume) name, the directory will default to the next logical volume which has a Lisp directory on it including or after the volume containing the currently-running virtual memory.* Thus if your disk has logical volumes `Lisp`, `Tajo`, and `LispFiles`, and the `Lisp` volume contains the running virtual memory, and only the `LispFiles` volume has a `Lisp` directory on it, then `{DSK}INIT.LISP` will refer to the file `{DSK}<LISPPFILES>INIT.LISP`. All the utility functions presented above default logical volume names in a similar way, except for those that can't, such as `CREATEDSKDIRECTORY`. If you want to find out what the default `Lisp` directory is, call

`(DIRECTORYNAME '{DSK})`

This defaulting convention is necessitated by several parts of the Interlisp system which create scratch files on the device `{DSK}` without specifying a directory (logical volume).

Displaying File System State

`DSKDISPLAY` is a library package which provides a display window for 1108 local file system. The window keeps track of what logical volumes you have on your local disk, which ones have valid `Lisp` directories on them, and how much space is left on each volume.

The file system display can be in one of three states: `ON`, `OFF`, or `CLOSED`. `ON` means the display window is updated whenever the file system state changes. (This continuous updating can slow down the file system significantly.) `OFF` means that the display window is open, but updated only when the user left-clicks it with the mouse. `CLOSED` means that the display window is closed and never updated. When the `DSKDISPLAY` package is loaded, the display mode is set to `CLOSED`.

The functional interface to the file system display is provided by

`(DSKDISPLAY newState) [function]`

DSKDISPLAY returns the old state of the file system display, and if newState is one of the litatoms ON, OFF, or CLOSED, then the display state will be changed to newState.

The mouse interface to the file system display is as follows: left-buttoning the display window will update it, and middle-buttoning the window will bring up a menu which allows you to change the display state.

Scavenging

Unlike previous releases, Intermezzo provides full disk scavenging service to guard against the unlikely event of file system failure. There are two classes of file system failure: Lisp directory failure, or lower-level (Pilot) failure. Scavenging service for Lisp directories is provided by the library package SCAVENGEDSKDIRECTORY; scavenging for Pilot is provided by either *Othello* or the *Installation Utility*.

Pilot failures manifest themselves as a "HARD DISK ERROR" break within Lisp. To fix such a failure, return to top level, log out of Lisp, get into either *Othello* or the *Installation Utility*, use the Scavenge option on the logical volume in question, and then reboot Lisp.

Lisp directory failures show up as infinite looping or other aberrant behavior while doing a directory search or enumeration. To repair the directory, return to top level, load the package SCAVENGEDSKDIRECTORY, and call

(SCAVENGEDSKDIRECTORY volumeName) [function]

Should you have any doubt which logical volumes to scavenge, scavenge them all. Should you not be sure which scavenger to use on a volume, use them both, Pilot first, then Lisp directory. Neither scavenger will harm an intact volume.

LocalFile Implementor's Guide
Filed as [eris]<lispcore>internal>doc>localFileImpl.tedit
Written by Tayloe Stansbury 18-Feb-85
Modified by Tayloe Stansbury 15-Aug-85 12:27:46

The Dandelion/Dove local file system is implemented in two files: LocalFile and DskDisplay. User-level documentation for the file system can be found in the 1108 User's Guide or on [eris]<lispcore>doc>localfile.tedit.

The Dandelion/Dove local file system has two layers: the Pilot layer and the Lisp streams layer. The Pilot layer emulates a subset of the Pilot file system, as described in the Pilot Programmer's Manual. The Lisp streams layer implements the Lisp streams specification laid out in [eris]<lispcore>internal>doc>streams.tedit.

The Pilot layer is implemented by three modules in the file LocalFile:

1. LFFALLOCATIONMAPCOMS, which keeps track of which pages have been allocated and which are free. LFFallocationMap provides the functionality of [idun]<apilot>11.0>pilot>private>volallocmapimpl.mesa, though its implementation is only very loosely based on that file.
2. LFFILEMAPCOMS, which keeps track of the mapping between file ID numbers and runs of disk pages. This mapping is stored in a specialized B-tree. LFFileMap provides the functionality of [idun]<apilot>11.0>pilot>private>volfilemapimpl.mesa, though its implementation was based more on [idun]<apilot>100>pilot>private>volfilemapimpl.mesa, and later updated to be compatible with the Mesa 11.0 release of Pilot.
3. LFPILOTFILECOMS, which has a primitive notion of file, as embodied in its datatype FileDescriptor. LFPilotFile handles things like creating, extending, shrinking, and deleting files; reading and writing file pages; labels; and volume root directories (which map file types onto higher level directories -- e.g. Lisp file type -> Lisp directory ID, Mesa file type -> MFile directory ID, etc.). LFPilotFile does not emulate any particular Mesa file, but rather grew up as the gray area between the two layers became more well-defined during the evolution of the Lisp local file system.

The Lisp stream layer is defined by three more modules in the file LocalFile:

1. LFDIRECTORYCOMS, which implements the Lisp directory. The Lisp directory maps symbolic Lisp file names onto Pilot file ID numbers, and handles directory search and directory enumeration.
2. SCAVENGEDSKDIRECTORYCOMS, which implements a scavenger for the Lisp directory. It works by purging the old Lisp directory, creating a new one, using the BTree to figure out what Lisp files there are on the volume, using the leader page of each Lisp file to figure out what its name is, and then inserting an entry in the new directory for each Lisp file. There is no Pilot-level scavenger implemented in Lisp; for that we rely on the Othello Scavenge Logical Volume command.
3. LFCOMS, which implements all other operations of the local disk file device. LocalFile uses Pilot files as backing files for Lisp streams: page 0 of the Pilot file becomes the stream's leader page (containing stuff GETFILEINFO and the scavenger will be interested in), page 1 of the Pilot file becomes page 0 of the stream, etc. Pilot backing files may be longer than the Lisp stream they hold.

In addition, the file DskDisplay provides a window which displays file system status. This file is separate because it relies on the window system and therefore must be loaded considerably later in the loadup process than LocalFile need be.

Some future projects for the file system (apart from fighting off the stream of ARs):

1. Modify the READPAGES and WRITEPAGES methods to transfer contiguous pages all at once, and set the MULTIBUFFERHINT to be T.
2. Rewrite DiskDlion (which implements the Dandelion/Dove disk head) so that disk requests that cross cylinder boundaries are handled in runs rather than a page at a time (for both the Dandelion and Dove). Without this, large disk requests (which happen especially during deleting) can tie up the system for quite a while.
3. Modify DiskDlion to do a process switch while waiting for the disk. Currently it busy waits.
4. Currently too much of the file system is uninterruptable. Unfortunately, the primitive UNINTERRUPTABLY now prevents process switch in addition to preventing keyboard interrupts. What we really need is a construct that will do the latter but not the former. (There is now an AR for such a beast.) Given that, we should remove as many calls to UNINTERRUPTABLY as possible.
5. Currently files are allocated 20 pages at a time, regardless of whether the openfile request came with a size hint. This was originally because allocating long files sometimes took long enough that NS connections got dropped. Once #2 is accomplished, long allocations should be quite a bit faster though; and #s 3 and 4 will make it possible for other processes to sneak in while allocations are going on. Then it would make sense to change the file system to allocate files all at once when a hint is provided.

6. Currently, allocation map buffers and file map buffers are written out only once per stream creation or deletion. Without automatic built-in scavenging, that strikes me as a bit unsafe (although it does speed things up some). It would probably be better to write them out once every allocation or deallocation (and the performance penalty for doing so will not be too great if allocation requests are larger as a result of #5).

7. Longer term: rewrite the directory so that it uses some form of tree search. The linear search currently used gets unacceptably slow for large directories.

Should you have any questions, do not hesitate to contact me. My mail address is Stansbury.pa, and my extension is 4330. Good luck!

Making a Patch for Distribution

- (1) Fix the Ar(s) in <Lispcore>Sources>, <Lispcore>Library>, the microcode, or the emulator, whichever is appropriate.

For Library-file patches:

- (2) Move the entire LCOM/DFASL file(s) onto <Lispcore>Patches>.
- (3) In the Workaround section of each AR you've fixed, note the name of the file(s) you moved onto the patch directory, and the date/time you moved them.

For Lisp system-code patches:

- (2) Decide which functions, variables, etc. are needed for the patch file. Determine what pre-requisite patches (if any) need to be applied before this one makes sense.
- (3) Make sure you have the latest RELEASETOOLS loaded (the Medley sysout as Cheryl made it doesn't).
- (4) Use the "Patch" command to build a patch-file template:
patch *ar1 ar2* ...
where *ar1*, *ar2*, etc. are the AR numbers of the ARs this patch will fix.
Patch will create the COMS for a file named **AR-*ar1*-PATCH**, and will bring them up in an SEdit window for you to edit.
- (5) Edit the COMS for the patch file to include all the functions, variables, etc. that are needed.
- (6) Connect to {Eris}<Lispcore>Patches>.
- (7) MAKEFILE and compile the patch file there.
- (8) In the Workaround field for each Ar you fixed in this patch, note the name of the patch file and the date & time you made it.

For microcode patches:

For emulator patches:

Hardware:

- Bigger Disks
- Faster local IFSSs
- Color display
- Sell better peripherals
 - Laser writer
 - better file server
 - SCSI
- Labelless Disks
- Streaming Tape

Documentation:

- Updated and online doc
- Improved documents
- Consistent paradigm for writing documents (IM formatting replacement)
- Make system easy to learn

Window system / User interface:

- Interruptable pop-up Menus
- TIP tables
- Remote Windows
- Faster text display
- Imager
- Image Streams
 - clean up code
 - collapse common code
 - add clipping
- Rethink & overhaul Image Objects
- Loopsified window system
- Consistent look & feel (window UI and commands)
 - User interface as consistent as Viewpoint or Macintosh
 - Same keying or mousing gets same op everywhere
- Cheap, Pervasive Text (TTYIN replacement)
- Symbol ESC completion on type-in

Programming Environment:

- Same machine low-level debugger
- Source level debugger
- Good stepper
- NS based Teleraid
- Teleraid and CL stack ops
 - revert
 - return from
- Teleraid Inspect
- Interactive Interface to File manager
- Better File Browser
- Definition groups
- Extensible SEdit
- Programmers Interface to SEdit
- More Common Lisp integration

Higher level language features:

- Unification
- Single, Common prettyprinter
- Fast prettyprinter
- Path name cleanup
 - use pathnames all the way down
 - get SAME object always for same file
- Single Compiler
- Decouple DWIM - CLISP
- Fast sequence functions
- Ropes

System building and installation:

- Non-Dorado loadups
- FASL in the init
- Packages in the init
- Common Lisp primacy, eg in the init
- State saving smaller than sysout-sized
- Revamp software installation process
 - better user interface
 - written in Lisp, not Mesa
 - FAST
 - include error Recovery

Language kernel:

- Big Reference counts w/microcode support
- Full GC (circular objects)
- Get arith to IEEE compliance
- GCable Symbols

- Unboxed floating point compler
- Finalization of GCed objects
- Smaller base sysout
- More Common Lisp microcode, eg EVAL
- Common Loops IN the system

Operating system:

- Pre-emptive scheduler
- Non-consing Synchronizers (monitorlocks)

Device drivers:

- SNA networking
- Floppy speed
- Improve local disk speed
- Real Subdirectories
- Support File Cacher
- Reliable TCP & RS232
- Run in XNS only world
- Run in Stand Alone World
- Sun NFS support
- XNS over phone lines

Applications:

- Video Image Manipulation
- Bitmap Editor as good as MacPaint
- Mail support for
 - X.400
 - SMTP
- Release Lafite
- Lisp based File Server
- 3270 terminal emulator
- Good 3rd party software development path
- Database
- Spreadsheet
- AR category clean up
- Programming system Management Tools
- Adobe Tools
- Notecards
- TEdit
 - Footnotes
 - Index
 - toc
 - egns
 - styles
 - LPT output
 - change masks
 - better than Tex
 - WYSIWG Page mode
 - DES/DIS/DIF/VP convert
- Easier Landscape Printing
- YACC and such
- C, Fortran, Pascal, Cobol

Work environment:

- Work at Home
- Better design process

NS Character Set Issues:

Greg Nuyens. Aug 16,1984

This note describes a proposal for incorporating the NS character set standard into Interlisp-D. This proposal will encompass the following areas:

- implementation of a "long" character data type

It specifically omits the following:

- external file representations. **

- the reworks necessary to fonts, etc. to attain DIG, though this proposal is made with these issues in mind.

Definitions

The following terms are defined for use in the remainder of the document.

character: an instance of the "long" character data type.

byte: an 8 bit unsigned integer corresponding to the current Interlisp-D character.

character set: the (conceptual) array of 256 characters into which all the representable distinct characters have been separated. An example is character set 0 which contains most but not all of the characters currently used in Interlisp. For instance an umlaut is not included in character set 0.

font: A style in which characters are rendered. A font includes (conceptually) an image for every representable character ($2^{16} - 256$). A font is a member of the cross product of Family, Weight, Slope, Face, Nominal Size, Rotation, and Expansion. An example is TimesRoman 12 Bold Italic Regular Compressed with 90 degree rotation. To contrast character set and font, note that only one character set (0 ??*) contains the greek letter alpha, but every font contains a rendering for that character. In practice few (no?) fonts will contain all the images of all the characters.

Overview

Before discussing how these functions will change, an overview of the scheme for the internal functions:

Error in IMAGEOBJ
GETFN: SKIO.GETFN

There will be two types of character data, the previously described long and short characters. However, any single string will consist of only one type of character. Thus strings as currently represented, (effectively byte arrays) will continue to exist. However, there will also be arrays of long characters. Any functions which receive arguments of mixed type (e.g. RPLCHAR given an string of short chars and a long char to replace with) will produce results composed of long chars only.

Thus, the current implementation of substrings (as "tails" of strings) will not suffice, since the original string may be coerced upward to a long char string by an operation occurring after a substring has been returned. The plan is to use forwarding pointers in string space and store substrings as <header,offset,length> triplets. Thus when a short char string is coerced upward into a long char string, the original header will have a pointer to the new location. Any substrings returned previous to the coercion will still be valid, since they reference the (now indirect) original header. The offsets will always be scaled by a bytes-per-character value implicit in the type of string (1 for short strings, 2 for long strings).

This scheme guarantees the advertised property of substrings that they are indeed shared tails. That is, a destructive change to a substring will affect the string. This will be true regardless of any coercion of the string that occurs.

Also recognize that any coercions performed on substrings will change the representation of the whole string.

Changes to Interlisp Functions

In previous discussions the two following lists of functions affected by this proposal were identified:

First the functions which deal with the internal representation of character data:

RPLSTRING(X N Y)

this function must change so that if either argument is "long", both are coerced to long. For the character argument, this is simply to add the default character set. For the string however, it will be necessary to copy the string coercing each character by appending the default character set.

RPLCHARCODE(X N CHARCODE)

must now take long or short charcodes, handling them as will RPLSTRING.

GNC{CODE}(X)

will always return an atom representing a long character (Does this mean having all the single character atoms? NO--they can be MKATOMed on the fly.) {or for GNCCODE, a long charcode}

GLC{CODE}(X)

as above

NTHCHAR{CODE}(X N FLG RDTBL)

will always return an atom representing a long character.

NCHARS(X FLG RDTBL)

Returns number of characters independent of representation. When FLG is T the prin-2 length (as yet unspecified for long chars) will be used.

STRPOS{L}(PAT STRING START SKIP ANCHOR TAIL) (and MAKEBITTABLE)

If either PAT or STRING is constructed of long chars, then the comparison will take place as though both were long. However, no destructive changes will be made.

CHARACTER(N)

Always produces an atom whose printname is the long character whose representation is N.

CHCON(X FLG RDTBL)

Still produces a list of charcodes. These may be short or long charcodes

SUBSTRING(X N M OLDPTR)

Performs the substring operation, guaranteeing the EQ invariant for substrings. The internal rep'n will be the header together with the offset and length, so that if upward conversion later happens, shared tails remain.

ALLOCSTRING(N INITCHAR OLD)

As before except, that if INITCHAR > 255 then the resulting string will be a long string. (Coercing the OLD string's char array as needed.)

MKSTRING(X FLG RDTBL)

As before, except the PName may be long.

CONCAT{LIST}{U ...} {L*}

If any of the arguments are long strings, the result is a long string.

STREQUAL

This will be true when the characters are the same, regardless of the representation.

and the following functions which must know the external representation of character data

FILEPOS

SETFILEPTR

BIN

READC

\INCHAR

\OUTCHAR

COPYBYTES

COPYCHARS

Efficiency Concerns

the common case of bytes stays fast (though some penalty)

readc is already coercing bytes upward into smallp's why not into long chars?

will it be necessary to have all $2^{16} - 256$ unit length pname atoms exist.
(what does readc currently return)?

The intention is that the common case of short char arguments will retain their current efficiency.

For Immediate Action:

The following opcodes are not emitted, have no microcode, and can be immediately recycled:
(Someone should volunteer to do this and announce to xclispcore)

Opcode#(octal)	Name
007	CDDR ;; unwind
036	PUTHASH ;; findkey
041	BOUT ;; still seems to be emitted
043	LIST1 ;; restlist
044	DOCOLLECT ;; there is apparently a holdout that emits this.
045	ENDCOLLECT ;; ditto.
177	AUDIO ;; retcall
374	RESERVED for Dolphin

The following opcodes are emitted, but have no microcode -- and may be recycled after recompilation of all sources. (Someone should volunteer to remove all optimizers -- Macros, Dopvals, Dopcodes, etc. -- associated with these opcodes and announce to xclispcore)

Opcode#(octal)	Name
033	GETPROP (?)
035	GETHASH
050	ELT
051	NTHCHC
052	SETA
053	RPLCHARCODE
055	EVALV
160	ATOMNUMBER
313	GETBASEFIXP
314	PUTBASEFIXP

Recycling these opcodes demands recompilation, but this might be an additional reason for declaring .DCOM files not readable in Lute.

For Discussion and Design:

Uncoordinated changes:
(that is, changes which do not alternate the meaning of existing constructs)

(Order of tasks is random -- not priority order)

Approx. time	Task
Day	Add opcode for Read-Char (like NTHCHC, BIN) convert (byte/word) to (Characterp/Smallp)
Day	Add opcode for = Compare two numbers for equality -- does coercions
Day	Add opcode for ASH (arithmetic shift) Arithmetically shift integer X bits to left if X pos. or X bits to right if X neg.
Day	Port LISTGET to 1186 8K microcode
Day	CL versions for ASSOC, FMEMB, EQUAL, (MEMBER) Differ from IL versions in that EQL is used rather than EQ --- microcode may only do EQ test on Symbols and punt otherwise
Day	Debug EQL, ARG0 -- insure correct and efficient algorithm
Week	Complete port of unboxed scalar F.P. opcodes to 1186 (includes basic arith., and comparison)
Week	Microcode more cases for RECLAIMCELL RECLAIMCELL currently only reclaims Cons's -- should be extended to FIXP and Floatp boxes.

Month	Microcode additional CL arith. operators (Truncate and friends) Spec. here not yet clear -- will appear after benchmarking
?	CL:EVAL (like EVAL but for commonlisp)
?	REF
Month	Finish array microcode

Coordinated changes:
(that is, changes which require changes both to lisp and microcode)

(Order of tasks is random -- not priority order)

Approx. time -----	Task ----
Week	CREATECELL countdown Modify GC so reclaim occurs after countdown to zero from some settable start - Important for tighter control over GC -
Month	GC hash algorithm changes Address issues of GC hash table overflow - Important for Cons'y benchmarks -
Week	Subtyping in TYPEP Make TYPEP more useful May render TYPEMASK.N redundant
Week	Replace DTEST and TYPECHECK to be more TYPEP like (two type bytes rather than one?)
Month +	FN call changes for &optional, &key, &rest, closures and arg # checking, field descriptors, multiple values, discriminators (Larry, Bill and Pavel have volunteered to work on this)

Interlisp-D Opcodes

Written by: Masinter, van Melle, Sybalsky
Stored on [Eris]<LispCore>Internal>Doc>Opcodes.TEdit
RamVersion: 26,24 (recently incremented)
LispVersion: 113400

```

/*****
/*
*/
/*      Copyright 1989, 1990 Venue
/*
/*
/*
/*      This file was work-product resulting from the Xerox/Venue
/*      Agreement dated 18-August-1989 for support of Medley.
/*
/*
/*
/*****

```

UFNs—Handling undefined op-codes

When the microcode (or C emulator) doesn't handle an opcode, it "punts" to the UFN for that opcode: a Lisp function that does what the opcode should do.

To find out what function to call, the microcode looks at a 256-cell block of storage called the "UFN table" (pointed to in Lisp by `\UFNTable`). The UFN table contains, for each opcode

(FNINDEX WORD)	Atom number (really “definition index”) of the function to be called.
(NEXTRA BYTE)	# of extra bytes to be pushed as argument to the UFN (either 0, 1, or 2).
(NARGS BYTE)	# of arguments to call the UFN function with.

The Op-code descriptions

In multibyte opcodes ($\text{len}-1>0$), alpha is byte 1, beta is byte 2, and gamma is byte 3.

TOS refers to the argument on the top of the stack; TOS-1 is arg one back, etc.

@[x] is the contents of the word pointed to by x.

#	name	len-1	stk level effect	UFN table entry
0	-X-			

used only to denote end of function, never executed.

#	name	len-1	stk level effect	UFN table entry
1	CAR	0	0	\CAR.UFN

If arg not LISTP

```

    If NIL, return NIL
    else call UFN

```

If cdr code=0, follow indirect pointer. Take value of car field & return it. (Cons cells are 32 bits: first 8 are cdr code, rest are "carfield")

[required by diagnostics (except car[NIL]); implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
2	CDR	0	0	\CDR.UFN

If arg not LISTP
 if NIL, return NIL
 else call UFN

if cdr code=0, follow indirect pointer
 if cdr code=200q, return NIL
 elseif cdr code gt 200Q, CDR is on same page as cell,
 in cell page+2*(cdr code-200Q)
 else CDR is contained in cell at PAGE+2*(cdr code).
 (Cons cells are 32 bits: first 8 are cdr code, rest are "carfield")

[required by diagnostics (except cdr[NIL]); implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
3	LISTP	0	0	LISTP

Return arg if LISTP (NTYPX=LISTPType), else NIL

[required by diagnostics; implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
4	NTYPX	0	0	NTYPX

Return type number of arg (right half of word at MDSTypeTable + [tos rsh 9])

[required by diagnostics; implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
5	TYPEP	1	0	\TYPEP.UFN

return arg if type=alpha byte, else NIL

[required by diagnostics; implemented in all ucodes; similar to LISTP]

#	name	len-1	stk level effect	UFN table entry
6	DTEST	2	0	\DTEST.UFN

return arg if typename=(alpha,beta), else call UFN or atom number 372 (\DTESTFAIL) with tos and (alpha,beta). (typename is word 0 of type's DTD; DTD is DTDBase+(type# lsh 4))

[required by diagnostics; implemented in all ucodes]

#	name	len-1	stk level effect	UFN table entry
{ 7	CDDR	0	0	CDDR

TAKE CDR Twice [not currently used or implemented or emitted.]}

REPLACED BY :

7	UNWIND	?	?	\UNWIND.UFN

(N is the alpha byte, KEEP is the beta byte) Unwinds the dynamic stack of the current frame to absolute stack depth N, performing any unbinding indicated by bind marks found along the way. If KEEP is 0, the original top of stack is discarded, otherwise it is pushed after unwinding everything else. This opcode is essentially the same as UNBIND or DUNBIND, except that you stop when the stack depth is N, rather than stopping as soon as you have processed the first bind mark.

The stack depth N is measured in cells (doublewords) starting at the base of the pvar region. N=0 means the stack is utterly empty (including the pvar region; i.e., the end of stack pointer (pointer to next stack block) would be the same as PV). Of course, N=0 cannot be used at all in the present architecture, since there is always at least a quadword pad between the frame header and the start of the dynamic stack. If we get rid of that quadword, then N=0 could have meaning in a frame that had an empty pvar region, though that is not true of any closure target, the current sole user of this opcode.

Note that taking the stack depth as alpha byte means this opcode cannot unwind to any deeper than depth 255. For sake of reference, the largest pvar region in Full.sysout is for the function \CURVE, whose pvar region is 92 cells long (59 locals and 32 fvars), which means it could still achieve a dynamic depth of an additional 173 cells before UNWIND would care (it actually never exceeds a depth of 30).

```
{let SP be the stack pointer; i.e., TOS = @SP}
TOP _loc[pvar0]-2 + 2*N
if KEEP neq 0
  then TEMP TOS
until (SP _ SP - 2) = TOP
  do if @SP is bind mark
    then perform its unbinding
if KEEP neq 0
  then push TEMP
```

#	name	len-1	stk level effect	UFN table entry
10	FN0	2	1	
11	FN1	2	0	
12	FN2	2	-1	
13	FN3	2	-2	
14	FN4	2	-3	

call fn (alpha,beta) with N args [required]

#	name	len-1	stk level effect	UFN table entry
15	FNX	3	FNX	

call fn (beta,gamma) with alpha args [required]

#	name	len-1	stk level effect	UFN table entry
16	APPLYFN	0	-1	

call fn (tos) with (tos-1) args after popping tos & tos-1 [required. Right now, it goes to \INTERPRETER if TOS isn't a litatom. May add requirement that will work with code blocks.]

#	name	len-1	stk level effect	UFN table entry
17	CHECKAPPLY*	0	0	\CHECKAPPLY*

If TOS is a literal atom whose definition cell has CCODEP on and ARGTYPE=0 or 2, return it, otherwise call UFN. Note that CHECKAPPLY* is always immediately followed by an APPLYFn. If it would save some time, you might be able to immediately jump to the APPLYFN code. (note: definition cell: bit 0 is CCODEP, bit 1 is "fast" {this fn has empty nametable}, bits 2-3 are ARGTYPE)

[not required; implemented on Dorado]

#	name	len-1	stk level effect	UFN table entry
20	RETURN	0	0	\HARDRETURN

```
do return except when:
  slow bit in returner is on
  and
  returnee usecount not 0
  or
```

returners BF usecount is not 0
 or
 returnee not immediately followed by
 a free block
 or
 the basic frame of the returner.

In any of those conditions, call UFN or context switch to hardreturn context (which?). [required]

#	name	len-1	stk level effect	UFN table entry
21	BIND	2		

push binding mark, bind variables, popping values of stack. [required]

alpha byte is [#NILS << 4 + #BINDS].

beta byte is [FirstPVAR], which is 1-origin (i.e., 0 is PVAR1?? it looks like --JDS)

BIND takes #BINDS values off the top of stack and binds FirstPVAR and successive PVARs to those values. It then sets the #NILS PVARs beyond that to NIL.

Finally, a "binding mark" is pushed on the top of the stack:

```
*-+-+-+~( #NILS + #BINDS )-+-+-+FirstPVAR << 1-+-+-+*
```

Binding marks are identified on the stack because they're negative: The high bit is guaranteed to be 1.

#	name	len-1	stk level effect	UFN table entry
22	UNBIND	0		

remember tos, pop until binding mark, unbind variables, push old tos [required]

#	name	len-1	stk level effect	UFN table entry
23	DUNBIND	0	(DUNBIND)	

pop until binding mark, unbind variables [required]

#	name	len-1	stk level effect	UFN table entry
24	RPLPTR.N	1	-1	\RPLPTR.UFN

deleteref value at @(tos-1)+alpha.

addref (tos)

store (TOS) at @(tos-1)+alpha [leave high byte of destination intact]

pop (return (TOS-1)).

If reference count failure, call GCTABLESCAN (atom ????) on punt [not required; in Dorado, 12K]

#	name	len-1	stk level effect	UFN table entry
25	GCREF	1	0	\HTFIND

perform ref count operation on TOS according to alpha byte:

0 - addref (add 1 to reference count)

1 - delref (subtract 1 from reference count)

2 - stkref (turn on "stack reference" bit)

If DELREF causes new refcnt to go to 0 & stk bit off, return arg, else always return NIL. On reference count failure, call UFN (no GCTABLESCAN). [not required; in D0, Dorado]

#	name	len-1	stk level effect	UFN table entry
26	ASSOC	0	-1	ASSOC


```

        word 15:      NEXTPAGE
\CDR.NIL= 200q

```

LISTPDTD is the DTD for type LISTP, i.e., at DTDbase + (LLSH 5 4)

Subroutine MAKECONSCELL[page] (given page, return new cell from it):

```

    new cell is at page + page:nxtcell
    new CNT is old CNT - 1; punt if CNT was zero
    new NXTCELL is new cell's cdr code

```

Subroutine NEXTCONSPAGE:

```

    if LISTPDTD:NEXTPAGE # 0 then return it, else punt
    (lisp code scans for page with cnt>1)

```

CONS(X Y) // note: this may not be right. Check sources for truth

If Y is NIL:

```

    get NEXTCONSPAGE
    MAKECONSCELL on it
    store new cell with \CDR.NIL in cdrcode (hi byte)
    X in rest of cell

```

Elseif Y is a listp and the CNT in Y's page > 0, then

```

    MAKECONSCELL[Y's page]
    store X as CAR, CDR code = ((LOLOC Y) and 377q] rsh 1) + 200q

```

Else:

```

    get NEXTCONSPAGE
    MAKECONSCELL on it
    store Y in new cell (hi byte 0)
    (remember this as Z)

    MAKECONSCELL on same page
    store X in new cell, with hi byte= [(LOLOC Z) and 377q] rsh 1

```

ADDREF X

ADDREF Y

increment LISTPDTD:COUNTER

DELREF result

[not required, in Dorado, 12K]

#	name	len-1	stk level effect	UFN table entry
33	CMLASSOC	0	-1	CL::%%SIMPLE-ASSOC

Takes to two arguments off the stack and returns the of the simplest case of cl:assoc. Equivalent to ASSOC opcode, except punts if the key argument is not an immediate datum. [not required, not implemented on 4K, Dorado]

#	name	len-1	stk level effect	UFN table entry
34	FMEMB	0	-1	FMEMB

if TOS=NIL, return NIL

if TOS is not LISTP, call UFN

if (CAR TOS)=TOS-1, return TOS

else TOS_(CDR TOS), do jump to . [i.e., iterate]

Be sure to allow interrupts.

[not required; in 12K]

#	name	len-1	stk level effect	UFN table entry
35	CMLMEMBER	0	-1	CL::%%SIMPLE-MEMBER

Takes to two arguments off the stack and returns the of the simplest case of cl:member. Equivalent to FMEMB opcode, except punts if the key argument is not an immediate datum. [not required, not implemented on 4K, Dorado]

#	name	len-1	stk level effect	UFN table entry
{ 36	PUTHASH	0	-2	PUTHASH

[not required, not implemented]}

REPLACED BY :

36 FINDKEY

```

alpha = arg#
tos = key
for z from arg# to numargs - 1 by 2
  if arg(z) = key then return(z + 1)
return(NIL)

```

#	name	len-1	stk level effect	UFN table entry
37	CREATECELL	0	0	CREATECELL

Create a new cell of type TOS (a smallposp):
 DTD _ DTDSPACE + (type lshift 4)
 NewCell _ DTD:FREE (2 words)
 DTD:FREE _ @(NewCell) (2 words)
 if DTD:FREE is now NIL, signal a gc punt at end of opcode
 increment DTD:COUNTER, signal a gc punt if counter goes negative
 Zero out DTD:SIZE words starting at NewCell (always an even number)
 Deleteref NewCell
 TOS _ NewCell

[not required; in Dorado, D0]

#	name	len-1	stk level effect	UFN table entry
40	BIN	0	0	\BIN

If TOS is not of type STREAM (13q) then PUNT

Format of stream is (only some fields are used by microcode):

```

word 0: COFFSET          ; a byte offset from BUFFER
word 1: CBUFSIZE         ; size of input buffer in bytes
word 2&3: flags [byte] = READABLE (bit 0), WRITABLE (bit 1),
                      EXTENDABLE (bit 2), DIRTY (bit 3),
                      PEEKEDCHARP (bit 4), ACCESSBITS (bit 5-7)
                      BUFFER [24 bits] ; pointer to data
word 4: BYTESIZE
      CHARSET            ; 8 bits each
word 5: PEEKEDCHAR       ; valid when PEEKEDCHARP true
word 6: CHARPOSITION
word 7: CBUFMAXSIZE      ; maximum size of output buffer

```

If COFFSET >= CBUFSIZE then PUNT [buffer overflow]

If READABLE is off then PUNT

Fetch and remember the byte at BUFFER + COFFSET[byte offset]

Note that this address is guaranteed to be valid at this point,
 but it could pagefault.

Update the stream:

store COFFSET _ COFFSET + 1

Return the remembered byte as a small positive number.

[not required; in Dorado, 12K]

#	name	len-1	stk level effect	UFN table entry
41	BOUT	0	-1	\BOUT

If TOS-1 is not of type STREAM (13q) then PUNT. (see format under BIN)

If TOS is not a small positive number (< 400Q) then PUNT.

if WRITABLE is off then PUNT

```

if BUFFER is NIL then PUNT
if COFFSET >= CBUFMAXSIZE then PUNT
deposit byte from TOS at BUFFER + CCOFF[byte offset]
Update the stream:
    store COFFSET _ COFFSET + 1
    set DIRTY flag to 1 [if it isn't already]
return the smallposp one (1)
[not required; not implemented; not even generated by compilers (3/13/89)]

```

#	name	len-1	stk level effect	UFN table entry
42	PROLOGOPDISP	0	0	none

Implements the Prolog Opcode Dispatch. Uses the Prolog registers PC, N, USQbase, and uSQtablebase.

It takes one arg (DEST). In pseudo-RTL:

```

if smallp(DEST) then PC _ PC + DEST
else PC _ DEST
N _ logand(PC ^ 00FF'x)
opcode _ lrrsh(PC ^ 8)
if uLMBase(opcode) = 1 then
{ LispPC _ USQbase + uSQtablebase(opcode)
  return to Lisp }
else
{ PC _ PC + 1
  (run microcode version) }

```

#	name	len-1	stk level effect	UFN table entry
{ 43	LIST1	0	0	CONS

(perform (CONS TOS NIL)) not required, not implemented}

REPLACED BY :

43 RESTLIST

```

alpha = skip -- number of args to skip
tos = last -- last arg#
tos-1 = tail
IF tail = NIL THEN
    page _ NEXTCONSPAGE
    GOTO make
ELSE
    AddRef tail
    page _ CONSPAGE[tail]
    GOTO make
make:
    get [cnt,,next] from page
make1:
    tail _ CONSCell (CAR = IVar(last), CDR = tail)
    AddRef IVar(last)
    IF skip = last THEN GOTO fin
    last _ last - 1
    GOTO make1
noroomonconspage:
fin:
    store updated [cnt,,next]
    update ListpDTD:COUNTER
    DelRef tail
    IF noroomonconspage THEN UFN
    ELSEIF ListpDTD:COUNTER overflow then GCPUNT
    ELSEIF overflow entries then GCHANDLEOVERFLOW
    ELSE NEXTOPCODE

```

#	name	len-1	stk level effect	UFN table entry

44 MISCN 2 1 + (-n) \MISCN.UFN

Miscellaneous opcode for opcodes needing n args from the stack. The alpha byte contains the sub-opcode number and the beta byte contains the number of arguments on the stack. This opcode was added specifically for bytecode emulated implementations, where the opcodes could be written in C. This opcode provides the same functionality of the SUBRCALL opcode, except it has the added flexibility of having the opcodes UFN (on both Suns & D-Machines). The UFN vectoring routine is written to adjust the stack according to the number of arguments stated in the beta byte, and there is a UFN handler for each sub-opcode. The opcode is generated using the (MISCN NAME & REST ARGS) macro & optimizer defined in LLSUBRS. The NAME parameter must be registered in \MISCN-TABLE-LIST list, which is of the form (name index ufn-name). The \INIT-MISCN-TABLE function initializes the MISCN's sub-opcode UFN vector.

The predefined MISCN sub-opcodes are as follows:

index	name	function
0	USER-SUBR	This is for the user-supplied subr C coded subrs. It contains its own sub-opcode division based on the 1st argument on the stack. Like MISCN, USER-SUBR requires that the user-subrs be registered with the variable \USER-SUBR-LIST (name index ufn) by calling the \INIT-USER-SUBR-TABLE function. Thus user-defined subrs can each have thier own ufn handler which will be indexed through the MISCN & USER-SUBR UFN mechanism. This opcode can be generated using the (USER-SUBR NAME & REST ARGS) macro found in LLSUBRS.
1	CL:VALUES	Return multiple values
2	CL: SXHASH	Common Lisp hash-bits function for EQUAL hash-tables
3	CL: EQLHASHBITSFN	[Not currently implemented]
4	STRINGHASHBITS	IL hash-bits function for STREQUAL harrayp's
5	STRING-EQUAL-HASHBITS	IL hash-bits function for String-EQUAL harrayp's
6	CL:VALUES-LIST	Return a list of multiple values.

To reserve new MISCN & USER-SUBR entries, you should set the global values for \MISCN-TABLE-LIST and \USER-SUBR-LIST in the LLSUBRS file & re-write the file to insure that you will have unique numbers. The function WRITECALLSUBRS whould also be called to generate a new subrs.h file, which contains the C constant definitions for the proper indexes in the C code.

The args to the MISCN UFN routines consist of (INDEX ARG-COUNT ARG-PTR), where INDEX is your sub-opcode number, ARG-COUNT is the number of args to be found on the stack, and ARG-PTR is a pointer to the 1st arg found on the stack. The rest of the args can be found by using (\ADDBASE ARG-PTR (LLSH n 1)) for the n-1th arg.

USER-SUBR UFNs have similar args of (USER-SUBR-INDEX ARG-COUNT ARG-PTR), where USER-SUBR-INDEX is the user-subr sub-opcode index, and ARG-COUNT & ARG-PTR are the same as in MISCN UFNs.

CAUTION: Since the stack affect is variable, thus not known to the compiler, the optimizer may do something funny to the stack args around your call. You should check the emitted code to be sure that things compiled correctly. Putting your calls in small functions will help.

#	name	len-1	stk level effect	UFN table entry
45	<unused>	0	-1	(was ENDCOLLECT)

[not required; not implemented, will be eliminated]

#	name	len-1	stk level effect	UFN table entry
46	RPLCONS	0	-1	\RPLCONS

takes two args (LST ITEM):

check (LISTP LST)

 LST's pages CNT field # 0 (see CONS above),

 LST's cdrcode = 200q.

call UFN if any of these are not true

MAKECONSCCELL on LST's page

store ITEM as in cell, with cdr code = 200q (\CDR.NIL)
 store as LST's new cdrcode (((LOLOC newcell) and 377) rsh 1) + 200q.
 ADDREF item
 increment LISTPDTD:COUNTER
 return new cell
 [not required; in 12K]

#	name	len-1	stk level effect	UFN table entry
50	ELT	0	-1	ELT

(ELT array index)

Check if TOS-1 is type ARRAYP, call UFN if not
 Check if TOS is smallpos, call UFN if not

Array descriptor:

word 0,1: Flags(8) , , base(24)
 Flags = Orig(1), unused(1), Readonly(1), unused(1), type(4)
 word 2: Length
 word 3: Offset

Compute index = (TOS) - Orig

if index < 0 or index >= length, call UFN.

index _ index + Offset

dispatch on type (note that index*2 may overflow):

[0] (byte) return (GETBASEBYTE base index)
 [1] (smallpos) return (GETBASE base index)
 [2] (fixp) return 32 bits at base+index*2 as a fixp (possibly smallp)
 [3] (hash) return (GETBASEPTR base index*2)
 [4] (code) same as byte
 [5] (bitmap) same as smallpos
 [6] (pointer) return (GETBASEPTR base index)
 [7] (float) return 32 bits at base+index*2 as a floatp
 [11.] (double-pointer) same as hash
 [12.] (mixed) same as hash

[not required; not implemented yet]

#	name	len-1	stk level effect	UFN table entry
51	NTHCHC	0	-1	NTHCHARCODE

Same as ELT, except type of TOS-1 is STRINGP, the type of the array is always 0, and (optionally) return NIL instead of calling UFN when index is out of range. [not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
52	SETA	0	-2	SETA

(SETA array index value)

Check array and compute index as with ELT.

If ReadOnly is true, call UFN.

In all cases, leave value on stack on exit.

Dispatch on type:

[0] (byte) perform (PUTBASEBYTE base index value)
 [1] (smallpos) perform (PUTBASE base index value)
 [2] (fixp) unbox integer value, deposit 32 bits at base+index*2
 [3] (hash) perform (RPLPTR base+index*4 value)
 [4] (code) same as byte
 [5] (bitmap) same as smallpos
 [6] (pointer) perform (RPLPTR base+index*2 value)
 [7] (float) unbox float value, deposit 32 bits at base+index*2
 [11.] (double-pointer) same as hash
 [12.] (mixed) same as hash

[not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
53	RPLCHARCODE	0	-2	RPLCHARCODE

[SPECIFICATION INCOMPLETE]

[not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
54	EVAL	0	0	\EVAL

takes single argument ARG

If ARG=NIL, T, or smallp, return ARG

If ARG is an atom, attempt free variable lookup:

If bound, return value

If top value is not NOBIND (atom #1), return top value

else ufn-punt

[optional: if ARG is FIXP, FLOATP, return ARG]

[optional: if ARG is LISTP, punt to \EVALFORM (atom 370q)]

else ufn-punt

[not required; in Dorado, 4K]

#	name	len-1	stk level effect	UFN table entry
55	(was EVALV)			

#	name	len-1	stk level effect	UFN table entry
56	TYPECHECK.N	1	0	\TYPECHECK.UFN

identical to DTEST; only UFNs different

#	name	len-1	stk level effect	UFN table entry
57	STKSCAN	0	0	\STKSCAN

TOS is VAR.

If TOS is not litatom, punt.

Returns 24 bit pointer to cell where VAR is bound.

Note: must check VAR=NIL, and return pointer to NIL's value cell. (Free variable lookup algorithm fails if given NIL, at least on Dorado.)

If variable was bound on stack, the value returned will be a pointer into stack space. If variable is not bound, value will be pointer to top level value cell.

[not required; in Dorado (I think), not in DLion? In Maiko emulator]

#	name	len-1	stk level effect	UFN table entry
60	BUSBLT	1	-3	\BUSBLT.UFN

Talks to the BusMaster peripheral adapter.

Alpha bytes:

0	WORDSOUT
1	BYTESOUT
2	BYTESOUTSWAPPED
3	NYBBLESOUT
4	WORDSIN
5	BYTESIN
6	BYTESINSWAPPED
7	NYBBLESINSWAPPED

[not required; in 12K only]

#	name	len-1	stk level effect	UFN table entry
61	MISC8	1	-7	\MISC8.UFN

Miscellaneous opcode for operations needing 8 args.

Alpha bytes:

Alpha	name	function
0	IBLT1	special-purpose halftone-drawing routine for spectrogram creation
1	IBLT2	ditto

[not required; in 12K only]

#	name	len-1	stk level effect	UFN table entry
62	UBFLOAT3	1	-2	\UNBOXFLOAT3

in 12K only

Alpha bytes:

0	POLY
1	3X3
2	4X4
3	133
4	331
5	144
6	441

for matrix multiply, polynomial evaluation
alpha byte 7: Unboxed ASET

#	name	len-1	stk level effect	UFN table entry
63	TYPEMASK.N	1	0	\TYPEMASK.UFN

similar to TYPEP, except checks if high byte of type table AND with alpha is non-zero, returns TOS if so, NIL otherwise.

#	name	len-1	stk level effect	UFN table entry
64	PROLOGREADPTR			
65	PROLOGREADTAG			
66	PROLOGWRITETAGPTR			
67	PROLOGWRITE0PTR			
70	PSEUDOCOLOR			
72	EQL			

#	name	len-1	stk level effect	UFN table entry
73	DRAWLINE	0	-8	\DRAWLINE.UFN

takes 8 (!) args from top of stack, does line draw inner loop

#	name	len-1	stk level effect	UFN table entry
74	STORE.N	1	0	\STORE.N.UFN

takes quantity at TOS and stores it at TOS-alpha.

#	name	len-1	stk level effect	UFN table entry
75	COPY.N	1	1	\COPY.N.UFN

pushes quantity at (TOS-alpha/2). COPY.N 0 = COPY

#	name	len-1	stk level effect	UFN table entry
76	RAID	0	0	RAID

[used only for UFN]

#	name	len-1	stk level effect	UFN table entry
77	\RETURN			

used only for UFN for LLBREAK

#	name	len-1	stk level effect	UFN table entry
100-106	IVAR	0	1	
push IVAR#(opcode-100) [required]				

#	name	len-1	stk level effect	UFN table entry
107	IVARX	1	1	
push IVAR#alpha [required]				

#	name	len-1	stk level effect	UFN table entry
110-116	PVAR	0	1	
push PVAR#(opcode-110) [required]				

#	name	len-1	stk level effect	UFN table entry
117	PVARX	1	1	
push PVAR#(alpha) [required]				

#	name	len-1	stk level effect	UFN table entry
120-126	FVAR	0	1	
127	FVARX	1	1	
Push the indicated FVAR. [required]				

#	name	len-1	stk level effect	UFN table entry
130-136	PVAR_	0	0	
137	PVARX_	1	0	
Set the indicated PVAR from tos, do not pop. [required]				

#	name	len-1	stk level effect	UFN table entry
140	GVAR	2	1	
Push @(VALSPACE+2*(alpha,beta))				

[required; may want to change to check if high order bit on, and UFN]

#	name	len-1	stk level effect	UFN table entry
141	ARG0	0	0	\ARG0
check TOS smallp, call UFN if not check TOS between 1 and #args in current function replace TOS with value of Ith variable, counting from 1 [to do range check, must fetch flags; if not fast, fetch BLINK. #args is computable from difference of BLINK and IVAR] [not required; not implemented yet]				

#	name	len-1	stk level effect	UFN table entry
142	IVARX_	1	0	

store TOS as new value of IVAR alpha
[required]

#	name	len-1	stk level effect	UFN table entry
143	FVARX_	1	0	

free variable assignment. When value cell is global, perform GVAR_ operation
[can call \SETFREEVAR.UFN (atom# ???) instead]

#	name	len-1	stk level effect	UFN table entry
144	COPY	0	1	

push TOS again
[required]

#	name	len-1	stk level effect	UFN table entry
145	MYARGCOUNT	0	1	\MYARGCOUNT

Push as a smallpos the number of arguments in current frame.
See ARG0. (probably should use common subroutine)
[not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
146	MYALINK	0	1	

Returns stack-index of beginning of ALINK of current frame.
This pushes the "ALINK" field of the current frame, with the low
bit turned off less ALINK.OFFSET (= 12Q).
[required]

#	name	len-1	stk level effect	UFN table entry
147	ACONST	2	1	

Push {0, (alpha,beta)}
[required]

#	name	len-1	stk level effect	UFN table entry
150	'NIL	0	1	
151	'T	0	1	
152	'0	0	1	
153	'1	0	1	

Push the indicated constant.
[required]

#	name	len-1	stk level effect	UFN table entry
154	SIC	1	1	
155	SNIC	1	1	
156	SICX	2	1	

Push:
alpha as a smallposp,
alpha as a smallneg (extend leftward with 1's),
(alpha,beta) as smallposp, respectively.
[required]

#	name	len-1	stk level effect	UFN table entry
157	GCONST	3	1	

Push {alpha, (beta,gamma)}

[required]

#	name	len-1	stk level effect	UFN table entry

160 ATOMNUMBER 2 1
 same as SICX. Different opcode for benefit of code walkers.
 [required]

#	name	len-1	stk level effect	UFN table entry

161 READFLAGS 0 0
 \READFLAGS
 TOS is a virtual page# as a smallposp
 TOS _ virtual memory flags of that page, as a smallposp
 Flags are:
 bit 0: referenced
 bit 2: write-protect
 bit 3: dirty
 Vacant is denoted write-protect + dirty
 [This is the same as XNovaOp ReadFlags, with AC0 -> loloc[TOS]]
 [required]

#	name	len-1	stk level effect	UFN table entry

162 READRP 0 0
 \READRP
 TOS is a virtual page# as a smallposp
 TOS _ the corresponding real page, as a smallposp
 [This is the same as XNovaOp ReadRP, with AC0 -> loloc[TOS]]
 [required]

#	name	len-1	stk level effect	UFN table entry

163 WRITEMAP 0 -2
 \WRITEMAP
 TOS-2 is a virtual page# as a smallposp
 TOS-1 is a real page as a smallposp
 TOS is a word of flags as a smallposp
 Make the indicated virtual page# be associated with the given
 real page, with status flags. Real page is immaterial if flags = VACANT
 Return the virtual page #
 [This is the same as XNovaOp SetFlags, with AC0 -> loloc[TOS-2],
 AC1 -> loloc[TOS-1], AC2 -> loloc[TOS]]
 [*not yet in Dorado]

#	name	len-1	stk level effect	UFN table entry

164 READPRINTERPORT 0 +1
 \READPRINTERPORT
 TOS _ current value from printer port, as a smallposp
 Ufn if machine cannot do this.
 [not in 4k]

#	name	len-1	stk level effect	UFN table entry

165 WRITEPRINTERPORT 0
 0 \WRITEPRINTERPORT
 Printer _ TOS, interpreted as a smallposp
 Ufn if machine cannot do this.
 [not in 4k]

#	name	len-1	stk level effect	UFN table entry

166 PILOTBITBLT 0 -1
 \PILOTBITBLT
 Performs Pilot-style bitblt.
 TOS is constant zero, which can be used for maintaining state.
 TOS-1 is a pointer to a bitblt table, which is 16-aligned.
 [not required, implemented]

#	name	len-1	stk level effect	UFN table entry
167	RCLK	0	0	\RCLKSUBR

Store into words pointed to by TOS the processor clock [up to 32 bits, left justified].
[required]

#	name	len-1	stk level effect	UFN table entry
170	MISC1	1	0	\MISC1.UFN
171	MISC2	1	-1	\MISC2.UFN

These are miscellaneous opcodes that dispatch on alpha to provide infrequent and/or machine-specific operations. To save microcode space (currently), the two opcodes share the same dispatch table, i.e., the alpha's do not overlap. There are two opcodes principally so that there can be a reasonable ufn handler: MISC1 takes 1 arg, MISC2 takes

2. Current values for alpha:

- 0 STARTIO[bits] Currently only for Dolphin ethernet. Perform the "StartIO" function with bits given as smallp TOS. (Resets Ethernet to known quiet state).
- 1 INPUT[devreg] Perform input from some device. TOS is smallp device register specification (on Dolphin: 4 bits of task, 4 bits of device reg; on DLion: 4 bits absolute). Returns TOS = smallp value input from device.

DLion codes:

```

for INPUT {alpha = 1 mod 4}
TOS = 00 mod 16, _ EIData
TOS = 01 mod 16, _ EStatus
TOS = 02 mod 16, _ KIData
TOS = 03 mod 16, _ KStatus
TOS = 04 mod 16, _ uSTATE
TOS = 05 mod 16, _ MStatus
TOS = 06 mod 16, _ KTest
TOS = 07 mod 16, MP code 9122
TOS = 08 mod 16, _ Version
TOS = 09 mod 16, <12K> _ BusExt L <4K> MP code 9122
TOS = 10 mod 16, <12K> _ BusExt M <4K> MP code 9122
TOS = 11 mod 16, <12K> _ uFLmode <4K> MP code 9122
TOS = 12 mod 16, MP code 9122
TOS = 13 mod 16, MP code 9122
TOS = 14 mod 16, MP code 9122
TOS = 15 mod 16, MP code 9122
2     OUTPUT[value, devreg]     Perform output to some device. TOS is smallp
device register spec as with INPUT; TOS-1 is the smallp
value to output.
for DLion:
for OUTPUT {alpha = 2 mod 4}
TOS = 00 mod 16, <12K> BusExt L _ <4K> IOPOData _
TOS = 01 mod 16, IOPctl _
TOS = 02 mod 16, <12K> uFLmode _ <4K> KOData _
TOS = 03 mod 16, Kctl _
TOS = 04 mod 16, EOData _
TOS = 05 mod 16, EICtl _
TOS = 06 mod 16, DCtl _
TOS = 07 mod 16, uBBTime _ {display rate}
TOS = 08 mod 16, uLispOptions _
TOS = 09 mod 16, Pctl _
TOS = 10 mod 16, Mctl _
TOS = 11 mod 16, <12K> BusExt M _ <4K> MP code 9120
TOS = 12 mod 16, EOctl _
TOS = 13 mod 16, KCmd _
TOS = 14 mod 16, <12K> PPort _ <4K> MP code 9120
TOS = 15 mod 16, POData _
9 Dorado only, RWMUFMAN

```

#	name	len-1	stk level effect	UFN table entry

172 RECLAIMCELL 0 0 \GCRECLAIMCELL

Check type of TOS; let DTD be pointer to DTD of this type

If not LISTP then punt

Reclaim list:

```

code_PTR:cdrcode
if (code and 200q) = 0 then punt      [or optional: if code = 0 then punt]
FreeListCell(PTR)
val_ deleteref(PTR:carfield)         * deleteref CAR
if code # \CDR.NIL
then PTR_PTR:pagebase + (code lsh 1) * point to cdr or lvcdr
  [if (code and 200q) = 0             * optional
   then FreeListCell(PTR)             * cdr indirect--free cell
   PTR_GetBasePtr(PTR)]
  if deleteref(PTR)                   * deleteref CDR
  then val_PTR
return val

```

FreeListCell(PTR):

```

PAGE_ address of PTR's page
if PAGE:Nextpage < 0 then punt        * only when page was full
PTR:cdrcode _ PAGE:nextcell
PAGE:nextcell _ word# of PTR
PAGE:count _ PAGE:count + 1

```

How to reclaim other types, roughly (needs type table change):

```

if Type bit "ok to reclaim" is off, call UFN
store DTD:FREELST in first two words of DATUM
store DATUM in DTD:FREELST

```

[not required; implemented for Listp on D0, non-listp on Dorado?, ? for 12K]

#	name	len-1	stk level effect	UFN table entry

173	GCSCAN1	0	0	\GCSCAN1

scan HTMAIN from (TOS)-1 to 0 for a cell with
collision bit on or else stack bit & reference cnt both are 0
if none found, return NIL
else return new index.

```

note: design allows NWWInterrupts to be processed
note: can actually perform GCRECLAIMCELL on the
cell indicated if stack bit off and ref cnt=0)

```

[not required; in all]

#	name	len-1	stk level effect	UFN table entry

174	GCSCAN2	0	0	\GCSCAN2

```

similar to GCSCAN1, but scan for word
with collision bit on or stack bit on.
Note: can optionally turn stack bit off, check if
count is 1 and zero entry, continue scanning
Note: design allows NWWInterrupts to be processed

```

[not required; in all]

#	name	len-1	stk level effect	UFN table entry

175	SUBRCALL	2		

Call Bcpl subr number alpha with beta arguments.

The following have some microcode on the DLion:

```

17'b Raid
15'b Logout
06'b BackGround
11'b DspBout
20'b Pup
22'b SETSCREENCOLOR
23'b ShowDisplay

```

#	name	len-1	stk level effect	UFN table entry
176	CONTEXT	0	0	\CONTEXTSWITCH

switch to context (TOS).

#	name	len-1	stk level effect	UFN table entry
177	(was audio)			

[not required; not currently implemented]

#	name	len-1	stk level effect	UFN table entry
200-217	JUMP	0	JUMP	
220-237	FJUMP	0	CJUMP	
240-257	TJUMP	0	CJUMP	
260	JUMPX	1	JUMP	
261	JUMPXX	2	JUMP	
262	FJUMPX	1	CJUMP	
263	TJUMPX	1	CJUMP	
264	NFJUMPX	1	NCJUMP	
265	NTJUMPX	1	NCJUMP	

Assorted jumps. The offset of the jump is given in the succeeding bytes, sign-extended to the left in the case of the single-byte offsets. The offset is relative to the start of the instruction. The opcodes with implicit offset run from +2 thru +21q.

JUMP* are unconditional.

FJUMP* and TJUMP* perform the jump only if TOS is NIL or non-NIL, respectively.

NFJUMPX and NTJUMPX perform the jump only if TOS is NIL or non-NIL, respectively.

Additionally, they pop the stack only if the jump is not taken.

[required]

#	name	len-1	stk level effect	UFN table entry
266	AREF1	0	-1	%AREF1

Perform a one-dimensional array access:

(AREF1 array index)

- 1.) Check that array is a oned-array -- if not punt
- 2.) Check that $0 \leq \text{index} < \text{total size for array}$
- 3.) Compute (index + offset for array)
- 4.) Extract base, and type number -- and pass base, type number, index + offset to array-read subroutine and return result on top of stack.

#	name	len-1	stk level effect	UFN table entry
267	ASET1	0	-2	%ASET1

Perform a one-dimensional array set:

(ASET1 new-value array index)

- 1.) Check that array is a oned-array -- if not punt
- 2.) Check that $0 \leq \text{index} < \text{total size for array}$
- 3.) Compute (index + offset for array)
- 4.) Check array not read-only
- 5.) Extract base, and type number -- and pass newvalue, base, type number, index + offset to array-write subroutine and return newvalue on top of stack.

#	name	len-1	stk level effect	UFN table entry
270-276	PVAR_↑	0	-1	
Store TOS into indicated PVAR, pop stack. [required]				

#	name	len-1	stk level effect	UFN table entry
277	POP	0	-1	
Pop stack. [required]				

#	name	len-1	stk level effect	UFN table entry
300	POP.N	1	(variable)	
POP (alpha+1) elements off top of stack, POP.N 0 = POP, POP.N 1 = POP POP, etc.				

#	name	len-1	stk level effect	UFN table entry
301	ATOMCELL.N	1	0	
if TOS is atom (0,,low), then replace with (alpha,,low+low), with carry into alpha. This is used for getting the PList, Def, Val cell of litatoms. If TOS HI is not 0, call UFN. This will allow assigning definitions, plists and values to non-litatoms.				

#	name	len-1	stk level effect	UFN table entry
302	GETBASEBYTE	0	-1	\GETBASEBYTE
Retrieve byte at offset TOS from (TOS-1).				

#	name	len-1	stk level effect	UFN table entry
303	INSTANCEP	2	0	\INSTANCEP.UFN
return T if typename is subtype of (alpha,beta), else return NIL. (typename is word 0 of type's DTD; DTD is DTDBase+(type# lsh 4) not locked down supertype is word 15 of DTD, 0 means no supertype) [currently only in 12k Dandelion]				

#	name	len-1	stk level effect	UFN table entry
304	BLT	0	-2	\BLT
(BLT destinationaddr sourceaddr nwords) Move nwords from source to destination. If nwords < prespecified constant (currently 10q), then operation is uninterruptable, else must be prepared to service interrupts. On page fault or interrupt, update stack according to how much is moved, and back up pc. Words are moved right to left (high addresses to low), if it makes a difference. Result is unspecified.				

#	name	len-1	stk level effect	UFN table entry
305	MISC10	2	-9	\MISC10.UFN
Perform miscellaneous operation on 10 arguments. alpha operation 0 PIXELBLT [not required; in 12k only]				

#	name	len-1	stk level effect	UFN table entry
306	(unused)			

#	name	len-1	stk level effect	UFN table entry
307	PUTBASEBYTE	0	-2	\PUTBASEBYTE
Store TOS at offset TOS-1 from (TOS-2), punting if TOS is not smallposp. Currently ucode punts and ufn errors if offset isn't a smallp.				

#	name	len-1	stk level effect	UFN table entry
310	GETBASE.N	1	0	\GETBASE

TOS _ @(TOS+alpha) as a smallposp.

#	name	len-1	stk level effect	UFN table entry
311	GETBASEPTR.N			

TOS _ 24-bit pointer @(TOS+alpha).

#	name	len-1	stk level effect	UFN table entry
312	GETBITS.N.FD			

take 1 arg on stack (PTR) and 2 bytes (n, fd). fetches the "field" fd from the word PTR + n. fd is a mesa field descriptor: the left 4 bits is the number of the "first" bit of the field, while the right 4 bits is the width of the field-1. E.g., 0:17 is the full word, 0:0 is the leftmost bit.

#	name	len-1	stk level effect	UFN table entry
313	Unused			

#	name	len-1	stk level effect	UFN table entry
314	CMLEQUAL	0	-1	CL: EQUAL

Takes two arguments off the stack and performs some cases of the cl:equal predicate. Punts if either argument is a not an immediate datum or a number.

[not required, not implemented on 4K and Dorado]

#	name	len-1	stk level effect	UFN table entry
315	PUTBASE.N	1	-1	\PUTBASE.UFN

Store TOS as word at location (TOS-1)+alpha

Pop (Return TOS-1).

Punt if TOS not smallposp. Note that UFN will specify extra byte for punt.

#	name	len-1	stk level effect	UFN table entry
316	PUTBASEPTR.N	1	-1	\PUTBASEPTR.UFN

Takes (PTR, NEWVAL) on stack, leaves PTR on stack, stores NEWVAL at PTR+N . (note: no punt case)

#	name	len-1	stk level effect	UFN table entry
317	PUTBITS.N.FD	2	-1	\PUTBITS.UFN

Takes (PTR, NEWVAL) on stack, stores bits of NEWVAL at FD field of PTR+N. Returns PTR.

Punt (UFN) if NEWVAL is not smallposp.

#	name	len-1	stk level effect	UFN table entry
320	ADDBASE	0	-1	\ADDBASE
321	VAG2	0	-1	\VAG2
322	HILOC	0	0	
323	LOLOC	0	0	

as before

#	name	len-1	stk level effect	UFN table entry
324	PLUS2	0	-1	PLUS

325	DIFFERENCE	0	-1	DIFFERENCE
326	TIMES2	0	-1	TIMES
327	QUOTIENT	0	-1	QUOTIENT

(same as I- versions, except UFN different. Optionally perform as F- opcode if one of arguments is floating.)

#	name	len-1	stk level effect	UFN table entry
330	IPLUS2	0	-1	\SLOWIPLUS2
331	IDIFFERENCE	0	-1	\SLOWIDIFFERENCE
332	ITIMES2	0	-1	\SLOWITIMES2
333	IQUOTIENT	0	-1	IQUOTIENT
334	IREMAINDER	0	-1	IREMAINDER

unbox TOS & TOS-1

(if SmallPos then 0,,loloc, if SmallNeg then -1,,loloc,
ttypest if FIXP then fetch 32 bit quantity)

perform 32x32 operation, and then

if overflow occurs, punt

[used to say: call OFLOWMAKENUMBER (atom ???)

with result mod 2^{32} as two 16 bit smallposps.

This can't work; what did we mean?]

If no overflow:

if hi part 0, return SmallPosHi,,lo

if hi part -1, return SmallNegHi,,lo

else need to return large integer. Two choices:

1) set up as if in call to MAKENUMBER (atom ???) with 2 args being

Hi and Lo part of result, as smallposps; or

2) Perform CREATECELL of type FIXP, and then store results

in generated box; return new box

[only smallpos x smallpos required on IPLUS, IDIFFERENCE;

Current implementation status:

Only smallpos x smallpos on ITIMES in both microcodes

only smallpos/smallpos for REMAINDER, QUOTIENT in Dorado]

#	name	len-1	stk level effect	UFN table entry
335	IPLUS.N	1	0	\SLOWIPLUS2

add TOS+alpha

#	name	len-1	stk level effect	UFN table entry
336	IDIFFERENCE.N	1	0	\SLOWIDIFFERENCE

subtract TOS-alpha

#	name	len-1	stk level effect	UFN table entry
337	unused			

#	name	len-1	stk level effect	UFN table entry
340	LLSH1	0	0	\SLOWLLSH1
341	LLSH8	0	0	\SLOWLLSH8
342	LRS1	0	0	\SLOWLRS1
343	LRS8	0	0	\SLOWLRS8

unbox TOS, perform 32 bit operation and box results

as with 2 arg fns

[smallposp -> smallposp required, can UFN in other cases]

#	name	len-1	stk level effect	UFN table entry
344	LOGOR2	0	-1	\SLOWLOGOR2

345	LOGAND2	0	-1	\SLOWLOGAND2
346	LOGXOR2	0	-1	\SLOWLOGXOR2

see IPLUS etc above

[smallposp -> smallposp required, can UFN in other cases]

[32x32 bit implemented in Dorado, D0]

#	name	len-1	stk level effect	UFN table entry
347	LSH	0	-1	LSH

shift TOS-1 arithmetically by TOS.

#	name	len-1	stk level effect	UFN table entry
350	FPLUS2	0	-1	FPLUS2
351	FDIFFERENCE	0	-1	FDIFFERENCE
352	FTIMES2	0	-1	FTIMES2
353	FQUOTIENT	0	-1	FQUOTIENT

[not required; in Dorado, 12K]

#	name	len-1	stk level effect	UFN table entry
354	UBFLOAT2	1	-1	\UNBOXFLOAT2

alpha bytes:

0 ADD x+y

1 SUB x-y

2 ISUB y-x (currently unused)

3 MULT x*y

4 DIV x/y

5 GREAT x>y (returns T/NIL rather than unboxed floating)

6 MAX (max x y) currently unused

7 MIN (min x y) currently unused

8 REM (x remainder y), i.e. x-(floor x/y)*y

9 (UBAREF A I)

Same as AREF1, except that this one returns an unboxed number

implementations: Dorado has GREAT only 12K has all but REM

#	name	len-1	stk level effect	UFN table entry
355	UBFLOAT1	1	0	\UNBOXFLOAT1

alpha byte:

0 BOX (tos -> floating box (tos))

1 UNBOX (tos -> floating unbox (tos), float if FIXP)

2 ABS (currently unused)

3 NEGATE (currently unused)

implemented all on 12K

#	name	len-1	stk level effect	UFN table entry
356	AREF2	0	-2	%AREF2

Perform a two-dimensional array access:

(AREF2 array i j)

1.) Check that array is a twod-array -- if not punt

2.) Check that $0 \leq i < \text{bound0}$

3.) Check that $0 \leq j < \text{bound1}$

4.) Compute $(j + i * \text{bound1})$

5.) Extract base, and type number -- and pass base, type number, $(j + i * \text{bound1})$ to array-read subroutine and return result on top of stack

#	name	len-1	stk level effect	UFN table entry
357	ASET2	0	-3	%ASET2

Perform a two-dimensional array set:
(ASET2 newvalue array i j)

- 1.) Check that array is a twod-array -- if not punt
- 2.) Check that $0 \leq i < \text{bound0}$
- 3.) Check that $0 \leq j < \text{bound1}$
- 4.) Compute $(j + i * \text{bound1})$
- 5.) Check array not read-only
- 6.) Extract base, and type number -- and pass base, type number, $(j + i * \text{bound1})$ to array-write subroutine and return newvalue on top of stack.

#	name	len-1	stk level effect	UFN table entry
360	EQ	0	-1	

return T or NIL if (tos)=(tos-1)

#	name	len-1	stk level effect	UFN table entry
361	IGREATERP	0	-1	\SLOWIGREATERP
362	FGREATERP	0	-1	FGREATERP

[IGREATERP required; FGREATERP not implemented]

#	name	len-1	stk level effect	UFN table entry
363	GREATERP	0	-1	GREATERP

Same as IGREATERP (see PLUS, etc)
[not required]

#	name	len-1	stk level effect	UFN table entry
364	EQUAL	0	-1	EQUAL

If args are EQ, return T
If either arg is litatom, return NIL
else call UFN
[not required; not implemented]

#	name	len-1	stk level effect	UFN table entry
365	MAKENUMBER	2	-1	MAKENUMBER

TOS-1 and TOS are smallposp's denoting the hi and lo halves of a 32-bit number.

Return a fixp that represents it:

```
If loloc[TOS-1] = 0
  then return SmallPl,,loloc[TOS]
elseif loloc[TOS-1] = 177777q
  then return SmallNeg,,loloc[TOS]
else CREATECELL[\FIXP]
Store loloc[TOS-1] and loloc[TOS] as its hi and lo halves
return the new cell
[implemented on 4K, Dorado]
```

#	name	len-1	stk level effect	UFN table entry
366	BOXIPLUS	0	-1	\BOXIPLUS
367	BOXIDIFFERENCE	0	-1	\BOXIDIFFERENCE

Same as IPLUS2, IDIFFERENCE, except store result @TOS -- first arg is number box (for which optionally check) -- and no overflow check.

#	name	len-1	stk level effect	UFN table entry
370	FLOATBLT	0	-3	\FLOATBLT

Miscellaneous floating point array ops; will eventually be renamed MISC5. Provides access to just about everything the Weitek FP chip does. Operates on two arrays; puts results in a third.
args: (BASE1, BASE2, DEST, N).

Alpha bytes:

0	FLOATWRAP
1	FLOATUNWRAP
2	FLOAT
3	FIX
4	FPLUS
5	FDIFFERENCE
6	FDIFFERENCE
7	(FPLUS (ABS source1) (ABS source2))
10	(ABS (FDIFFERENCE source1 source2))
11	(ABS (FPLUS source1 source2))
20	FTIMES

[not required; implemented on 1108X only]

#	name	len-1	stk level effect	UFN table entry
371	FFTSTEP	0	-1	\FFTSTEP

Takes FFTTABLE as TOS; performs one FFT step thereupon.

[not required; implemented on 1108X only]

#	name	len-1	stk level effect	UFN table entry
372	MISC3	0	-1	\MISC3.UFN

Miscellaneous 3-arg opcode.

Alpha bytes:

0	EXPONENT(source dest n) source is vector of floatps, dest is vector of words store exponent of source for n in dest
1	MAGNITUDE source is a vector of complex, dest is a vector of float store magnitude of source in dest
2	FLOAT source is a vector of word, dest is a vector of float float source & store in dest
3	COMP source is a vector of float, dest is a vector of complex spread source into dest, storing 0's.
4	BLKFMAX
5	BLKFMIN
6	BLKFABSMAX
7	BLKFABSMIN
8	FLOATTOBYTE source is vector of float (must have even number of elements), dest is vector of words
9	ARRAYREAD (base typenumber index) Dispatch on typenumber and perform a typed get.

[not required; implemented on 1108X only]

#	name	len-1	stk level effect	UFN table entry
373	MISC4	0	-1	\MISC4.UFN

Miscellaneous 4-arg opcode.

Alpha bytes:

0	TIMES
1	PERM
2	PLUS
3	DIFF

- 4 SEP
- 6 \BITMAPBIT bitmap x y newvalue (optional)
- 7 ARRAYWRITE (newvalue base typenumber index)

Dispatch on typenumber and perform a typed put
 [some confusion on how 0,2,3 different from corresponding TIMES, PLUS, DIF [not required;
 implemented on 1108X only]

#	name	len-1	stk level effect	UFN table entry
374	reserved on D0, UPCTRACE on Dorado			
375	SWAP	0	0	
376	NOP	0	0	
377	=			

Jan Pedersen
23 June 1986

Opcodes listed by entry in UFN table

op	'09	'08	'00	'32	'32L	'86-4	'86-8	Name
000	X	X	X	X	-	X	X	-X-
001	X	X	X	X	X	X	X	CAR
002	X	X	X	X	X	X	X	CDR
003	X	X	X	X	X	X	X	LISTP
004	X	X	X	X	X	X	X	NTYPX
005	X	X	X	X	X	X	X	TYPEP
006	X	X	X	X	X	X	X	DTEST {new COERCE}
007	-	-	-	-	-	-	-	CDDR {unused}
010	X	X	X	X	X	X	X	FNO
...	X	X	X	X	X	X	X	...
015	X	X	X	X	X	X	X	FNX
016	X	X	X	X	X	X	X	APPLYFN
017	-	-	-	-	X	-	-	CHECKAPPLY{?}
020	X	X	X	X	X	X	X	RETURN
021	X	X	X	X	X	X	X	BIND
022	X	X	X	X	X	X	X	UNBIND
023	X	X	X	X	X	X	X	DUNBIND
024	X	X	X	X	X	X	X	RPLPTR.N
025	X	X	X	X	X	X	X	GCREF
026	X	-	-	-	-	-	-	ASSOC
027	X	X	X	X	X	X	X	GVAR←
030	X	-	X	X	X	X	X	RPLACA
031	X	-	X	X	X	X	X	RPLACD
032	X	X	X	X	X	X	X	CONS
033	-	-	-	-	-	-	-	{unused}{will be GETPROP}
034	X	-	-	-	-	-	-	FMEMB
035	-	-	-	-	-	-	-	{unused}{will be GETHASH}
036	-	-	-	-	-	-	-	{unused}{will be PUTHASH}
037	X	-	X	X	X	X	X	CREATECELL
040	X	-	X	X	X	X	X	BIN
041	-	-	-	-	-	-	-	BOUT {unused}
042	P	-	-	-	-	-	-	{Prolog}OPFETCHPLUSOPDISP
043	-	-	-	-	-	-	-	{unused}{will be LIST1}
044	-	-	-	-	-	-	-	DOCOLLECT {unused}
045	-	-	-	-	-	-	-	ENDCOLLECT {unused}
046	X	-	X	-	-	X	X	RPLCONS
047	X	-	-	-	-	-	-	LISTGET
050	-	-	-	-	-	-	-	ELT
051	-	-	-	-	-	-	-	NTHCHC
052	-	-	-	-	-	-	-	SETA
053	-	-	-	-	-	-	-	RPLCHARCODE {unused}
054	X	X	X	X	X	X	X	EVAL
055	-	-	-	-	-	-	-	{unused}EVALV
056	X	X	-	-	X	X	X	TYPECHECK
057	X	X	X	?	X	X	X	STKSCAN
060	X	-	-	-	-	-	-	BUSBLT
061	X	-	-	-	-	-	-	MISC8{IBLT1 and IBLT2}
062	X	-	-	-	-	-	-	POLY {Poly; Mat. Multiply}
063	X	X	-	-	X	X	X	TYPEMASK.N
064	P	-	-	-	-	-	-	{Prolog}PROLOGREADPTR
065	P	-	-	-	-	-	-	{Prolog}PROLOGREADTAG
066	P	-	-	-	-	-	-	{Prolog}PROLOGWRITETAGPTR
067	P	-	-	-	-	-	-	{Prolog}PROLOGWRITEOPTR

247	X	X	X	X	X	X	X	TJUMP07
250	X	X	X	X	X	X	X	TJUMP10
...	X	X	X	X	X	X	X	...
257	X	X	X	X	X	X	X	TJUMP17
260	X	X	X	X	X	X	X	JUMPX
261	X	X	X	X	X	X	X	JUMPXX
262	X	X	X	X	X	X	X	FJUMPX
263	X	X	X	X	X	X	X	TJUMPX
264	X	X	X	X	X	X	X	NFJUMPX
265	X	X	X	X	X	X	X	NTJUMPX
266	X	-	-	-	-	-	-	ARRAYINDEX1
267	X	-	-	-	-	-	-	ARRAYINDEX2
270	X	X	X	X	X	X	X	PVAR0←
...	X	X	X	X	X	X	X	...
276	X	X	X	X	X	X	X	PVAR6←
277	X	X	X	X	X	X	X	POP
300	X	X	-	-	-	X	X	POP.N
301	X	X	-	-	-	X	X	ATOMCELL.N
302	X	X	X	X	X	X	X	GETBASEBYTE
303	-	-	-	-	-	-	-	{unused}
304	X	X	X	X	X	X	X	BLT
305	X	-	-	-	-	-	-	PIXELBLT
306	-	-	-	-	-	-	-	{unused}
307	X	X	X	X	X	X	X	PUTBASEBYTE
310	X	X	X	X	X	X	X	GETBASE.N
311	X	X	X	X	X	X	X	GETBASEPTR.N
312	X	X	X	X	X	X	X	GETBITS.N.FD
313	-	-	-	-	-	-	-	{unused}{new GETBASEFIXP}
314	-	-	-	-	-	-	-	{unused}{new PUTBASEFIXP}
315	X	X	X	X	X	X	X	PUTBASE.N
316	X	X	X	X	X	X	X	PUTBASEPTR.N
317	X	X	X	X	X	X	X	PUTBITS.N.FD
320	X	X	X	X	X	X	X	ADDBASE
321	X	X	X	X	X	X	X	VAG2
322	X	X	X	X	X	X	X	HILOC
323	X	X	X	X	X	X	X	LOLOC
324	X	X	X	X	X	X	X	PLUS2{see notes}
325	X	X	X	X	X	X	X	DIFFERENCE{see notes}
326	X	X	X	X	X	X	X	TIMES2{see notes}
327	X	X	X	X	X	X	X	QUOTIENT{see notes}
330	X	X	X	X	X	X	X	IPLUS2{see notes}
331	X	X	X	X	X	X	X	IDIFFERENCE{see notes}
332	X	X	X	X	X	X	X	ITIMES2{see notes}
333	X	X	X	X	X	X	X	IQUOTIENT{see notes}
334	X	X	X	X	X	X	X	IREMAINDER{see notes}
335	-	-	-	-	-	X	X	{unused}{IPLUS.N}
336	-	-	-	-	-	X	X	{unused}{IDIFFERENCE.N}
337	-	-	-	-	-	-	-	{unused}
340	X	X	X	X	X	X	X	LLSH1{see notes}
341	X	X	X	X	X	X	X	LLSH8{see notes}
342	X	X	X	X	X	X	X	LRSH1{see notes}
343	X	X	X	X	X	X	X	LRSH8{see notes}
344	X	X	X	X	X	X	X	LOGOR2{see notes}
345	X	X	X	X	X	X	X	LOGAND2{see notes}
346	X	X	X	X	X	X	X	LOGXOR2{see notes}
347	-	-	-	-	-	-	-	{unused}{new ALSH}
350	X	-	X	X	X	-	X	FPLUS2
351	X	-	X	X	X	-	X	FDIFFERENCE
352	X	-	X	X	X	-	X	FTIMES2
353	X	-	X	X	X	-	X	FQUOTIENT
354	X	-	-	-	-	-	-	UBFLOAT2 {UFADD, UFSUB, UFISUB, UFMULT, UFDIV, UGREAT, UMAX, UMIN, UREM}
355	X	-	-	-	X	-	-	UBFLOAT1 {UTOB, BTOU, UABS, UNEG, UFIX}
356	X	-	-	-	-	-	-	ARRAYREAD{GENERAL, UNBOXED}
357	X	-	-	-	-	-	-	ARRAYWRITE{GENERAL, UNBOXED}
360	X	X	X	X	X	X	X	EQ
361	X	X	X	X	X	X	X	IGREATERP
362	X	-	X	X	X	-	X	FGREATERP

363	X	X	X	X	X	X	X	GREATERP
364	X	X	?	?	X	-	X	EQUAL
365	X	-	X	X	X	-	X	MAKENUMBER
366	X	-	X	X	X	-	-	BOXIPLUS
367	X	-	X	X	X	-	-	BOXIDIFFERENCE
370	-	-	-	-	-	-	-	MISC5
371	X	-	-	-	-	-	-	FFTSTEP
372	X	-	-	-	-	-	-	MISC3
								{Floating Point Array ops: EXP, MAG, FLOAT, COMPLEX, BLKMAX, BLKMIN, BLKABSMAX, BLKABSMIN, FLOATTOBYTE}
373	X	-	-	-	-	-	-	MISC4
								{Floating Point Array ops: TIMES, PERM, PLUS, DIFFERENCE, MAGIC, BITMAPBIT}
374	-	-	?	-	X	-	-	{reserved for DOLPHIN}
375	X	X	X	X	X	X	X	SWAP
376	X	X	X	X	X	X	X	NOP
377	-	-	-	-	-	-	-	{unused}

notes:

4K microcode:

PLUS2, DIFFERENCE, TIMES2, QUOTIENT will ufn if args not INTEGERS

IPLUS2, IDIFFERENCE will accept FIXP's as arguments, but will ufn if result is not a smallp or smallneg

ITIMES2, IQUOTIENT, IREMAINDER will ufn if both args are not smallp

12K microcode:

PLUS2, DIFFERENCE, TIMES2, QUOTIENT will try floating point if args not INTEGERS

IPLUS2, IDIFFERENCE will accept FIXP's as arguments, and box the result if it is not a smallp or smallneg

ITIMES2, IQUOTIENT, IREMAINDER will ufn if both args are not smallp

Date: 24 Jun 86 14:36 PDT
 From: Masinter.pa
 Subject: Opcode survey
 To: Pedersen.pa
 cc: xclispcore↑

edits are in bold. We should meet to review what's in microcode and what the priority list should be. I suggest we do this at the end of the Common Lisp status meeting... does anyone have any objection to discussing it then?

OPCODES IMPLEMENTED IN MICROCODE BY MACHINE

Jan Pedersen

Key: '09 = 12K Dandelion
 '08 = 4K Dandelion
 '00 = Dolphin
 '32 = Dorado (as reported by Gwan)
 '32L = Dorado (as reported by Larry)
 '86-4 = 4K Daybreak
 '86-8 = 8K Daybreak
 X = Has microcode
 P = Prolog microcode set
 - = Doesn't have microcode
 ? = Don't know

Opcodes listed by entry in UFN table

op	'09	'08	'00	'32	'32L	'86-4	'86-8	Name
000	-	-	-	-	-	-	-	-X- *nobody implements:
causes an error!*								
001	X	X	X	X	X	X	X	CAR
002	X	X	X	X	X	X	X	CDR
003	X	X	X	X	X	X	X	LISTP
004	X	X	X	X	X	X	X	NTYPEX
005	X	X	X	X	X	X	X	TYPEP
006	X	X	X	X	X	X	X	DTEST {new COERCE}
007	-	-	-	-	-	-	-	CDDR {unused}
010	X	X	X	X	X	X	X	FNO
...	X	X	X	X	X	X	X	...
015	X	X	X	X	X	X	X	FNX
016	X	X	X	X	X	X	X	APPLYFN
017	-	-	-	-	X	-	-	CHECKAPPLY{?} [not 86-8?]
020	X	X	X	X	X	X	X	RETURN
021	X	X	X	X	X	X	X	BIND
022	X	X	X	X	X	X	X	UNBIND
023	X	X	X	X	X	X	X	DUNBIND
024	X	X	X	X	X	X	X	RPLPTR.N
025	X	X	X	X	X	X	X	GCREF
026	X	-	-	-	-	-	-	ASSOC [08 but not 86-8?]
027	X	X	X	X	X	X	X	GVAR←
030	X	-	X	X	X	X	X	RPLACA
031	X	-	X	X	X	X	X	RPLACD [which cases?]
032	X	X	X	X	X	X	X	CONS
033	-	-	-	-	-	-	-	{used for GETPROP}
034	X	-	-	-	-	-	-	FMEMB
035	-	-	-	-	-	-	-	{used for GETHASH}
036	-	-	-	-	-	-	-	{unused, named PUTHASH}
037	X	-	X	X	X	X	X	CREATECELL
040	X	-	X	X	X	X	X	BIN
041	-	-	-	-	-	-	-	BOUT {unused}
042	P	-	-	-	-	-	-	{Prolog}OPFETCHPLUSOPDISP
043	-	-	-	-	-	-	-	{unused, named LIST1}
044	-	-	-	-	-	-	-	DOCOLLECT {unused}
045	-	-	-	-	-	-	-	ENDCOLLECT {unused}
046	X	-	X	-	-	X	X	RPLCONS
047	X	-	-	-	-	-	-	LISTGET
050	-	-	-	-	-	-	-	ELT
051	-	-	-	-	-	-	-	NTHCHC
052	-	-	-	-	-	-	-	SETA

350	X	-	X	X	X	-	X	FPLUS2
351	X	-	X	X	X	-	X	FDIFFERENCE
352	X	-	X	X	X	-	X	FTIMES2
353	X	-	X	X	X	-	X	FQUOTIENT
354	X	-	-	-	-	-	-	UBFLOAT2 {UFADD, UFSUB, UFISUB, UFMULT, UFDIV, UGREAT, UMAX, UMIN, UREM}
355	X	-	-	-	X	-	-	UBFLOAT1 {UTOB, BTOU, UABS, UNEG, UFIX}
356	X	-	-	-	-	-	-	ARRAYREAD{GENERAL,UNBOXED}
357	X	-	-	-	-	-	-	ARRAYWRITE{GENERAL,UNBOXED}
360	X	X	X	X	X	X	X	EQ
361	X	X	X	X	X	X	X	IGREATERP
362	X	-	X	X	X	-	X	FGREATERP
363	X	X	X	X	X	X	X	GREATERP
364	X	X	?	?	X	-	X	EQUAL
365	X	-	X	X	X	-	X	MAKENUMBER
366	X	-	X	X	X	-	-	BOXIPLUS
367	X	-	X	X	X	-	-	BOXIDIFFERENCE
370	-	-	-	-	-	-	-	MISC5
371	X	-	-	-	-	-	-	FFTSTEP
372	X	-	-	-	-	-	-	MISC3
								{Floating Point Array ops: EXP, MAG, FLOAT, COMPLEX, BLKMAX, BLKMIN, BLKABSMAX, BLKABSMIN, FLOATTOBYTE}
373	X	-	-	-	-	-	-	MISC4
								{Floating Point Array ops: TIMES, PERM, PLUS, DIFFERENCE, MAGIC, BITMAPBIT}
374	-	-	?	-	X	-	-	{reserved for DOLPHIN}
375	X	X	X	X	X	X	X	SWAP
376	X	X	X	X	X	X	X	NOP
377	-	-	-	-	-	-	-	{unused}

notes:

4K microcode:

PLUS2, DIFFERENCE, TIMES2, QUOTIENT will ufn if args not INTEGERS

IPLUS2, IDIFFERENCE will accept FIXP's as arguments, but will ufn if result is not a smallp or smallneg

ITIMES2, IQUOTIENT, IREMAINDER will ufn if both args are not smallp

12K microcode:

PLUS2, DIFFERENCE, TIMES2, QUOTIENT will try floating point if args not INTEGERS

IPLUS2, IDIFFERENCE will accept FIXP's as arguments, and box the result if it is not a smallp or smallneg

ITIMES2, IQUOTIENT, IREMAINDER will ufn if both args are not smallp

—End of message—

SEdit Linearizer Internal Documentation

Fields in the context:

CurrentNode

The node whose linear form is currently being computed.

LinearPointer

Points to the "next" item in the linear form. This is the item with which the next generated item will be compared, and the item before which it will be inserted if it doesn't match.

LinearPrev

If LinearPrev is a cons, (CDR LinearPrev) is LinearPointer (LinearPrev is one behind LinearPointer in the linear form). Otherwise, it's a node, and (fetch LinearForm of LinearPrev) is LinearPointer. Used to fixup linear form.

CurrentLine

The LineStart most recently generated in the linear form.

CurrentX

The X coordinate at which the next linear item will be displayed.

RightMargin

The right margin for generating the linear form.

CurrentBlock

The LineBlock describing the most recently generated linear items on this line. Reset to FirstBlock at the end of each line displayed.

FirstBlock

The beginning of the LineBlock list. The LineBlocks from FirstBlock to CurrentBlock describe the segment of the linear form between CurrentLine and LinearPointer, indicating which parts are already available in the window for BITBLTing and which will have to be repainted.

Matching?

Means something like: the linear form we're generating has been matching the linear form that was already there (at least since the beginning of CurrentNode)

Below?

T if the linear form we're generating is definitely off the bottom of the screen. NIL if it might have to be displayed. 'new if we're redisplaying from scratch (nothing to BLT).

Visible?

T if we're matching and the bits we're matching are actually on the screen.

RelinearizationTimeStamp

NIL if we're prettyprinting, otherwise incremented by 1 each time we relinearize from the top. Used to determine the validity of cached info in LineStarts

RepaintStart

RepaintLine

RepaintX

If there are no bits to be reused at the end of a line, we postpone displaying it until we find something that needs to be moved (or we get to the end of the window). This can go on for many lines. during this time, RepaintStart records where the painting needs to start from, RepaintLine has the y information, and RepaintX records where the painting will start from. It may always be the case that

RepaintLine = (CAR RepaintStart)

RepaintX = (fetch Indent of RepaintLine)

Fields in EditNodes:

StartX

The X coordinate at the time this node was linearized. not sure what 0 means (something magic). \\reuse.linear.form seems to think this means that the node is atomic and hasn't been displayed before.

The coordinate system:

The top left corner is (1,-1) (or maybe (0,0)?). Therefore everything's displayed with positive x and negative y (bottom right quadrant).

List Formats

SEdit allows one to specify how a class of forms are to be pretty printed. This is done by defining a list format on a symbol. This causes all forms whose car is this symbol to be displayed with the specified format.

List formats for most Common LISP and Interlisp special forms are provided with SEdit. The source code for these definitions can be found on the Lisp Library Floppy #XXX.

```
(def-list-format name {doc-string} {format-name | &key :args :sublists :inline :miser :last :indent}) [Definer]
```

Tells SEdit how to prettyprint forms whose CAR is NAME.

Short form

If *FORMAT-NAME* is provided then NAME is defined to be formatted just like *FORMAT-NAME*. If, for example one had defined a list format for *dotimes* one could then define *dolist* to be formatted the same way with:

```
(def-list-format dotimes dolist)
```

Long Form

The keyword arguments have the following meanings:

:ARGS -- value should be a list of names of list formats. These formats are assigned to the elements of the list in order starting with the first element (which will be NAME). Note that these formats override any formats that would normally be assigned to the elements of the list (based on their first elements). NIL is allowed in the :ARGS list, and means do *not* override the format of this element; that is, allow it to be formatted normally. Also, a symbol S is allowed in the :ARGS list if S has earlier been assigned a format; this means to assign S's format to this element. There are also two special keywords allowed as entries in the :ARGS list: :KEYWORD and :RECURSIVE. :KEYWORD means that if the element assigned this format is a symbol then treat it like a keyword, i.e., put it in bold face. (This list uses the convention that all symbols which allow declarations in their body [such as DO and LET] are formatted as keywords.) :RECURSIVE means to assign this element the same format as is being defined; that is, the entire top level format is assigned recursively to this element. This is very useful for formats like :DATA format (see below). If L has more elements than there are entries in the :ARGS list, the last entry in the :ARGS list is repeated for all the extra elements of L. Hint: most :ARGS entries have NIL as their last element. If no :ARGS list is specified, the elements of L get their natural formatting.

:SUBLISTS -- value should be a list of element positions (counting from 1) or T. T means all of the arguments should be parsed as lists even if they are NIL (so NIL will display as () rather than NIL). A list of element position means those element positions will be parsed as lists. For example LET has :SUBLISTS (2) meaning the second element of a form whose first element is LET is a list (i.e., the binding list). DO has :SUBLISTS (2 3), DEFUN has :SUBLISTS (3) and COND has :SUBLISTS T. Default is :SUBLISTS NIL meaning print all NIL args as NIL not ().

:INLINE -- value can be T or NIL (default NIL). If T, the form will go all on one line if it fits. If NIL, the form will be broken across lines at arg boundaries even if it would all fit on one line. For example, OR has :INLINE T and LET has :INLINE NIL.

:MISER -- value can be :ALWAYS, :NEVER, or :TOFIT (default :TOFIT). Specifies when to use miser indentation. The default means use miser indentation if non-miser indentation would force the arguments into miser indentation. **[need to explain what miser mode is]**

:LAST -- value should be a format specification like those in the :ARGS list. This format specification will be applied to the last element of L but *only* if doing so would supercede the last entry in the :ARGS list. In other words, if the last element of L would receive the repeated format from the :ARGS list, it gets the :LAST format instead. This option is really only useful for pathologically formatted forms like Interlisp's SELECTQ.

:indent -- An indentation specification is either a symbol (normally a keyword) or a list. If it's a symbol, it's looked up on the SEDIT::*INDENT-ALIST* (which see) and the SEdit-internal indent specification found

there is used. If it's a list, it consists of some optional keywords (described below) followed by argument group specifications. Each argument group specification is either a number or a list containing a single number. In both formats, the number indicates that that many arguments should be grouped together at a single indentation level. The simple number format means that each of those arguments should go on its own line (they will line up vertically with each other), while the number-in-a-list format means that the arguments in the group can go together on a single line if they fit. The indentation level for each argument group is determined by how many groups follow it in the indentation list. Each group is indented 1 level further in than the group which follows it; thus, the first argument group is indented most, the next one next most, and so on until the last one, which is always indented one step in from lambda-body level.

This is best explained with examples. A simple example is LET, whose indentation specification is (1). This means that LET will be followed by a single distinguished argument group consisting of one element (the binding list) which will be indented one step in from the let body. Another simple example is DO, whose indentation specification is (2). This means that DO will be followed by a single distinguished argument group consisting of two elements (the binding list and the termination clause) which will be indented one step in from the do body. It also means that the bindings and the termination will be required to go on separate lines. Contrast DO with DEFUN, whose indentation specification is ((2)). Like DO's spec, DEFUN's spec says there is one group with two members (the name and the lambda-list), but unlike DO's spec, DEFUN's spec says that the first two args can go on the same line if they fit there. Finally, consider a possible spec for MULTIPLE-VALUE-BIND of (1 1) which says that the first group consists of one arg (the variable list) and the second group consists of one arg (the form to eval). The form to eval will be indented one step in from the body, and the list of variables will be indented one step in from there.

Note that a group specification of 0 (zero) is allowed: this occupies an indentation step but does not put any arguments at that level. But we do not allow (0) as a group specification since this would not be any different than plain 0 and probably means that the specification is confused in some way.

The keywords allowed at the beginning of an indent specification are:

:BREAK or **:NOBREAK** or **:FIT** -- These specify placement of the first argument in the first group. Default is **:FIT**, which means put this arg on the same line as the CAR of the form if it fits there in preferred mode, otherwise put it on the next line. Note that if the first arg goes on the same line as the CAR, its placement specifies the indentation level for the entire first group. That way long CARs will move the first group over to the right. (This makes the binding and termination of both DO and DO* line up, for example.) Specifying **:NOBREAK** means the first arg is forced to go on the same line as the CAR. Specifying **:BREAK** means the first arg is forced to go on the next line (and thus at the indentation level derived from the number of groups). UNWIND-PROTECT is a good example of using **:BREAK** to force the first arg onto its own line. Note that you can only specify one of **:NOBREAK**, **:BREAK**, or **:FIT**.

:TAGBODY -- Normally all forms in the body (whether atomic or not) go at the same indent level. Specifying **:TAGBODY** indicates that atomic body elements (*not* atomic elements of the argument groups) should be extended to line up with the CAR of the entire list (such as PROG or TAGBODY, which see for examples).

:STEP -- This can be specified as many times as desired and each time increases the indentation of the body (and thus all the argument groups) by one step. If you just want to move some of the groups in but not all of them (and not the body) then use 0 groups at the appropriate place instead of using **:STEP** at the beginning. **:STEP** is very useful with **:TAGBODY**.

By the way, the normal body indentation is taken from the INDENT-BASE field of the LISP edit environment, which is initialized to the width of a capital 'M' in the SEdit default font. The normal indentation step is taken from the INDENT-STEP field of the LISP edit environment, which is initialized to twice the width of a capital 'M' (that is, twice INDENT-BASE). These defaults are chosen so that, in a fixed-width font, the body of a form lines up two characters in from the '(' of the form, and each argument group line up two characters in from the next one (or the body). If you want non-standard values for either of these parameters, you can change the values in the LISP edit environment and then reinitialize

your SEdit formats. Also, if you change font profiles, reinitializing SEdit will fix up the indents appropriately.

SEdit Internal Documentation

The formatting methods for lists

The `assign-format`, `compute-format-values`, and `linearize` methods of lists are now driven by tables encapsulated in list-format objects, allowing easy special formatting for particular lisp forms. This note primarily documents the format of list-format objects, and along the way mentions how they are used to implement those three methods. (note that dot-lists currently aren't handled by this mechanism)

Finding the right list-format

`assign-format-list` finds an appropriate list-format to control the list's formatting, using one of four options:

- if the list is assigned a format which is a list-format object, use that;
- else if the car of the list can be found in a list-formats table,
 - use the associated object from that table;
- else if the car of the list has a known clispword property,
 - use the clisp list-format
- else use the default list-format

since the same list-format object will be needed for computing width estimates and and linearization, `assign-format` caches it in the unassigned file. there are two types of list-formats; standard and non-standard. standard list-formats contain information to control the standard list formatting methods. nonstandard list-formats are an escape mechanism for situations where the required formatting is too hairy for the standard methods; they simply provide replacements for the `assign-format`, `compute-format-value`, and `linearize` methods. this is implemented by all list-formats having a non-standard? field, and list-formats whose non-standard? field is t having 3 additional fields: `set-format-list`, `cfv-list`, and `linearize-list`. there's not much else to say about non-standard list-formats, except that at present the only one is the format for clisp. for the rest of this document we'll talk about standard list-formats.

A general rule

several list-format fields contain lists of entries which correspond to the list node's subnodes. these lists all have roughly the same form:

(*last first second ... nth*)

where *first* is the information to be used for the formatting the first subnode, *second* the information for the second, ..., *nth* the information for the nth and all subsequent nodes, except that *last* is the information to be used for the last (some forms (e.g. `il:selectq`) have special formatting for the very last item). *n* depends on how many of the nodes need special formatting. `lambda`, for instance, uses lists of the form (*a b c a*) — thus the first element is formatted using *b*, the second with *c*, and all subsequent ones with *a*. the default list format uses lists of the format (*a*), since all subnodes are formatted similarly. `il:selectq` uses lists of the form (*a b c d*) — so *b* applies to the `il:selectq` atom, *c* to the evaluated expression, *d* to each clause, and *a* to the final otherwise clause.

the other important fact about these lists is that they're blind to comments; comment subnodes are ignored when figuring out which information goes to which subnode, and the comments themselves are formatted by hardwired rules.

Assigning formats

`assign-format-list` uses the `list-formats` field of the list-object, which is a list in the format described above. each element of the list is the format to assign to the corresponding subnode (`nil`, `:keyword`, or a list-format object — or `:recursive`, which means to assign this node's list-format to its subnode).

Linearizing

each list-format contains descriptions of two possible presentations of that list — a “preferred” format and a “miser” format, in the list-pformat and list-mformat fields of the list-format. linearization decides which presentation is appropriate, based on the width estimates of this node (see below for a description of where they come from) and the horizontal space available.

the list linearization will always have the form `(" subnode <space> subnode <space> ... subnode ")`, where `<space>` is either a one-space-wide horizontal movement, or a line break (with some indentation) (we assume the list contains no comments for now). thus, the problem of formatting the list reduces to specifying, for each element after the first, whether to space or break, and if you break, how much to indent. using a list in the above format, a specification is given for each subnode after the first as to how this decision is to be made.

these spacing specifications are expressions in a simple language. each expression must at a minimum specify the indentation if a line break is made here. the simplest expressions are just that — an integer, giving the indentation (in pixels) from the opening `"`. this expression won't break unless it has to, but if it does it will use that indentation. to force a break, an expression of the form `(break . exp)` is used (where `exp` is a nested expression). to line up the presentation of subnodes, spacing specifications can set a tab stop and then later position relative to it. `(set-indent . exp)` works just like `exp`, except that after it has been determined where this subnode is to be positioned, the tab stop is set at that point. a later spacing specification of `(from-indent . exp)` means that if we break at this point, the indentation is to be taken relative to the tab stop. note that the order of parts in an expression isn't important; `(break from-indent . 3)` has the same effect as `(from-indent break . 3)`.

the remaining types of expressions allow the formatting to depend on a variety of conditions, such as whether the previous node's presentation fit on one line, or whether this node is atomic. they all have the form `(condition exp1 . exp2)`, where `condition` is one of the atoms below, `exp1` is the spacing specification to be used if the condition holds, and `exp2` is the spacing specification to be used if it doesn't. (remember that a spacing specification is interpreted to determine the space preceding a subnode; in the list below, "previous node" is the node before the one whose placement is being determined, and "next node" is the node whose placement is being determined).

prev-inline:	did the previous subnode's presentation fit on one line?
next-inline:	does the next subnode's width estimate indicate that it will fit on this line?
next-preferred:	does the next subnode's width estimate indicate that it will fit in preferred format?
prev-atom, next-atom:	is the previous / next node atomic (i.e. has no subnodes)?
prev-keyword, next-keyword:	is the previous / next node an atom in the keyword package?
prev-lambdaword, next-lambdaword:	is the previous / next node a lambda keyword (<code>&aux</code> , <code>&rest</code> , etc)?
whole-inline:	do the width estimates of the node being formatted indicate that it will fit on one line?

Formatting comments

the preceding discussion assumed that the list being formatted contained no comments. the formatting of comments is completely automatic (i.e. out of the control of the list-format), primarily to simplify list-formats. they are formatted like other subnodes, except that they don't use the spacing specifications. single-semi comments are positioned at a fixed horizontal position (at the end of the current line if it isn't too long, otherwise on a new line). triple-semi comments always start on a new line, at the left margin. double-semi comments are a little trickier; they start at the current tab stop, after the spacing specification for the next node has been interpreted. this requires looking ahead in these cases, and determining what the result of interpreting the space specification will be. it would probably be worth figuring out simpler ways to do this.

subnodes following comments always start on a new line (i.e. their spacing specifications are interpreted as if they began (break)).

Computing width estimates

cfv-list must compute two values: the width of this node if its presentation can fit on one line (or nil if it can't), and the width of this node in its "preferred" presentation. the second of these is computed by simulating the linearization process using the preferred spacing specifications, and always assuming the worst when evaluating the prev-inline, next-inline, next-preferred, and whole-inline conditions. the inline width is determined by an even simpler scheme: if all of the subnodes can go inline, and the list-inline? field of the list-format isn't nil, the list can go inline with width equal to the sum of its subnode's inline widths, plus the appropriate blanks and parens.

Node Types and Node Type Methods in SEdit

All SEdit editing operations and presentations are controlled by methods associated with classes of nodes in the edit tree. By defining a new class of nodes with appropriate methods, or modifying an existing class, SEdit may be configured for a wide range of editing tasks. This document describes the programming involved in defining such a class, using the basic Interlisp-D type definitions as examples.

Node Types and the Edit Tree

The edit tree is a data structure maintained by SEdit as a representation of the structure being edited. The tree is initially constructed by parsing the data structure to be edited as a hierarchical data structure. Each node in the tree corresponds to a part of the edited structure, either an instance of a lisp datatype or a combination of several data structures. For instance, atoms and strings are usually represented by separate nodes in the tree, but a sequence of several cons cells may be represented by a single list node. Instances of a single data type may be represented by a variety of node types, depending on the context in which they appear. All editing operations are defined in terms of these nodes. The definitions for editing Interlisp code define ten node types:

1. atoms (actually litatoms and numbers)
2. strings
3. lists (NIL-terminated sequences of cons cells)
4. dotted lists (sequences of cons cells terminated by something other than NIL)
5. CLisp expressions (if, fetch, iteration, etc.)
6. forms (lisp function calls)
7. LAMBDA expressions (also includes LETs, PROGS, etc.)
8. quoted structures (i.e. two element lists whose first element is the atom QUOTE)
9. unknown (any data type other than litatom, number, string, or cons)
10. root (a special node type for the root of the edit tree)

Node types 5 through 8 are special cases of type 3, which are recognized in some contexts.

Creating the Edit Tree

To create the edit tree, SEdit conducts a preorder traversal of the given data structure. At each step, it dispatches on the datatype of the structure and calls the corresponding function, which is responsible for building the rest of the tree by making a call to `\create.node` and recursive calls to the parser (`\parse`). Each node built will record (among other things)

- the node type (chosen by the parse function)
- the node's super node
- the node's subnodes (one for each recursive call to `\parse`)
- the node's depth in the tree
- the structure which was parsed to create this node

Because the correct parsing of a node often depends on contextual information, the parser allows each parse method to pass an argument to its subnodes. For no particularly good reason this argument is called the *parse mode*. The interpretation of the parse mode is up to the individual parse methods. The default mode is NIL; most parse methods ignore any unrecognized mode. In addition to the default mode, the Interlisp code definitions use a couple of other modes (the atoms `Binding`, `BindingList`, and `KeyWord`).

Edit Node Types

The type information SEdit associates with each node is actually a set of methods, i.e. functions to perform various actions on nodes of that class. Each node type must provide methods to perform seventeen different actions. This is, of course, poor man's object oriented programming. One day in the grand and glorious future Interlisp-D will metamorphose into something which properly supports object oriented programming, whereupon SEdit will be rewritten. In the meantime, this works well enough.

The methods for node types fall into four groups:

- those used to generate the presentation of a node (2 methods)
- those used to place selections and points (6 methods)
- those used to effect editing commands (5 methods)
- those which perform various housekeeping functions (4 methods)

We'll consider each of these groups in turn, specifying in detail how the methods are invoked and what they are expected to do.

Presentations and the Linear Form

The visual presentation of the data structures is represented in SEdit by a structure called the linear form. The linear form is a sequence of presentation commands which produce the desired presentation. Presentation commands are very simple, and there are only four of them:

- insert a string of characters in a given font
- insert horizontal space
- insert a given bitmap
- start a new line, with a given indentation and vertical separation from the previous line

The `Linearize` method for a node type must construct the sequence of presentation commands for a node of that type, by inserting the appropriate commands in the correct order. It may call the `Linearize` method of each of its subnodes. For instance, a very simple algorithm for linearizing lists might be:

```
output "(" in the default font
for each subnode
    if this isn't the first subnode, output some space
    linearize the subnode
output ")" in the default font
```

The actual algorithm used by SEdit is somewhat more complicated; for instance, lists usually don't fit all on one line, so it inserts line breaks at appropriate points.

To format structures such as lists properly, it's important to know how much space the presentation of each subnode will occupy. This is a problem, since the amount of space the presentation of a structure occupies often depends on the amount of space available when it is presented. SEdit deals with this by computing *width estimates* for each node. These are five values computed by each node for use by its super node:

- a) if the node can be presented on a single line, the width of that presentation
- b) the width of the most readable presentation of this node (its preferred presentation)
- c) the width of the narrowest possible presentation of this node
- d) the length of the last line of the preferred presentation of this node
- e) the length of the last line of the narrowest possible presentation of this node

These width values allow the super node to make reasonable decisions on indentation and line breaks. The last line lengths are important because the super node may append additional material to the last line of a subnode's presentation, and would like to know how long the resulting line would be. These five values are stored as extra fields in each node, and referred to as the `InlineWidth` (NIL if the node can't be presented inline), `PreferredWidth`, `MinWidth`, `PreferredLLength`, and `MinLLength`, respectively.

In much of the program these width estimates are called *format values*. I've started calling them width estimates as a reminder that they are not required to be accurate, but should be quick to compute. It is important that a node's width estimates not indicate that it can be presented inline when it actually requires several lines, but other width values can be incorrect without breaking the program. On the other hand, incorrect width estimates will often lead to less than optimal presentations. The width estimates computed by the Interlisp code definitions are always correct (at least, that was the author's intention).

Each node type defines a method called `ComputeFormatValues` which will fill in the width estimates of a given node. To do so it may examine the structure the node represents and the format values of its subnodes. For instance, the `ComputeFormatValues` method for type list looks approximately like this:

```

InlineWidth is
  if all of the subnodes can be presented inline
    the sum of the InlineWidths of the subnodes,
    plus the width of two parentheses,
    plus the width of n-1 blanks
  else
    NIL
PreferredLLength is
  the PreferredLLength of the last subnode,
  plus the preferred indentation,
  plus the width of one parenthesis
MinLLength is
  the MinLLength of the last subnode,
  plus the width of two parentheses
PreferredWidth is
  the largest of
    the PreferredWidth of the first subnode,
    plus the width of one parenthesis
    the maximum PreferredWidth of the other subnodes,
    plus the preferred indentation
    the PreferredLLength of this node
MinWidth is
  the largest of
    the maximum MinWidth of any subnode,
    plus the width of one parenthesis
    the MinLLength of this node

```

Note that these calculations assume the node has at least two subnodes — special case rules are needed for fewer. Also, the formatting rules assumed are that

- the inline presentation of a list is an left parenthesis, followed by the subnodes separated by blanks, followed by a right parenthesis; all of the subnodes must present inline
- the preferred presentation of a list is an left parenthesis, followed by the subnodes, where each subnode after the first is on a new line indented by the preferred indentation; the final subnodes is followed by a right parenthesis
- the minimum presentation is similar to the preferred presentation, except that subnodes are indented only by the width of the left parenthesis

(Indentations are always non-negative, and specified relative to the horizontal position of the start of the node's presentation; thus the presentation of a node always occupies the quadrant to the right and below the point at which its presentation starts.)

The linearization method is given two arguments in addition to the node to be linearized and the usual context information. The first is a right margin, which it should try to keep its presentation within (if possible). This value is actually stored as another field in the node. The second argument is an index. In some situations, SEdit may request that the linear form of a node be recomputed starting part way through. In this case, the index given will be the index of a subnode, and the method should output just that part of the node's linear form which follows the presentation of that subnode.

Linearization methods are permitted to call five lisp functions to create the node's linear form:

- (\\output.string context string prin2? font)
Insert string in the specified font at this point in the linear form. context is the usual context encapsulation. string may be any Interlisp object; its standard printed representation will be used. if prin2? is true, the PRIN2 representation of string will be used instead.
- (\\output.space context width)
Insert a horizontal space of the specified width at this point in the linear form.
- (\\output.bitmap context bitmap)
Insert a bitmap at this point in the linear form. It will be aligned with the line's baseline.
- (\\output.cr context indentation lineskip)
Start a new line, with the specified indentation (relative to the start of this node's presentation) and separation from the previous line.
- (\\linearize subnode context right.margin)
Insert the linear form of a subnode at this point in the linear form. Its linear form should not extend beyond right.margin (if right.margin is not specified, it will default to this node's right margin).

Pointing and Selecting

When the user uses the mouse to place the caret point and/or select part of the edited structure, SEdit is faced with the task of mapping the mouse's (x,y) coordinates to the appropriate structure description. These descriptions take the form of datatypes called EditPoints and EditSelections. An EditPoint records

- the node which owns this point (i.e. the one which will be informed if something is inserted here)
- an index, which the owning node may use to record arbitrary information about the point's location
- a type, one of Structure, Atom, or String, indicating how characters typed here will be interpreted
- a line (in the linear form) and x offset within that line, indicating where the caret should be positioned in the window if this point is displayed

EditSelections are similar, except that they have two indices (since selections may cover a sequence of substructures), and describe two positions (bounding the part of the linear form which should be underlined to display this selection).

The responsibility for translating mouse positions to points and selections is shared between the kernel and the type methods. The kernel determines which part of the linear form is being pointed at, and then asks the node which produced that part of the linear form to determine the point or selection. Sometimes the node will decide that in fact its super node or one of its subnodes should really be responsible, and if so it may pass on the request to them. For instance, if the user tries to insert characters at the beginning of an atom, they may actually point to the space between that atom and a preceding structure. The node which output that space was the atom's super node, i.e. the enclosing list. When it receives a request to position a character point in the space between two

subnodes, it should realize that this was likely an attempt to actually edit one of the subnodes, and pass the opportunity to them.

The first method in this group is `SetPoint`. It is called with (among other things) the context (all methods are passed the edit context; from now on we'll stop mentioning it), the node, the index of the linear form item in which the mouse is positioned, the x offset within that item, and the type of point requested (i.e. the choice of mouse button used to place the point). It must fill in the fields of the point, or call some other node's `SetPoint` method to do it. It can call `\set.point.nowhere` to indicate that no point is near this mouse position, and thus the point returned should not allow input.

There are two other calling sequences a `SetPoint` method may be invoked with, to allow `SetPoint` requests to be readily passed between nodes. A subnode may pass a `SetPoint` request to its super node, indicating that the point is to be placed either immediately before or immediately after itself. For instance, this is what atomic structures do when asked to insert structure. To do this, the subnode calls `\punt.set.point`, passing a flag to indicate whether the point should be before or after this node. Conversely, a node may pass the `SetPoint` request to one of its subnodes, indicating that the point is to be placed at the beginning or end of that subnode (as in the example with left-clicking the space in a list, above). The `SetPoint` method can determine which case it is being asked to handle by examining its arguments.

`SetPoint` methods usually calculate the position of the point when they set it. If the linear form changes, or the point is set by some other means, it becomes necessary to recompute the position of the point so that the caret may be displayed. The `ComputePointPosition` method of the node owning the point is responsible for filling in the line and x offset values on request.

Similarly, the `SetSelection` method and `ComputeSelectionPosition` methods determine the current selection, given the mouse position. These are very similar to the corresponding methods for points, and may call `\set.selection.nowhere` or `\punt.set.selection`. `\set.selection.me` is a useful default `SetSelection` method; it simply sets the current selection to this node.

There are two additional methods related to selections. `GrowSelection` is called when the user uses multiple mouse clicks to select structures. If the mouse handler detects a multi-click sequence, it calls the `SetSelection` method for the first click, and `GrowSelection` for each subsequent click. The `GrowSelection` method of the owner of the current selection is responsible for enlarging the selection to include the next enclosing level of structure. `\grow.selection.default` is the `GrowSelection` method for most types; it simply calls `\punt.set.selection`, causing the super node to become the new selection owner.

The `SelectSegment` method handles right-button mouse actions, which extend the current selection to include the item pointed to. If the mouse is pointing at part of the linear form of the owner of the current selection, the `SelectSegment` method of that node will be called to fix up the selection. Otherwise, the deepest common super node of the node selected and the node pointed to will be asked to determine the new selection, given which of its subnodes the selection and mouse are in.

Editing Operations

When an editing operation (such as the deletion or insertion of material) is performed, an appropriate method is called for the affected node, and this method is expected to fix up both the tree and the actual structure being edited. These operations are all defined in terms of points and selections, and the method invoked is that of the owner of the point or selection.

The `Insert` method takes a previously created point owned by this node and either a string of characters or a list of nodes which are to become subnodes. The appropriate changes are to be made to the tree and structure, and the point should either be adjusted to be after the inserted material or cancelled, as appropriate. Some appropriate functions to call are:

`(\\note.change node context)`

This node has changed in some way which affects its presentation. Its width estimates should be recomputed, and the linear form update appropriately.

(\\subnode.changed node context)

If the structure resulting from an editing operation is no longer EQ to the original, \\subnode.changed will inform the node's super node that it must make the appropriate updates. For instance, if a character is inserted into a litatom it actually becomes a new litatom. No change actually takes place in either atom, but the structure which contained the original atom must change to refer to the new one.

The Delete method takes a previously created selection and deletes the selected material. It returns a flag indicating whether in fact the material can be deleted; many data structures do not allow material to be deleted. It may also be asked to set the caret point so that inserted material will replace that which has just been deleted.

The Replace method replaces the selected material with either a string of characters or a list of nodes (depending on the type of selection). \\replace.default is a default implementation of this method, which does a deletion followed by an insertion. This is satisfactory for many data structures, but fails for those which do not allow deletions (e.g. quoted structures).

The Split method is only required for those types which allow character points (i.e. atoms and strings). When a delimiter (e.g. blank or cr for atoms, double quote for strings) is inserted at such a point, the node's Split method will be called. The usual behavior is to change the caret point to a structure point, and to separate the structure into two if the point was not at the beginning or end of the structure.

The BackSpace method implements the action of the backspace key. Given the caret point, the method is expected to make the appropriate deletion and adjust the point accordingly. Sometimes only the latter is necessary; for instance, if the point is positioned immediately after a list, backspace merely moves the point so that it is after the last element of the list (this corresponds to the action of the right parenthesis moving the point to after the parenthesis).

Miscellaneous Methods

There are four other methods for node types to implement, which are invoked by SEdit when the effects of some editing command might be important to a node other than that directly affected. The first of these, SubNodeChanged, was mentioned above; when the structure associated with a subnode is replaced with one which is not EQ to the original one, the super node's SubNodeChanged method will be invoked so that it can fix up its structure appropriately.

Two other methods are used to implement copying and moving structures. When a move or copy selection is made, the CopySelection method of the node owning the selection will be invoked, and passed the selection, a flag indicating what type of selection was made, and a description of the destination. The destination will be either the context of this or another SEdit process, or NIL, indicating that the copy or move is being made to a non-SEdit process, so the material should just be BKSYBUFed. The default method \\copy.selection.default provides an implementation of this method suitable for most applications. If this is used, the CopyStructure method must be defined. CopyStructure is called with a node which has been constructed as a copy of an existing node. Because the copy operation should create new structure rather than just creating another pointer to the same structure, the CopyStructure method must fill in the Structure field of the node with the appropriate newly created structure. The Structure fields of its subnodes will already be copies of their structures, so all that is usually required is to create a new data structure out of these.

When material is moved or copied to other parts of the structure, the position at which it is inserted may imply a different parsing of the structure than that from which it is taken, since the parsing is context dependent (remember the parse mode?). In such a case, the new super node may ask the node to reparse itself, using the Reparse method. This usually involves minor adjustments to presentation, although in the worst case it may involve completely parsing the structure again from scratch.

Editing Interlisp Code with SEdit

The Interlisp editing definitions configure SEdit as an editor for programs written in Interlisp-D, and are ultimately intended as a replacement for DEdit, the system display editor. Although the current system is still missing many convenience features, it currently provides a workable alternative to DEdit. This document provides detailed information on using SEdit as a code editor. It is assumed that the reader has read the introduction to SEdit, and is familiar with the Interlisp-D programming environment. This part of SEdit is under active development; this document will be changed as improvements are made.

Running SEdit

After loading SEdit, the function `SEdit` allows it to be installed and de-installed as the default system display editor. Executing `(EDITMODE 'SEdit)` will cause future edit requests (from functions such as `DF`, from Masterscope, and from inspectors and browsers) to use SEdit instead of DEdit; executing `(EDITMODE 'DEdit)` will revert to using DEdit. The function will return the previous editor state (`SEdit` or `DEdit`).

Unlike DEdit, SEdit does not run in the process which invokes it. This has some important effects:

- a) SEdit processes can be started and stopped in any order. The windows may be shrunk and kept around indefinitely if desired.
- b) Calls to editing functions such as `DF` return as soon as the process is started, rather than waiting for the editing to be completed (however, when SEdit is invoked from Masterscope, it forces Masterscope to wait until the user indicates that they are done editing (by closing or shrinking the window); this is simply a convenience to avoid Edit where any commands from immediately covering the screen in hundreds of edit windows).
- c) As a consequence of (b), some functions normally performed by `DF` (such as informing the file package that the function has been changed and needs to be saved, unsaving the definition of a compiled function, or updating the "last edited" date) are instead performed by SEdit, and often at different times (since SEdit can't wait until the user is "done editing").

Commands

At present, SEdit has no attached menu of commands. Many of the commands in DEdit's menu (such as `Before`, `After` and `Replace`) are completely unnecessary in SEdit (because of its more uniform interface). Some of them (such as `Delete`, and soon `()in` and `()out`) are provided by keyboard commands. A menu may be added in the future with the introduction of more obscure commands.

Pointing and Selecting

Like TEdit, SEdit maintains a current insertion point at which typed, copied, or moved material will be inserted. The point is set by moving the mouse to the desired position and clicking a mouse button, and is indicated by a flashing caret. Unlike TEdit, SEdit has two types of points: structure points and character points. Structure points are allowed within non-atomic structures which have a variable number of components (i.e. lists); they indicate that another Lisp structure can be inserted at the indicated position. In normal (NIL-terminated) lists, structure points can be placed before the first element, after the last element, or between any two adjacent elements; in a non-NIL terminated (dotted) list, points may not be placed anywhere after the dot. Character points are positions at which individual characters may be inserted (rather than whole Lisp structures), and are allowed in atoms and strings. The caret changes to reflect the type of point: structure points look like "▲" and character points look like "▲"

Similarly, both structures and individual characters can be selected. A selection may be

- one of the characters in the pname of an atom or string

- a sequence of consecutive characters in the pname of an atom or string
- a lisp structure presented as an entity (e.g. an entire list, string, atom, quoted structure, etc.)
- a sequence of such structures appearing consecutively in a list

Note that not all lisp structures are presented as distinct entities, and so not all will be selectable. For instance, the individual cons cells comprising a list are usually not separately selectable. Also, extra characters added to the presentation as punctuation are not individually selectable; you can't select the left parenthesis of a list, or the closing quotation mark of a string.

The type of selection and point made depends on the mouse button used. The left button selects characters and places character points; the middle button selects structures and places structure points (this is supposed to be reminiscent of TEdit). The right button extends the current selection to the smallest selection which covers the current mouse position. As an added convenience, clicking with the left or middle button more than once in the same spot will enlarge the current selection one step, through enclosing layers of structure.

Inserting and Replacing

To insert characters in an atom or string, use a mouse button to place a character point at the desired location and just type the characters. As each character is typed, it will be inserted and the caret point will be moved after it. The Backspace key deletes the character to the left of the caret.

To insert new structures in lists, place a structure point at the desired location and type one of

- a left parenthesis to insert a new list
- a double quotation mark (") to insert a new string
- a normal character (i.e. one with syntax class OTHER) to insert an atom beginning with that character

In the first case, an empty list will be inserted and the caret will be moved inside it. In the second, an empty string will be inserted and the caret will become a character point inside the string. In the third case, a new atom will be inserted, and the caret will become a character point to allow appending more characters to the atom's name.

There are a few other characters which are recognized specially:

- a right parenthesis places the caret point immediately after the list immediately enclosing it
- a double quotation mark, while inserting characters in a string, places the caret point immediately after the string (if it was after the last character in the string) or splits the string into two strings (if it was between two characters)
- a blank or carriage return, while inserting characters in an atom, places the caret point immediately after the atom (if it was after the last character) or splits the atom into two atoms (if it was between two characters)

These characters all leave the caret point ready to read another structure. The rules may sound a little bizarre, but they work out to give just the right behavior — typing in the printed representation of a lisp structure will give you that structure. (At present, this only works for (undotted) lists, string, litatoms, and numbers; soon dotted lists and quoted structures will also be implemented).

Special characters, such as parentheses, spaces, and double quotation marks, can be inserted in atoms by preceding them with the escape character (a percent sign).

To replace structure, it is selected "pending delete". Pending deletion selections are made whenever the current selection is extended using the right mouse button (as in TEdit). To distinguish them from normal selections they are displayed by outlining the selected material, rather than underlining it. When structures or characters have been selected pending delete, typing anything will cause them to be replaced with the new material.

Copying, Moving, and Deleting

Structures or characters may be copied or moved from one part of an SEdited structure to another, between two SEdited structures, or between an SEdited structure and any other Interlisp process which will accept or produce character string representations of lisp structures. To copy material, place a point of the appropriate type at the desired destination, and then select the desired material while depressing the Copy key (Shift on the Dorado keyboard). Selections made with the Copy key depressed will be displayed by a gray underline. As soon as the Copy key is released, a copy of the current selection is inserted at the point. If the TTY process is a non-SEdit process, a printed representation of the selected material will be BKSYBUEF for it to read.

Moving material is done in a similar fashion, except that the Move key is depressed while making the selection. Move selections are displayed by a gray outline. As soon as the Move key is released, the selected material will be inserted at the current caret point and deleted from its original position. On the Dorado keyboard, Move selections are indicated by depressing both the Shift and Control keys.

Material may also be deleted in this fashion. Depressing the Control key while making a selection will cause the selected material to be deleted as soon as the Control key is released. Delete selections are displayed by inverting the selected material (displaying it white-on-black instead of black-on-white).

Whenever selections are being made, the selection is considered complete only when the mouse buttons *and* any modifier keys (Copy, Move, etc.) have been released. Thus, selections requiring more than one mouse click (e.g. sequence selections) can be made by keeping the modifier key depressed throughout the process. Alternatively, if the wrong modifier key is initially depressed, it can be released and another depressed as long as a mouse button is held down during this time. To completely abort the selection, click a mouse button outside the window before releasing the modifier key.

There are two other ways of deleting material. The Backspace key, as was previously mentioned, deletes the character to the left of the caret (this strictly true only when the caret is a character point in an atom or string; at other times it does other, reasonable things — you'll have to try it out to find out exactly what). The Delete key deletes the current selection. (Note that this is different from the Control key, which is a selection modifier; with Delete a normal selection is made and then the Delete key is depressed, while the Control key is depressed while the selection is actually being made — the choice of which is to use is a matter of personal taste.)

Formatting

SEdit attempts to display the structure being edited in as readable a fashion as possible, while keeping within the width of the display window. It uses fairly conventional rules for pretty-printing lisp, augmented with some special formatting rules for Interlisp special forms (e.g. LAMBDA expressions and CLisp). The components of these special forms are given indentation based on their function within the form, and special keywords are displayed in bold face to improve readability. Some effort is made to propagate the width constraint information so that relatively uniform indentation is used, rather than having complex nested expressions end up mashed against the right edge of the window. In extreme cases SEdit will extend the presentation past the right edge of the window rather than produce too ugly a presentation. If this happens, a horizontal scroll bar will be added to the window to allow editing the whole presentation.

SEdit Windows

The windows SEdit creates behave like all good Interlisp windows. They may be moved, reshaped, and scrolled. If they are reshaped to a different width, the formatting is recomputed to make the best use of the available space. They may be shrunk, producing a relatively unexciting icon adorned with the name of the variable or function being edited. When they are shrunk, the process reading commands from the keyboard is deleted (to save stack space, and allow keeping a large number of SEdit windows around), but it is automatically recreated as soon as the window is expanded again, so this should be effectively invisible. Closing an SEdit window or icon terminates the editing and releases the data structures used.

Unlike DEdit, it is not clear when to consider an SEdit editing session complete. The user may start an edit process, do some editing, shrink the window, expand it again later, etc. For some purposes it is important to have such a notion. For instance, the file package must be informed when a function is edited, so that the new definition can be saved to the appropriate file. If a function has been compiled, and the source is then edited, the compiled code should be discarded. If the editor is invoked on a sequence of structures by Masterscope or EDITFNS, it must know when it is time to go on to the next structure. When a function has been edited, a new comment should be added recording the time and date and the initials of the programmer. All of these cases require some way of indicating that this set of editing operations is completed, even if the editing process is not to be terminated. At present, SEdit handles this by assuming that editing is complete when the window is closed or shrunk.

Confusing SEdit

SEdit currently assumes that no changes will be made to the structure being edited during an edit session, other than those made by SEdit itself. Of course, since SEdit exists in a lisp environment replete with shared structures and destructive functions, there is no way to enforce this. For instance, while editing a variable whose value is a list you could (from an executive window) run a function to destructively change that list. There is no way for SEdit to notice the change as it happens, and at present it will not recover gracefully if it encounters the inconsistency later on.

To avoid causing these problems to itself, SEdit automatically avoids starting two edit processes on the same variable or function. This is only a partial solution, however, and the user is advised to watch out for this problem. If you suspect that a structure being edited has been changed by some other process, simply close the window and start another edit process. This will force SEdit to reexamine the structure.

In the immediate future SEdit will be fixed to detect the most common case of such changes, namely corrections made by DWIM. It is also planned to make SEdit much more robust in the face of other changes it may detect (in the not quite as immediate future).

How SEdit Works

SEdit incorporates a variety of complex algorithms and data structures. Although the code is commented, there is no logical place within it to properly describe the use of these. Therefore this document provides an overview of the most interesting and tricky parts of the program. If you want to understand exactly what the code does, this isn't a replacement for actually reading it. This is, however, a highly recommended introduction to the code; it's probably not worth trying to understand the code if you haven't read this first.

The Code

The code for SEdit currently resides in five files, named SEdit, TopLevel, SEditWindow, Linear, and IntrLsp. The first four of these comprise the SEdit kernel and its interface to the rest of Interlisp-D; the last is the definitions which configure SEdit as an editor for Interlisp code. The approximate division of labor is:

SEdit

process initialization, keyboard command loop, top-level method invocation, building and manipulating the edit tree

TopLevel

interface to Interlisp-D editing functions and file system, starting and managing SEdit processes

SEditWindow

window system interface, mouse selection and pointing, scrolling

Linear

building and manipulating the linear form, optimized screen updating

IntrLsp

methods for the standard Interlisp types

Included with the code listings is a directory of SEdit function names, identifying the file within which they occur. Within each file listing the functions are sorted into alphabetical order. SEdit currently uses the convention of preceding its function names by a pair of backslashes, to clearly distinguish them from other parts of the Interlisp system. At some point a more conventional prefix will be chosen.

Control Flow

SEdit sessions are started via the system function EDITL, which originally invoked the TTY structure editor. When SEdit is enabled, EDITL is modified to call SEdit. SEdit first attempts to find an active SEdit already editing the same function or variable. If one is found, SEdit makes sure that the edit window is expanded and not buried on the screen, and then returns. In this way it avoids starting several SEdit processes on the same structure, which could lead to major confusion. If no active SEdit is found, it will start a new one. It places a window, either by asking the user where the window should be or by using the window position of a previous SEdit (to avoid prompting the user over and over again). Once it has a window, SEdit starts a new process running the keyboard processing loop, and waits for it to signal that it has initialized.

A further complication is that EDITL may be invoked with a sequence of TTY editor commands which are to be performed. Rather than attempt to implement the large and baroque command set of the TTY editor, SEdit simply calls the TTY editor to execute the command sequence, but with the constraint that should the command sequence involve a pause for user editing (i.e. the TTY: command), the TTY editor will call SEdit back. This mechanism is used heavily by Masterscope, which usually passes a sequence of editor commands which search for particular parts of the function and then allow the user to edit them.

If EDITL is invoked without any commands, it returns as soon as the command loop signals that initialization is under way. The editing will be handled by the new process, and whoever invoked SEdit can now go on with other things. Alternatively, if a command sequence is given, SEdit will

wait until the command process signals that the user has indicated editing is complete before returning. This is quite important; if EDITL doesn't wait under these conditions programs like Masterscope may try to start dozens of SEdit procedures, without giving the user time to actually do any editing.

The command loop which is executed by the SEdit process (`\Sedit`) first checks to see if this is a new SEdit context or an old one. If it's new, the structure must be parsed to generate the edit tree and linear form, window parameters must be initialized, and the initial presentation must be displayed. If this is a continuation of an old edit session, any pending adjustments to the presentation are completed, and the current selection is redisplayed. Either way, `\Sedit` then enters a loop reading single characters and executing the appropriate action.

Keyboard Input

The interpretation of characters typed from the keyboard is determined by an Interlisp readtable, which maps characters to syntax classes. The syntax classes understood by SEdit are:

STRINGDELIM: this is the delimiter for strings (i.e. double quote)

SEPRCHAR: white space, e.g. blank and carriage return

ESCAPE: when preceded by this character, other characters should be treated as syntax
OTHER (the usual escape character is %)

OTHER: a "normal character", suitable for inclusion in the pname of a litatom

(*type where when function*): a read macro. *function* is the function to be invoked when this character is typed. *type* determines what information will be passed to the function, and how its result will be interpreted; at present the only legal value is `INFIX`, indicating that the function will be passed the complete edit context, and is free to make any changes whatsoever. *where* and *when* control under what conditions the function will be invoked; the valid combinations are:

(ALWAYS IMMEDIATE)
the function will be invoked whenever the character is input (e.g. Delete)

(ALWAYS NONIMMEDIATE)
the function be invoked except when the character is typed as part of a string (e.g. left parenthesis)

(FIRST NONIMMEDIATE)
the function will only be invoked when this character is typed at a structure point, i.e. as an atom on its own (e.g. period)

Note that these are not exactly the conventional interpretation of readtable entries, but have been adapted to SEdit's needs.

The action produced by a character depends on both the syntax class of the character and the type of the current point (string, atom, or structure). For instance, a `SEPRCHAR` typed to a string point will insert that character in the string, typed to an atom point will split the atom, and typed to a structure point will simply be ignored. An `OTHER` character typed to a string or atom point will be inserted, but typed to a structure point will cause a new node, representing the single character atom, to be created and inserted.

Mouse Actions

In the Interlisp-D window system the effects of mouse movement and buttons are determined by a special mouse process, which invokes functions attached to the windows. The effect of this is that activities like selecting and pointing are actually run under the mouse process, rather than under the SEdit process. This has several ramifications. First, the mouse process must be able to easily access the state of the editing process, so that it has the necessary information available to carry out the actions. For this and other reasons the entire state of the SEdit process is encapsulated in a single data structure, called an `EditContext`, which is then attached as a property of the window.

Second, access to the editor's data must be controlled, so that multiple processes don't interfere with each other. This is achieved through a monitor lock on the `EditContext` (called the `ContextLock`).

The Linear Form

The data structure used to encode the linear form must satisfy several constraints. First, as its name suggests, it is treated as a sequence of items. The window is repeatedly being redrawn from the linear form, and usually the segment which is redrawn cuts across the tree structure. This requires being able to start iterating through the linear form from any point, and continue until some other arbitrary point, preferably without having to maintain a stack. On the other hand, it is generated and modified hierarchically; the kernel must be able to quickly extract and replace the linear form for any node in the tree, without affecting the nodes above or (in some cases) below it. To achieve this, the linear form for each node is stored as a list pointed to by the node, which ends with a pointer back to the node. This list may contain linear items and subnodes of the node, indicating that their linear forms are to be inserted at that point. Finally, each node contains a pointer (called the `LinearThread`) which points to the position in its supernode's linear form at which it appears. This threading allows any part of the tree structured linear form to be easily and efficiently traversed.

One complication to this scheme is that because of the reference counting garbage collector in Interlisp-D, we don't actually make the tail of the linear form point back to the node directly. This would introduce a circular chain of pointers, preventing these structures from ever being garbage collected. Instead, we use a data structure called a `WeakLink`, which contains one non-reference counted pointer field. Of course, the use of such pointers can make debugging hazardous; unfortunately, they are pretty much unavoidable in memory management scheme such as Interlisp-D's, and `SEdit` uses them in a number of places.

Most of the items which can appear in the linear form are relatively simple, but `LineStarts` are the exception. `LineStarts` are inserted by the linearization procedures to indicate that a new line is to be started, but are used by the kernel a lot of extra information. First, each `LineStart` is linked to the `LineStart` before and after it, allowing efficient access to sequential lines of the linear form. Second, each `LineStart` records the maximum ascent and descent of the items which appear on that line, since these determine the amount of vertical space required to determine the line. Since lines can contain arbitrary combinations of text in different fonts and sizes, as well as bitmaps of arbitrary size, each line's ascent and descent must be separately computed, and may change with any change to the nodes appearing on that line. Third, each `LineStart` has a pointer to the node in whose linear form it appears; although this could be determined by scanning along the linear form, it's used often enough that we cache it. Fourth, each `LineStart` stores the y coordinate of that line's baseline. This is a function of that y coordinate of the preceding line, the line ascents and descents, and the separation between the lines.

Thus, the algorithm for updating the window from the linear form is something like this:

```
(* pointer is the current position in the linear form)
while pointer is not NIL do
  if pointer is a list, then
    let item be the first element of the list
    if item is a number (* horizontal space), then
      increment current x by item
    elseif item is a LineStart, then
      set x to the indentation of item
      set y to the baseline of item
    elseif item is a bitmap or string item, then
      paint this item and increment x by its width
    else item is a subnode
      set pointer to the linear form of item
  else pointer is a WeakLink
    set pointer to the CDR of the linear thread of
    the node pointed to by this WeakLink
```

Incremental Relinearization

When the node structure is changed, SEdit is faced with the task of updating the screen as efficiently as possible. This is actually treated as two problems: first, the new linear form must be computed as efficiently as possible, and second, given the changes that occurred in the linear form, we wish to make the corresponding adjustments to the window, with a minimum of repainting.

The first problem is dealt with using a number of tricks. First, as changes are made to the tree no attempt is made to update the linear form immediately; all that is done is to keep a list of those nodes which have been changed and thus need their linear forms recomputed (this is the purpose of `\\note.change`). This list is kept sorted by depth, and duplicates are discarded. Once SEdit decides that it's time to update the window, it finds a minimal set of nodes to relinearize, such that all of the changed nodes are contained directly or indirectly (as subnodes of an included node), and the width estimates of these nodes have not been changed by the editing (hence the linear form of their super nodes won't have changed). The linear form of each of these nodes is recomputed. Second, during this relinearization, subnodes of a relinearized node will not be relinearized if their structure hasn't changed, and SEdit is able to determine that their resulting linear form won't have changed (e.g. their margins haven't changed). In practice, this typically results in a major decrease in the amount of relinearization involved.

The second problem is more difficult. The idea is that we want to reuse as much of the existing window display as possible. If part of the linear form is already displayed in the correct place, no changes should be made to it. If part of it is already displayed, but in the wrong place, SEdit tries to use a `bitblt` call to copy those bits to the correct location rather than reconstructing them. This is complicated by the fact that material copied may be less than one line or span several lines. The line its moved to may have greater or less ascent and descent than the one it came from; in fact, the ascent and descent of the destination line won't be known until it's completed. To deal with this, SEdit builds a second description of the line, in parallel with the construction of the linear form. This structure describes the line as a sequence of blocks, each of which represents a subsequence of the linear form. Each block may also already appear in the window, in which case the coordinates from which it can be retrieved are also recorded. At the end of each line, SEdit examines the sequence of blocks, determines which ones actually represent useable blocks of bits, copies them, and repaints any gaps. To avoid overwriting bits which it may later want to use, SEdit will sometimes shift whole parts of the screen out of the way; this introduces additional complications, since coordinate transformations are now required to determine the actual location of the desired bits.

A Word About Lines and LineStarts

SEdit defines a `LineStart` data type to record information about a line in the linear form. The `LineStart` does not record the actual linear items which appear on that line; that information is implicitly recorded in the linear form itself. The linear form is a list of linear items, and `LineStarts` are items. Thus, the items on a line are those which follow it in this list (until the next `LineStart`). Since this information is often needed, many of the places where one might expect a pointer to a `LineStart` actually contain a pointer to the linear form which start at that `LineStart`, i.e. a list, the first element of which is the `LineStart`. This is referred to as a `Line`.

Pretty Printing

The code listings at the end of this document were generated by SEdit's formatting routines. This seemed like an obvious thing to do, but a couple of caveats are in order. First, the formatting rules are still incomplete; in particular, they do a miserable job on **create** expressions (which have a very non-LISP like syntax). Second, quite little modification was required to add this capability to SEdit, but the result isn't an ideal pretty printer. It does a good job (better than Interlisp's pretty printer), and hopefully will soon do an even better job, but it's a very expensive way to do it. Constructing the complete edit tree for a one-shot linearization consumes excessive memory and time, and consequently printing large files of functions requires considerable patience.

Contexts, in detail

(an annotated description of the `Context` data structure definition)

Environment: an `EditEnv`

The `Environment` provides a number of parameters controlling the editing process. The idea is that different environments are built to describe different editing tasks (editing different languages, or just different edit styles), and then shared between all edit contexts of that type.

DisplayWindow: a `WINDOW`

This is the primary window, in which the edited structure is displayed.

EditType: probably a `litatom`

Doesn't affect the editing, but used (in conjunction with `IconTitle`) to help identify the source of the structure being edited. One of `VARS`, `FNS`, `PROP`, etc.

IconTitle: a string or `litatom`

The name of the structure being edited, used to title the window and icon. Also, in conjunction with `EditType`, used to determine whether an existing `SEdit` session is already editing a structure the user has asked to `SEdit`.

ContextLock: a `MONITORLOCK`

The monitor lock used to control access to this `Context`.

CompletionEvent: an `EVENT`

`SEdit` will signal this event when the user shrinks or closes the window, or otherwise indicates that they have done enough editing. When called under the TTY editor, the process which spawns the `SEdit` command loop will wait for this event.

WindowLeft, WindowRight, WindowBottom, WindowTop: integers

During screen update processes, `SEdit` caches information about the window dimensions here, since they're used so frequently.

CurrentX: an integer

CurrentLine: a `Line`

While generating the linear form, `SEdit` uses these to record the start of the current line and the horizontal position within that line.

LinearPointer: a position within the linear form

This is the current position within the linear form, for purposes of comparison and insertion.

LinearPrev: a position within the linear form

This is one step behind the `LinearPointer`, to allow fixing up pointers when an item is inserted. If `LinearPointer` points to the first item within the linear form of a node, `LinearPrev` will point to that node.

Root: an `EditNode`

Points to the root node in the edit tree.

LastLinearizedSubNodeIndex: an integer

During linearization, this field is used to record the last subnode linearized of the current node (as a consistency check).

ChangedNodes: a list of `EditNodes`

This list records which nodes in the tree have had their structure changed and hence require relinearization. It's headed with a dummy item (`NIL`) to simplify insertion, and sorted by decreasing depth. Most of the time, this list should contain exactly those items whose `Changed?` field is `T`.

CaretPoint: an `EditPoint`

Records the current insertion point.

Selection: an `EditSelection`

Records the current selection.

Caret: a CURSOR

This is the mark to be displayed at the current caret point (hollow or solid, depending on the type of point).

SelectionDisplayed?: a boolean

Records whether the current selection has been underlined, outlined, or whatever.

LastMouseX, LastMouseY: integers

LastMouseEvent: one of (Atom, Structure)

Records the position of the last mouse click, and the button used, to detect multi-click sequences.

\X, \T: integers

These slots in the EditContext are used at one point in the program to return multiple values from a function; too bad there's no better way.

FirstBlock: a LineBlock

The first in the sequence of blocks constructed to describe the current line.

CurrentBlock: a LineBlock

The last (so far) in the sequence of blocks being constructed to describe the current line.

Matching?, Below?, Visible?: booleans

These are used while constructing the block sequence to keep track of our current state.

RepaintStart: a position within the linear form

RepaintLine: a LineStart

RepaintX: an integer

Also used to keep track of state while constructing the block sequence.

ShiftY, ShiftDown, ShiftRight: integers

When the block shifter moves the rest of the window contents out of the way, these variables record what has been done to allow translating coordinates. Everything below ShiftY has been shifted down by ShiftDown; everything on the current line past the current position has been shifted right by ShiftRight.

RelinearizationTimeStamp: an integer

During relinearization, the position of a line may move. However, the old position of that line may be later needed to locate useful bits. When the position or dimensions of a possibly useful line are changed, the old values are cached. Since the values are only good for the current relinearization, they are marked with this time stamp, and each relinearization its value is increased.

Environments, in detail

(an annotated description of the Environment data structure definition)

ParseInfo: a PLIST

This PLIST maps TYPENAMES of edited data structures to the functions which can parse them.

ParseInfoUnknown: a function name

This is the function called to parse any data structure whose TYPENAME doesn't appear in ParseInfo.

DefaultFont, ItalicFont, KeywordFont: font descriptors

These are the fonts to be used when formatting.

DefaultLineSkip: an integer

If the linearization procedure doesn't specify the vertical spacing between lines this value will be used.

ReadTable: a `READTABLE`

This is the table used to determine the syntax of keyboard input and the interpretation of command characters.

SpaceWidth: an integer

This is the default space to leave between adjacent items in lists, etc.

DefaultIndent, MinIndent, MaxIndent, MaxWidth: integers

These values control the formatting of list structures.

LParenString, RParenString, DotString, QuoteString: `StringItems`

To prevent repeatedly constructing `StringItems` for the standard punctuation symbols, they're cached in the environment and shared. These must be reconstructed if the fonts are changed.

EditNodes, in detail

(an annotated description of the `EditNode` data structure definition)

NodeType: an `EditNodeType`

The type of this node, providing the set of methods.

ParseMode: a `litatom`

The parse mode in which this node was parsed.

SuperNode: an `EditNode`

This node's super node; `NIL` if this is the root.

Depth: an integer

The depth of this node within the tree; the root has depth 0.

SelfLink: a `WeakLink`

To avoid building uncollectable circular structures, the linear form ends with `WeakLink` back to the node. To avoid consing `WeakLinks`, we cons one and cache it here.

SubNodeIndex: an integer

This is the index of this node within its super node's subnodes.

Structure: anything

This is the structure this node actually represents.

Changed?: a boolean

True if this node has been changed and will require relinearization. Most of the time, true iff this node is on the context's list of changed nodes.

InlineWidth, PreferredWidth, PreferredLLength, MinWidth, MinLLength: integers

The width estimates computed by this node's `ComputeFormatValues` method.

SubNodes: a list of `EditNodes`

The subnodes of this node.

LinearForm: a list of linear items

The linear form of this node, terminating in a `WeakLink` back to the node.

LinearThread: a list of linear items

The position of this node within its super node's linear form (hence a list whose `CAR` is this node)

Unassigned: anything
Available for use by this node's methods to cache any additional information they wish to record.

StartX: an integer
The horizontal position at which the linear form of this node begins.

RightMargin: an integer
The right margin used when formatting this node.

ActualWidth: an integer
The maximum horizontal offset of any part of the presentation of this node from the starting position.

ActualLength: an integer
The offset of the end of the last line of this node's presentation from the node's starting position.

FirstLineLinear, LastLineLinear: Lines
The line on which this node's linear form begins, and the line on which it ends.

Inline?: a boolean (computed)
True iff FirstLineLinear and LastLineLinear are the same.

FirstLine, LastLine: LineStarts (computed)
The CARs of FirstLineLinear and LastLineLinear.

EditNodeTypes, in detail

(an annotated description of the EditNodeType data structure definition)

Name: a string or litatom
This is not used by SEdit, but provides a handy point of reference when debugging.

ComputeFormatValues, Linearize, ReParse, SubNodeChanged, SetPoint, ComputePointPosition, ComputeSelectionPosition, SetSelection, GrowSelection, SelectSegment, Insert, Split, Delete, Replace, CopyStructure, CopySelection, BackSpace : function names
These are the methods of this node type.

LineStarts, in detail

(an annotated description of the LineStart data structure definition)

NextLine, PrevLine: Lines
These lines immediately before and after this one in the linear form.

Node: an EditNode
This is the node in whose linear form this LineStart was generated.

LineAscent, LineDescent: integers
The maximum ascent and descent of any item on this line, and hence the dimensions of the line.

LineSkip: an integer
The vertical separation of this line from the line preceding it.

Indent: an integer
The amount by which this line is horizontally indented.

LineLength: an integer
The total length of this line (including indentation).

YCoord: an integer

The vertical position of the top of this line (including the LineSkip). Since we set up the window's coordinate system with (0,0) in the top left corner, YCoords are always negative or zero.

CachedY, CachedAscent, CachedDescent: integers

During relinearization, we may need to determine what the position and dimensions of a line were after we've changed them. When this is a possibility the old values are saved here.

CacheTime: an integer

When the old position and dimensions are cached, the current value of the context's RelinearizationTimeStamp is recorded here, so that we can quickly determine when the cached values are out of date.

LineHeight: an integer (computed)

The sum of LineSkip, LineAscent, and LineDescent.

BaseLineY: an integer (computed)

YCoord minus the sum of LineSkip and LineAscent.

NextLineY: an integer (computed)

YCoord minus LineHeight; should be equal to the YCoord of NextLine.

OldTop, OldBottom: an integer (computed)

If the cache values are up to date (comparing their time stamp with that of the context) use them; otherwise use the current values.

StringItems, in detail

(an annotated description of the StringItem data structure definition)

String: a litatom or string

The text to be displayed.

Font: a font descriptor

The font in which the text should be displayed.

PRIN2?: a boolean

True if the PRIN2-name of String should be used, rather than the pname (i.e. string delimiters will be displayed, command characters will be escaped).

Width: an integer

The width of this item when displayed.

EditPoints, in detail

(an annotated description of the EditPoint data structure definition)

PointNode: an EditNode

The owner of this point, i.e. the node in which insertion will take place.

PointIndex: an integer

Used by the PointNode to record the point's position within the node.

PointType: one of (Structure, Atom, String)

Determines how characters typed at this point are to be interpreted.

PointLine: a LineStart

PointX: an integer

The position in the window at which the flashing caret should be displayed while waiting for keyboard input.

EditSelections, in detail

(an annotated description of the EditSelection data structure definition)

SelectNode: an EditSelection
The owner of this selection.

SelectStart, SelectEnd: integers
Used by the SelectNode to record the selection's boundaries within the node.

SelectType: one of (Structure, Atom, String)
If typed input replaces this selection, this determines how it will be interpreted.

SelectStartLine, SelectEndLine: LineStarts
SelectStartX, SelectEndX: integers
The boundaries of the area to be highlighted when this selection is displayed.

DeleteOK?, ReplaceOK?: booleans
The use of these flags is not completely implemented yet.

LineBlocks, in detail

(an annotated description of the LineBlock data structure definition)

BlockStart: a position within the linear form
The start of the segment of linear form this block represents (the end is determined by the start of the next block).

BlockNewX: an integer
The horizontal position at which this block will be displayed.

BlockWidth: an integer
The width of this block when displayed.

NextBlock: a LineBlock
The next block in the sequence.

Bits?: a boolean
True if the segment of linear form represented by this block is already displayed on the screen.

BlockX, BlockBaseLine, BlockAscent, BlockDescent: integers
If Bits? is true, these fields give the current position of the block's presentation.

An Extensible Structured Data Editor for Interlisp-D

Lisp environments are populated by a variety of complex linked data structures, particularly linguistic structures (such as programs written in Lisp and other languages). These are often designed with visual representations (e.g. the pretty-printed form of Lisp code). Through such features as read macros and print definitions, the data structures may be converted to or from a textual representation. In many cases, the only convenient way to edit the structure may be to edit the textual representation and then re-interpret it. This approach suffers from several difficulties:

- writing the structure out and reading it back in will generally create an entirely new structure; this may be a problem if the structure is shared or part of a larger structure
- the requirement that enough information be written out to reconstruct the data when it is read back may conflict with the desire that the presentation be convenient to understand and change
- if the text editor is to provide any assistance (syntax checking, semantic checking, etc.) it must be integrated with the Lisp environment, and continually translating structures to and from their textual representation
- no allowance is made for nontextual graphic presentations

SEdit provides a different approach. Data structures are edited directly, with visual presentations provided as a means of communication. Like WSIWYG text editors, the editing window contains a continuously updated presentation of the data structure, and editing operations are input in terms of this presentation. Presentation and editing rules are defined for each data type, and a kernel program uses these rules to perform the editing.

A set of such rules have been written to configure the editor as a tool for editing Interlisp programs. Interlisp code is pretty-printed in the window, and editing operations use a simple "point and type" user interface. This provides a more convenient way of editing Interlisp than previously existing tools, and furthermore will allow the simple construction of editors for other languages implemented in the Interlisp environment (Prolog, CommonLisp, 3-Lisp, Loops, etc.).

A Sample Session

Before discussing the structure and implementation of SEdit, we will try to give a feel for what it does by a brief description of some simple editing. Of course, it is a little hard to convey the flavour of as interactive a process as editing with just a few words and pictures. This discussion assumes some familiarity with the uniform editing interface which many Interlisp-D programs (and in fact most Xerox software) supports, since that was the model for SEdit's interface. The basic principles are that

- there can be a *current insertion point*, usually indicated by some sort of caret, at which inserted material appears
- there can be a *current selection*, usually indicated by underlining or other highlighting, which indicates the material to be affected by a following command (commands may change the selected material, change some aspect of it (such as font or formatting), delete it, etc.)
- the point and selection are usually placed by positioning the mouse and pressing a button; the left or middle button places the point near the cursor and selects the indicated material, while the right button extends a previously made selection to include the indicated material
- material may be selected while holding down a modifier key; instead of becoming the current selection, an action is performed as soon as the key is released. The standard modifiers are Copy, which inserts a copy of the selected material at a previously

chosen insertion point; **Delete**, which deletes the selected material; and **Move**, which copies the selected material and then deletes it

To SEdit an Interlisp function after SEdit has been loaded, use the usual (**DF** *function*). This creates an edit window for the function. Unlike the older display editor (**DEdit**), **DF** will return immediately, since the editing is done in a separate process. The function body will be pretty-printed in the window:

```
SEdit fibonacci
(LAMBDA (n)
  (* mdd " 3-May-86 18:26")
  (if (LEQ n 1)
    then n
    else (PLUS (fibonacci (SUB1 n))
               (fibonacci (DIFFERENCE n 2))))))
```

Since SEdit knows that this is Interlisp code, it uses some formatting rules specific to Lisp as well as its general rules for formatting lists, atoms, and strings. Formatting rules specify visual presentations as the positioning of text strings (in any font) and arbitrary bitmaps.

SEdit allows operations both on complete Lisp objects (such as lists and atoms) and on parts of objects, such as the individual characters in an atom's name. To do this conveniently, it uses the convention that the left mouse button is used to point to parts of structures, while the middle mouse button always points to whole Lisp structures. Thus, left-clicking the **Q** in **LEQ** selects that character, but middle-clicking there will select the whole atom (this convention matches TEdit's character/word selection convention). Furthermore, the same convention applies to insertion; a left click between the **E** and **Q** will allow inserting more characters, but a middle click will allow inserting a new structure (after the **LEQ**, since the mouse was closer to the end of the atom than the beginning). SEdit indicates the type of insertion expected by changing the caret, to **▲** for a substructure and to **■** for a structure. Thus, left-clicking the right side of the **Q** in the above window produces:

```
SEdit fibonacci
(LAMBDA (n)
  (* mdd " 3-May-86 18:26")
  (if (LEQ▲ n 1)
    then n
    else (PLUS (fibonacci (SUB1 n))
               (fibonacci (DIFFERENCE n 2))))))
```

and any characters now typed would be appended to the atom **LEQ**. On the other hand, middle-clicking in the same place produces:


```
SEdit fibonacci
(LAMBDA (n)
  (* mdd " 3-May-86 18:26")
  (if (LEQ n 1)
    then n
    else (PLUS (fibonacci (SUB1 n))
               (fibonacci (DIFFERENCE n 2)))))
```

and if characters were inserted they would form a new atom.

Enclosing structures are selected by multiple clicks. In the previous example, if the middle button were clicked twice, the list (LEQ ...) would be selected; if it were clicked thrice, the (if ...) would be selected, etc. Sequences of structures or substructures are selected by extending the current selection with the right mouse button. If the atom LEQ has been selected, right-clicking the 1 would select the sequence of three structures LEQ, n, and 1 (this is *not* the same as selecting the list which contains these structures). Similarly, the onacc in the middle of fibonacci could be selected by left-clicking at one end of the sequence and right-clicking at the other end.

Some characters activate SEdit commands. For instance, typing a left parenthesis inserts an empty list at the current insertion point and positions the point inside it. Typing a right parenthesis positions the point just after the list immediately enclosing the current point. Typing a blank when inserting within an atom splits the atom at the point (unless the point is at one end of the atom) and switches to structure insertion. This allows Lisp list structures to be typed in as usual. For instance if we continue the above example by typing "(a b " the window will now appear like this:

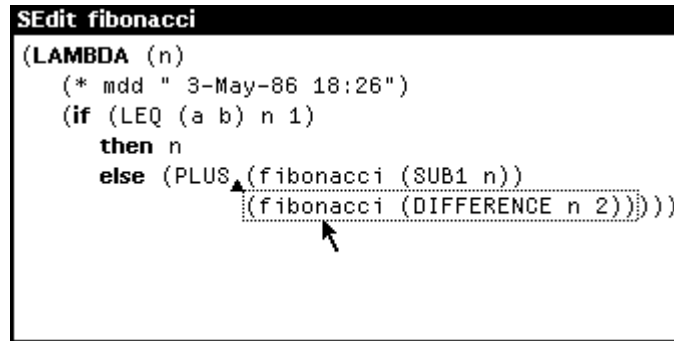
```
SEdit fibonacci
(LAMBDA (n)
  (* mdd " 3-May-86 18:26")
  (if (LEQ (a b) n 1)
    then n
    else (PLUS (fibonacci (SUB1 n))
               (fibonacci (DIFFERENCE n 2)))))
```

(Note that the matching right parenthesis appears as soon as the left parenthesis is typed; this is an immediate consequence of having the window always be a pretty-printed presentation of the underlying list structure. Double quotes work the same way for strings.)

Modified selections allow the easy rearrangement of existing structures. For instance, to switch the order of the two arguments to PLUS, simply

- a) middle-click after PLUS (sets the insertion point)
- b) hold down the Move key (we are going to move part of the structure to the current insertion point)
- c) middle-click on the second fibonacci twice (selects the whole function call)
- d) release the Move key (to indicate the selection is completed)

Just before the Move key is released, the window looks like this:



```
SEdit fibonacci
(LAMBDA (n)
  (* mdd " 3-May-86 18:26")
  (if (LEQ (a b) n 1)
    then n
    else (PLUS (fibonacci (SUB1 n))
               (fibonacci (DIFFERENCE n 2))))))
```

The selection is highlighted by a dotted outline to indicate that this is a Move selection (Copy selections have a dotted underline, pending-replace selections have a solid outline, and Delete selections are inverted). As soon as the change is made, the window is updated to show the resulting structure (SEdit determines what changes to the presentation result from the editing, and repaints as little of the presentation as possible).

An Overview of the Control Structure

The operations demonstrated above are actually performed by the cooperation of three separate pieces of code: the SEdit kernel, the SEdit user interface, and a set of methods defining the datatypes to be edited. By factoring the control this way, SEdit can be customized to edit new datatypes or provide different commands without having to modify or understand the complexities of the kernel (this is fortunate, since the kernel is quite complicated). The kernel invokes the datatype methods at appropriate times, manages the global sequencing of operations, and handles the optimization and caching necessary to update the screen efficiently. The datatype methods fall into three classes:

- methods for effecting and responding to changes in the structure being edited
- methods for positioning the point and selection appropriately
- methods for generating the visual presentation of the structure

The user interface translates mouse and keyboard events into appropriate calls on the SEdit kernel operations. It is not as modular as I would like; the keyboard event interpreter is table driven and readily extensible, but the mouse event interpreter is not extensible at present.

Each instance of SEdit has a *context*, which includes the structure being edited and the window in which the editing is being done. Editing operations are performed by calling kernel procedures and passing the context. The context is monitor locked, so operations may be invoked from several processes. Normally, each context has an associated keyboard process which reads keyboard commands and performs them. No editing state is maintained by the keyboard process; it can be deleted and recreated as convenient (but SEdit ensures that there is never more than one keyboard process per context).

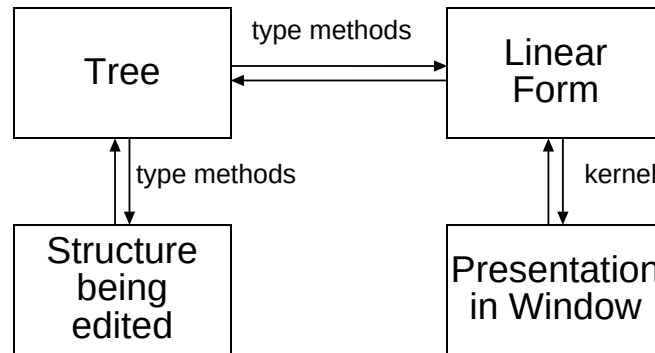
SEdit is installed by calling the function (`EDITMODE 'SEdit`), whereupon it becomes the system display editor and will be used whenever the user asks to display edit a structure (through edit commands, Masterscope searches, inspectors, browsers, or whatever). The context remains active until the edit window is closed (although the keyboard process disappears while the window is shrunk).

An Overview of the Data Structures

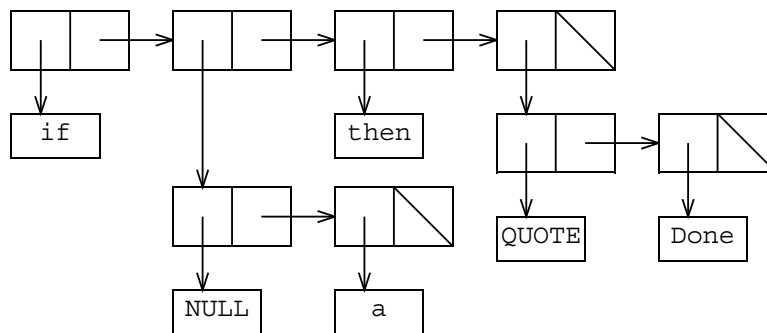
The SEdit user is aware of two data structures: the structure being edited, and the window in which it is presented. Internally, SEdit maintains two additional data structures: the *tree*, which is a representation of the structure being edited, and the *linear form*, which is a representation of the visual presentation appearing in the window. The tree provides a common representation for the underlying structures. This allows SEdit to implement a small number of simple editing operations uniformly described as tree mutations, while the type-specific methods map these into the actual changes required. It also caches various bits of information computed by SEdit. The linear form is a

sequence of strings, atoms, and bitmaps to be displayed, intermixed with horizontal and vertical positioning information.

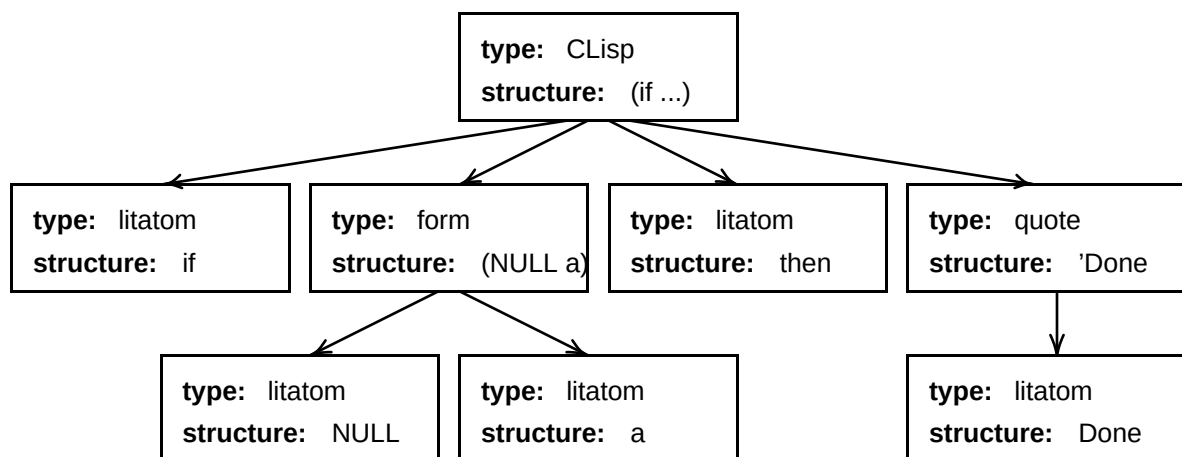
These four data structures are all representations of the same data. The constraints between them are maintained as follows:



As an example, consider editing the list structure `(if (NULL a) then (QUOTE Done))`. The underlying structure is composed of cons cells and atoms:



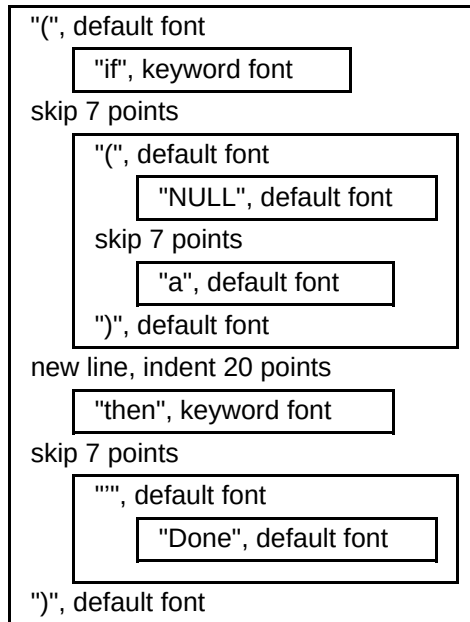
SEdit's tree will look like this:



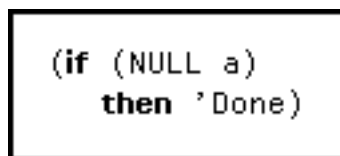
Note that each node in the tree has a type, indicating the type of the structure, and a field containing the structure itself. Note also that the types need not correspond directly with Interlisp's type

structure: in this example, many cons cells are grouped together as one node in the tree, and may be marked as type `list`, `CLisp` (a class of special form), `form`, `quote`, `binding`, etc. depending on the components of the structure and its position in the surrounding structure. Thus, the list (`QUOTE Done`) is classified as type `quote` above (and has one subnode), but would have been type `binding` (with two subnodes) had it appeared in a list as the second element of a form identified as a `LET` or `PROG`.

The linear form will be something like:



Note that although the linear form is a sequence (hence its name), it reflects the structure of the tree from which it was generated. Finally, the window will show some or all of the presentation described by the linear form:



SEdit has two other important data structures. The context was mentioned before; it includes the data structures mentioned above, information such as the current selection and current insertion point, and all the other little bits of state SEdit needs to keep track of what is going on. The *environment* contains all of the customization information which controls SEdit's behaviour: the tables which drive the generation of the tree from the original data structure, the table for interpreting keyboard events, parameters such as default spacing and fonts, etc. At present, there is just one environment defined — the one which configures SEdit specifically for editing Interlisp code, but by creating new environments the user could have SEdit contexts for editing Prolog, CommonLoops, etc., all coexisting.

(Actually, there *is* a second environment defined, which tricks SEdit into pretty-printing Interlisp code into a TEdit document (to produce the listings later in this document), but it hardly qualifies as an *edit* environment.)

Documentation

The rest of the documentation for SEdit is in five sections:

- a more detailed account of how to use SEdit to edit Interlisp code
- a description of how SEdit is configured for editing a new type
- a detailed description of SEdit's internals
- an annotated listing of the code which implements the basic Interlisp types
- an annotated listing of SEdit's kernel

The first of these is independent of the others, but the second should be read before attempting to understand any of the following sections. If you are planning to extend SEdit to edit your own language (or change the way it currently edits one), you should also look at the third of these — a few good examples are sure to convey what the specifications missed.

Other Comments

The formatting and incremental reformatting algorithms, the incremental window update algorithms, and the various data structures were all developed by the author, so there are no references for them. Although there are several extensible prettyprinters for Lisp environments (notably Waters' GPRINT), some do not allow a sufficiently general class of presentations, most only deal with formatting lines of fixed width characters, and none deal with the (crucial) problem of incremental reformatting.

At present, SEdit is usable but far from polished. There is still room for improvement in the screen updating, and many rather desirable commands are still missing (notably *undo* and *redo*). Extra formatting rules for CommonLoops are a high priority, as is completing the interface to other Interlisp tools (e.g. the file package). Finally, the author hates the name SEdit, and promises three free features of their choice to the first person who comes up with one he likes better.

How Sedit Formats LISP Forms

If you have a list format associated with the car of a lisp form, sedit will use the information in that list format to prettyprint the form. The details of internal list format structure are documented elsewhere; this note describes an easy-to-use interface to these internal structures.

Getting and Setting Formats

Formats are associated with the cars of lists. To find out what format is associated with a particular function name, say FOO, you say (GET-FORMAT 'FOO). This returns three values: (1) The external form of the format spec for FOO (described in the next section), (2) the internal form of the format spec for FOO (described elsewhere), and (3) one of the symbols :EXTERNAL (meaning FOO had both an external and an internal format), :INTERNAL (meaning FOO had an internal format but no external one), or NIL (meaning FOO has no associated format). To associate a format with 'FOO, you say (SETF (GET-FORMAT 'FOO) FSPEC), where FSPEC is an external format spec (as described below). If you want to give FOO the same format as BAR, you can (of course) say (SETF (GET-FORMAT 'FOO) (GET-FORMAT 'BAR)), as in (SETF (GET-FORMAT 'DO*) (GET-FORMAT 'DO)).

SEdit's initial formatting information is broken into two parts: those formats which only exist in internal form and those which exist in both internal and external form. The internal-only formats are not of concern to us here (they really exist only for special interlisp constructs like CLISP). The external formats are all gotten by looking at entries on the *LISP-FORMAT-ALIST*, which we describe below. The function (RESET-FORMATS) resets SEdit's formatting info to its initial state, and RESET-FORMATS is called as part of the initialization sequence. The function INSTALL-FORMAT-ALIST takes an alist with entries like those on the *LISP-FORMAT-ALIST* and installs the specified formatting information. INSTALL-FORMAT-ALIST also arranges that any calls to RESET-FORMATS will reinstall that formatting information in addition to the default information. It would be best, however, to get rid of format alists entirely and use defdefiners.

Documentation of Entries in the SEDIT:*LISP-FORMAT-ALIST*

Each entry in this list should be a symbol (or list of symbols) dotted with a format specification. The meaning of each entry (NAME . FSPEC) is: any list L whose car is NAME (or a member of NAME if it's a list) should be formatted according to FSPEC.

Format Specifications

A format specification consists of an indentation specification (described below) followed by a bunch of options in PLIST format. The allowed options are:

:INLINE -- value can be T or NIL (default NIL). If T, the form will go all on one line if it fits. If NIL, the form will be broken across lines at arg boundaries even if it would all fit on one line. For example, OR has :INLINE T and LET has :INLINE NIL.

:MISER -- value can be :ALWAYS, :NEVER, or :TOFIT (default :TOFIT). Specifies when to use miser indentation. The default means use miser indentation if non-miser indentation would force the arguments into miser indentation.

:ARGS -- value should be a list of format specifications. These formats are assigned to the elements of the list L in order starting with the first element (which will be NAME). Note that these formats override any formats that would normally be assigned to the elements of L (based on their first elements). NIL is allowed in the :ARGS list, and means do *not* override the format of this element; that is, allow it to be

formatted normally. Also, a symbol *S* is allowed in the `:ARGS` list if *S* has earlier been assigned a format; this means to assign *S*'s format to this element. There are also two special keywords allowed as entries in the `:ARGS` list: `:KEYWORD` and `:RECURSIVE`. `:KEYWORD` means that if the element assigned this format is a symbol then treat it like a keyword, i.e., put it in bold face. (This list uses the convention that all symbols which allow declarations in their body [such as `DO` and `LET`] are formatted as keywords.) `:RECURSIVE` means to assign this element format `FSPEC`; that is, the entire top level format is assigned recursively to this element. This is very useful for formats like `:DATA` format. If *L* has more elements than there are entries in the `:ARGS` list, the last entry in the `:ARGS` list is repeated for all the extra elements of *L*. Hint: most `:ARGS` entries have `NIL` as their last element. If no `:ARGS` list is specified, the elements of *L* get their natural formatting.

:LAST -- value should be a format specification like those in the `:ARGS` list. This format specification will be applied to the last element of *L* but *only* if doing so would supercede the last entry in the `:ARGS` list. In other words, if the last element of *L* would receive the repeated format from the `:ARGS` list, it gets the `:LAST` format instead. This option is really only useful for pathologically formatted forms like Interlisp's `SELECTQ`.

:SUBLISTS -- value should be a list of element positions (counting from 1) or `T`. `T` means all of the arguments should be parsed as lists even if they are `NIL` (so `NIL` will display as `()` rather than `NIL`). A list of element position means those element positions will be parsed as lists. For example `LET` has `:SUBLISTS (2)` meaning the second element of a form whose first element is `LET` is a list (i.e., the binding list). `DO` has `:SUBLISTS (2 3)`, `DEFUN` has `:SUBLISTS (3)` and `COND` has `:SUBLISTS T`. Default is `:SUBLISTS NIL` meaning print all `NIL` args as `NIL` not `()`.

Indentation Specifications

An indentation specification is either a symbol (normally a keyword) or a list. If it's a symbol, it's looked up on the `SEDIT::*INDENT-ALIST*` (which see) and the `SEdit`-internal indent specification found there is used. If it's a list, it consists of some optional keywords (described below) followed by argument group specifications. Each argument group specification is either a number or a list containing a single number. In both formats, the number indicates that that many arguments should be grouped together at a single indentation level. The simple number format means that each of those arguments should go on its own line (they will line up vertically with each other), while the number-in-a-list format means that the arguments in the group can go together on a single line if they fit. The indentation level for each argument group is determined by how many groups follow it in the indentation list. Each group is indented 1 level further in than the group which follows it; thus, the first argument group is indented most, the next one next most, and so on until the last one, which is always indented one step in from lambda-body level.

This is best explained with examples. A simple example is `LET`, whose indentation specification is `(1)`. This means that `LET` will be followed by a single distinguished argument group consisting of one element (the binding list) which will be indented one step in from the `let` body. Another simple example is `DO`, whose indentation specification is `(2)`. This means that `DO` will be followed by a single distinguished argument group consisting of two elements (the binding list and the termination clause) which will be indented one step in from the `do` body. It also means that the bindings and the termination will be required to go on separate lines. Contrast `DO` with `DEFUN`, whose indentation specification is `((2))`. Like `DO`'s spec, `DEFUN`'s spec says there is one group with two members (the name and the lambda-list), but unlike `DO`'s spec, `DEFUN`'s spec says that the first two args can go on the same line if they fit there. Finally, consider a possible spec for `MULTIPLE-VALUE-BIND` of `(1 1)` which says that the first group consists of one arg (the variable list) and the second group consists of one arg (the form to eval). The form to eval will be indented one step in from the body, and the list of variables will be indented one step in from there.

Note that a group specification of 0 (zero) is allowed: this occupies an indentation step but does not put any arguments at that level. But we do not allow `(0)` as a group specification since this would not be any different than plain 0 and probably means that the specification is confused in some way.

The keywords allowed at the beginning of an indent specification are:

:BREAK or **:NOBREAK** or **:FIT** -- These specify placement of the first argument in the first group. Default is **:FIT**, which means put this arg on the same line as the CAR of the form if it fits there in preferred mode, otherwise put it on the next line. Note that if the first arg goes on the same line as the CAR, its placement specifies the indentation level for the entire first group. That way long CARs will move the first group over to the right. (This makes the binding and termination of both DO and DO* line up, for example.) Specifying **:NOBREAK** means the first arg is forced to go on the same line as the CAR. Specifying **:BREAK** means the first arg is forced to go on the next line (and thus at the indentation level derived from the number of groups). UNWIND-PROTECT is a good example of using **:BREAK** to force the first arg onto its own line. Note that you can only specify one of **:NOBREAK**, **:BREAK**, or **:FIT**.

:TAGBODY -- Normally all forms in the body (whether atomic or not) go at the same indent level. Specifying **:TAGBODY** indicates that atomic body elements (*not* atomic elements of the argument groups) should be extended to line up with the CAR of the entire list (such as PROG or TAGBODY, which see for examples).

:STEP -- This can be specified as many times as desired and each time increases the indentation of the body (and thus all the argument groups) by one step. If you just want to move some of the groups in but not all of them (and not the body) then use 0 groups at the appropriate place instead of using **:STEP** at the beginning. **:STEP** is very useful with **:TAGBODY**.

By the way, the normal body indentation is taken from the INDENT-BASE field of the LISP edit environment, which is initialized to the width of a capital 'M' in the SEdit default font. The normal indentation step is taken from the INDENT-STEP field of the LISP edit environment, which is initialized to twice the width of a capital 'M' (that is, twice INDENT-BASE). These defaults are chosen so that, in a fixed-width font, the body of a form lines up two characters in from the '(' of the form, and each argument group line up two characters in from the next one (or the body). If you want non-standard values for either of these parameters, you can change the values in the LISP edit environment and then reinitialize your SEdit formats. Also, if you change font profiles, reinitializing SEdit will fix up the indents appropriately.

SEdit Internal Documentation

How Relinearization Works

The process by which SEdit optimizes formatting recomputation is strange and wonderful, so this is a long overdue attempt at explaining it.

We will start with a quick recap of SEdit's formatting model and the responsibilities of three node type methods: assign-format, compute-format-values, and linearize. We then describe the assumptions SEdit makes about when these have to be redone, and then describe the algorithm it uses to achieve this. We'll only go as far as getting the linear form fixed up; the step from there to updating the bits in the window is a whole 'nother story...

The formatting model

Linearization is the generation of a sequence of format tokens (space, string, bitmap, line break) from the internal tree representation of the data structure being edited. Doing a reasonable job for complex hierarchical structures involves a large number of constraints; SEdit uses a three pass algorithm to get its results:

formats: first, the presentation of a data structure often depends on the context in which it appears. for instance, a list occurring as the second element of a list beginning with let gets special treatment. another example is collapsing lists at a nesting level cut-off (actually, now that i think about it this would be much better done at parse time). SEdit currently does the first, but not the second (it ought to do both). to achieve this, each node can pass a 'format' to each of its subnodes. exactly what constitutes a format is arbitrary; it's up to the parent and child nodes to agree on what information will be passed (all current methods ignore format information they don't understand). at present lists pass their sublists list-format structures describing the appropriate presentation, and sometimes pass the atom :keyword to atomic elements which are to be printed in boldface.

each node type provides an assign-format method, which is called when the format of a node changes, and is responsible for propagating that change by calling assign-format on each of its subnodes; hence width estimates propagate from the top down.

width estimates: second, the presentation of a composite data structure will often depend on the size of its components. once the format information has been propagated to a node it will be asked to provide estimates of the size of its presentation, so that the nodes above it in the tree can plan their presentation intelligently. (note: unfortunately, most of the code calls width estimates "format values"; hence the method responsible is called compute-format-values). each node compute two numbers: inline-width and preferred-width. the inline width is the estimated width of this node's presentation assuming that there is room to fit it all on one line, or nil if the node's presentation will always span multiple lines. the preferred width is the width of the node's presentation assuming it will break at convenient spots. in computing these estimates a node needs access to the width estimates of its children; hence width estimates propagate from the bottom up.

linear form: finally, each node is asked to compute its linear form, by outputting a sequence of format tokens interspersed with the linear forms of its subnodes. the linearize method is told the horizontal position at which it is to begin, and the horizontal position of the right margin (which ought to try to stay within, but it's free to ignore). to get the best formatting linearization procedures generally have two formats: a preferred, reasonably-indented format and a tighter "miser" format, and choose the miser format whenever the width estimates indicate that the preferred format won't fit. each node makes this choice independently. the linear form is computed top down.

Incremental changes

The three-pass computation described above places (relatively) simple requirements on the methods, and suffices for a one-shot presentation. However, this is insufficient for SEdit; the presentation changes after each character typed, and repeating the above computation each time over the whole tree would

clearly be unacceptably expensive. Instead, SEdit tries to determine the regions of the tree whose presentation might have changed given the editing operations performed, and calls the presentation methods only when the results might be different.

formats: in determining the format for its subnodes, a node is only allowed to base its decisions on its type, its own format, and the structure of it and its immediate subnodes. thus a list can change the format of its subnodes if it is edited, or one of its immediate subnodes is edited, but not if a nested subnode is edited. thus we only need to rerun a node's assign-format method if

- it was edited, or
- one of its subnodes was edited, or
- the assign-format method of its supernode was run (for one of these three reasons) and it assigned a different format to this node than previously

width estimates: the width estimates of a node must be determined based on its structure, its format, and the width estimates of its subnodes. thus we only need to rerun a node's compute-format-values method if

- it was edited, or
- its format changed, or
- the compute-format-values method of one of its subnodes was run (for one of these three reasons) and it changed that node's width estimates

linear form: the linear form of a node also depends on its structure, its format, and the width estimates of its subnodes. in addition, it can depend on the space between its starting horizontal position and the right margin. thus we only need to rerun a node's linearize method if

- it was edited, or
- its format changed, or
- the width estimates of any of its subnodes changed, or
- changes to one of its supernodes has caused its presentation to begin at a different horizontal position relative to the right margin

Relinearization

to implement the optimizations suggested above, SEdit must first know what parts of the tree have changed since it was last presented. therefore all procedures which change the tree structure are responsible for calling note-change on any node they change. also, all nodes added to the tree are marked as needing re-presentation. note-change inserts the changed node into a queue (the changed-nodes field in the edit-context) which is kept sorted by increasing depth. relinearize-where-necessary then implements the following algorithm:

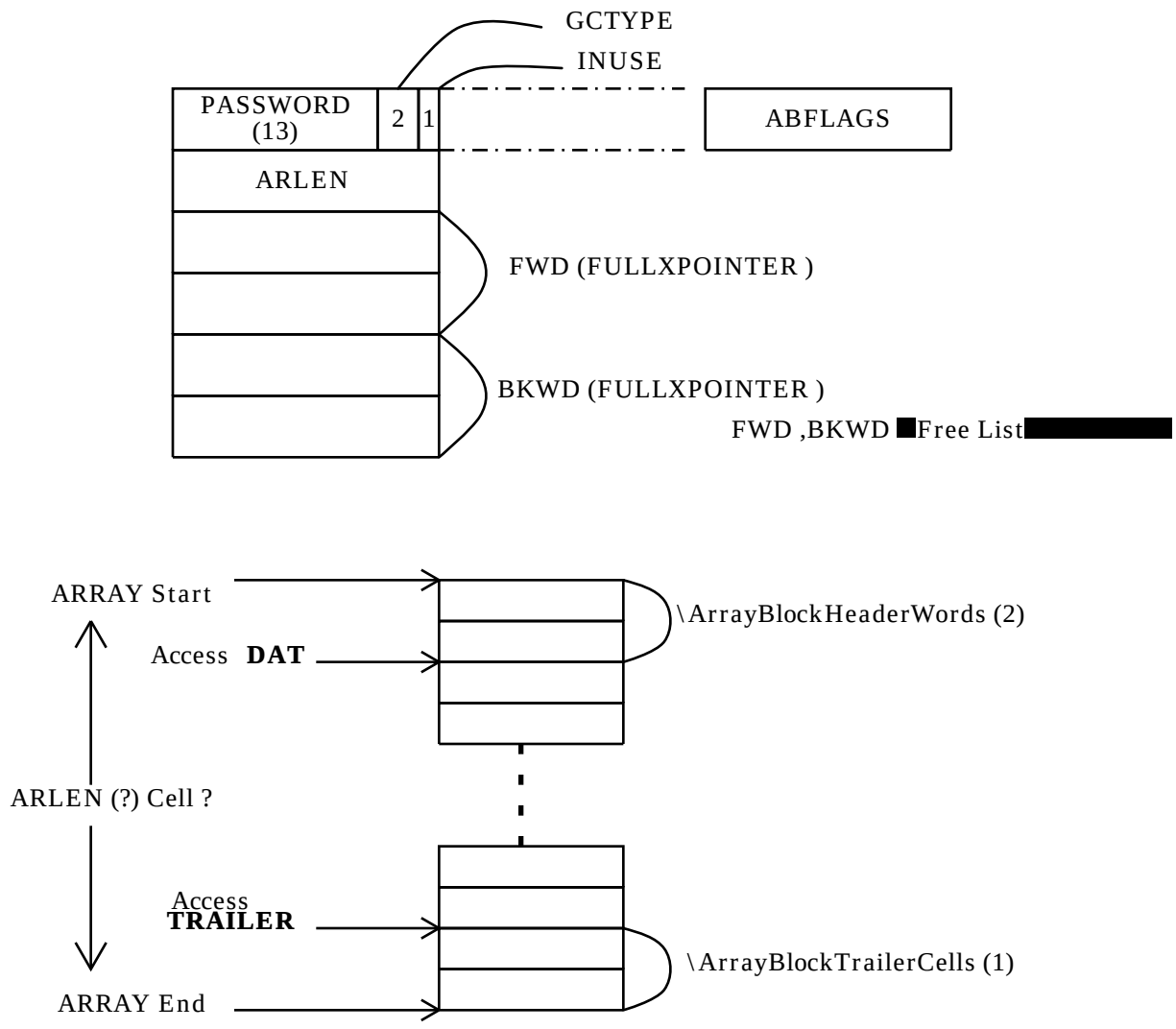
[inconsistency: note-change actually maintains the queue in order of decreasing depth (despite its comments to the contrary), and relinearize-where-necessary reverses it before it looks at it. this turns out to be fine, since it wants the queue in decreasing depth for the next step. ought to fix the comments though...]

1. for each node on the queue (from top to bottom), assign-format to its super node and add it to the queue (unless it's already in the queue) and then assign-format to it.
2. for each node now in the queue (from bottom to top) recompute its width estimates, and if they've changed push its super on the queue (where it will be picked up later in this step); if they don't change this is a point to start relinearizing from so push it onto another queue.
3. finally, reverse the new queue created in step 2 (so that it's now ordered from bottom to top) and for each node on it check to see that none of its super nodes are awaiting relinearization — if one is found, mark all the nodes between them changed so that the super's relinearization (yet to come) will include this node — otherwise just relinearize it.

relinearizing a node is guaranteed to call its linearize method. In addition, it will call the linearize method of any subnode which (a) has changed, or (b) has been moved (relative to the right margin) so

that it's linear form is no longer likely to be appropriate (the test for this is at the beginning of generate-linear-form) (and so on recursively into its subnodes). (and as i mentioned before, there's a whole extra story about how changes to the linear form are used to determine changes to the screen; this is yet to be documented at all). when relinearization of the original node is complete, an additional test is made: if the last line of the new linear form is a different length than the last line of the old linear form, this relinearization continues with the linear form of the super node, starting after the node just linearized (since the super may have made linearization decisions based on where that node ended). this process terminates when (a) a node's new linear form ends at the same horizontal position as before, or (b) the top of the tree is reached.

Structure of ARRAYBLOCK



□ □ □ □ □



COMP.STTAG

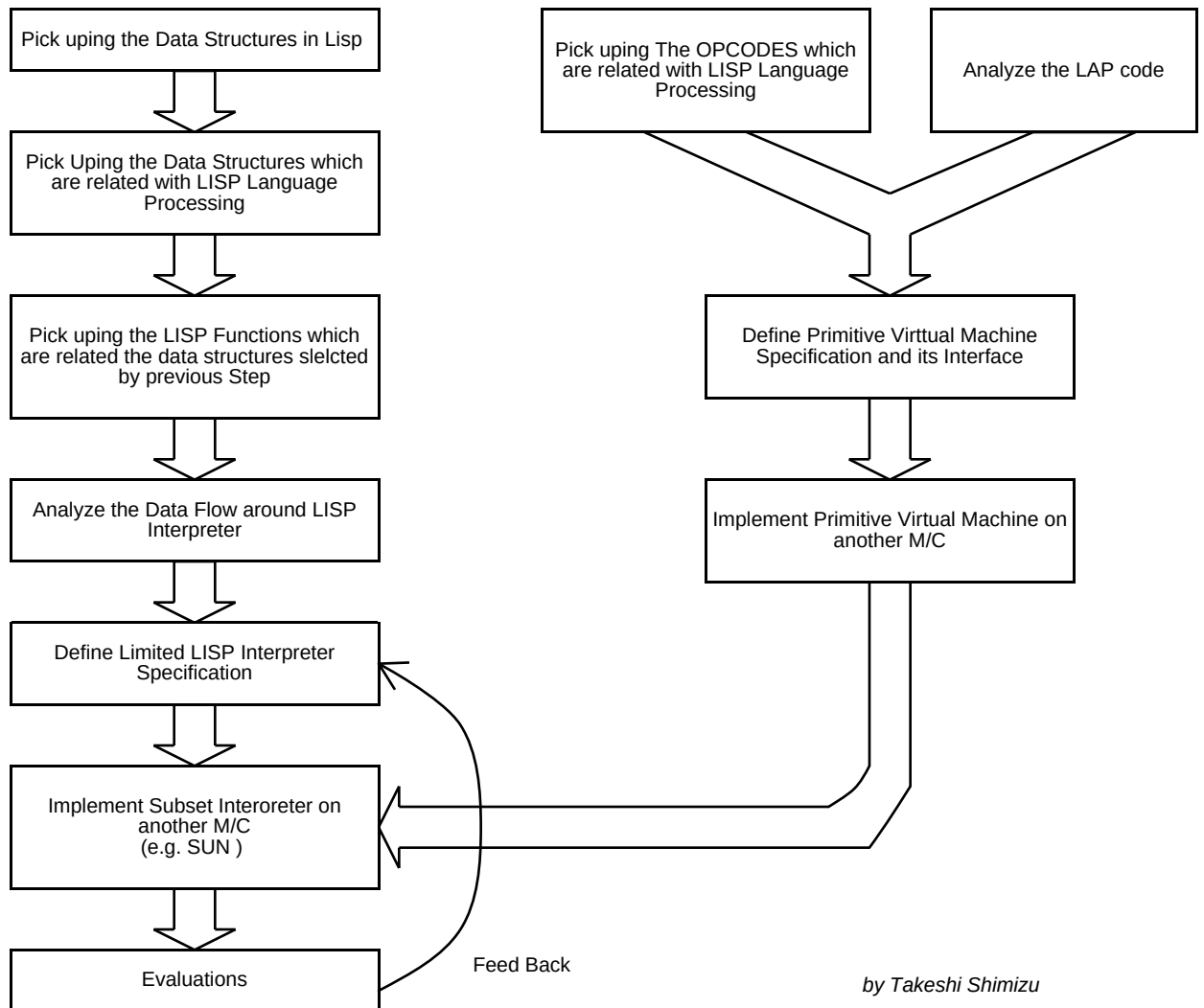
(NLV _ (fetch (TAG LEVEL) of TAG))
(NF _ (fetch (TAG FRAME) of TAG))

(OR NLV NF)

LEVEL !=NIL



No .		Func. Name		File Name	
A B S T R U C T					
	Type & Name		Comments		
A R G S					
R E T V A L					
L O C A L S					
E X T E R N S					



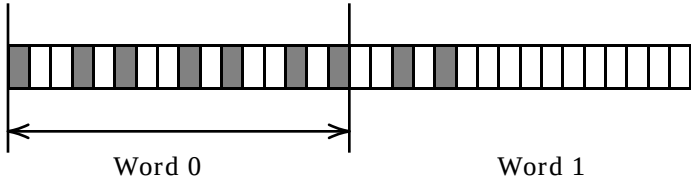
Interlisp-D Machine Development Concept (Primary Step)



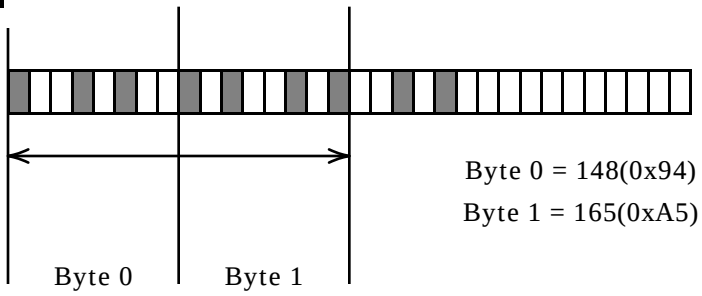
Interlisp  BITMAP 

BitMap 

 = 0  = 1

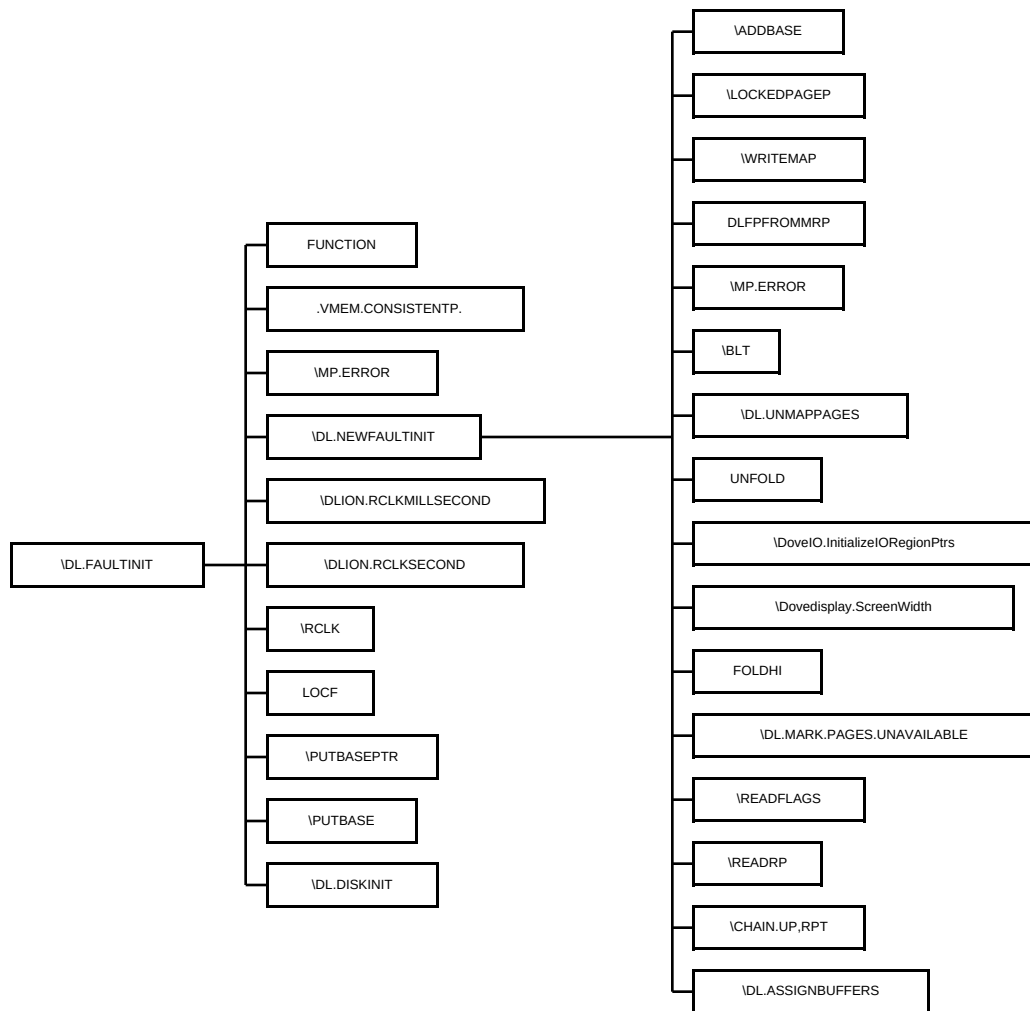


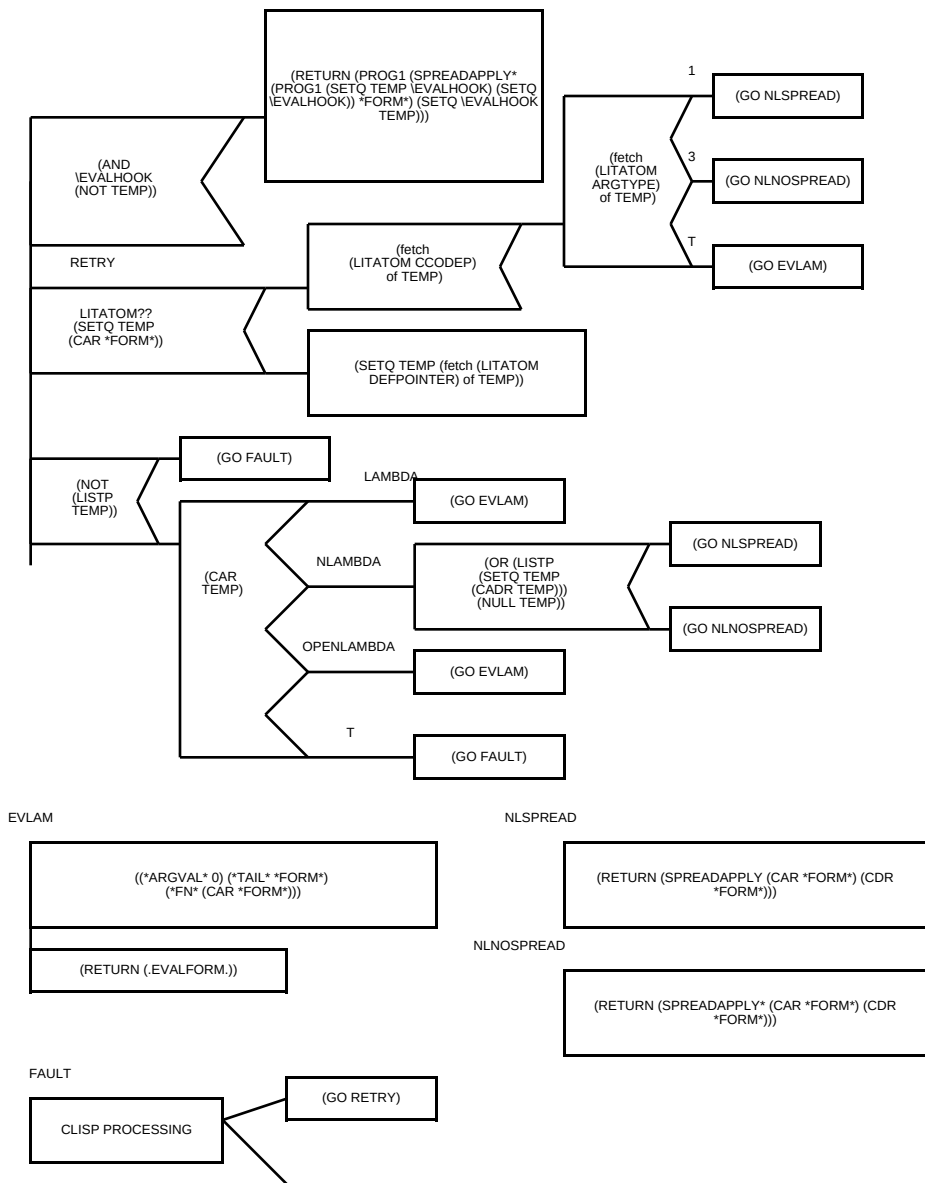
Word Access = 38053(0x94A5)

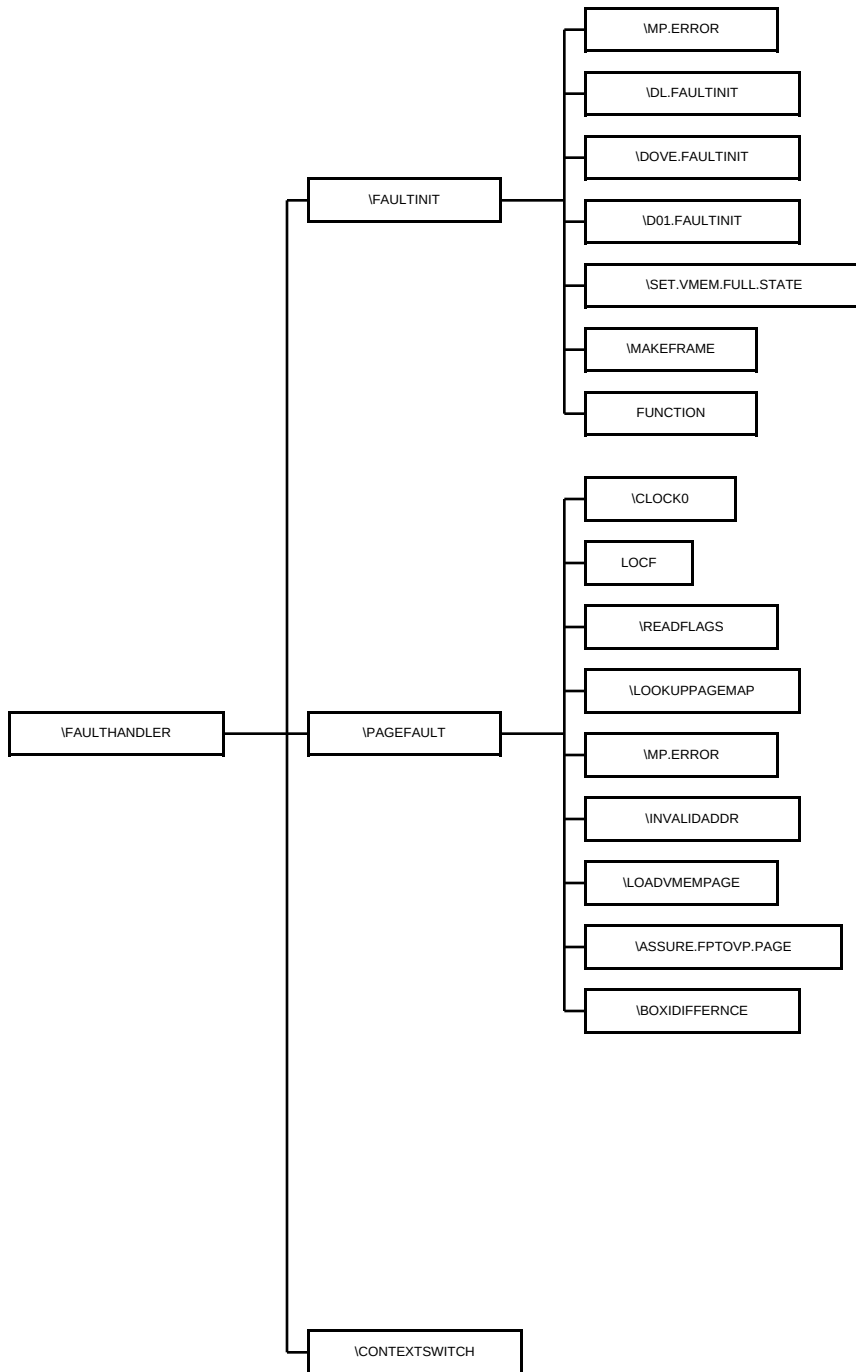


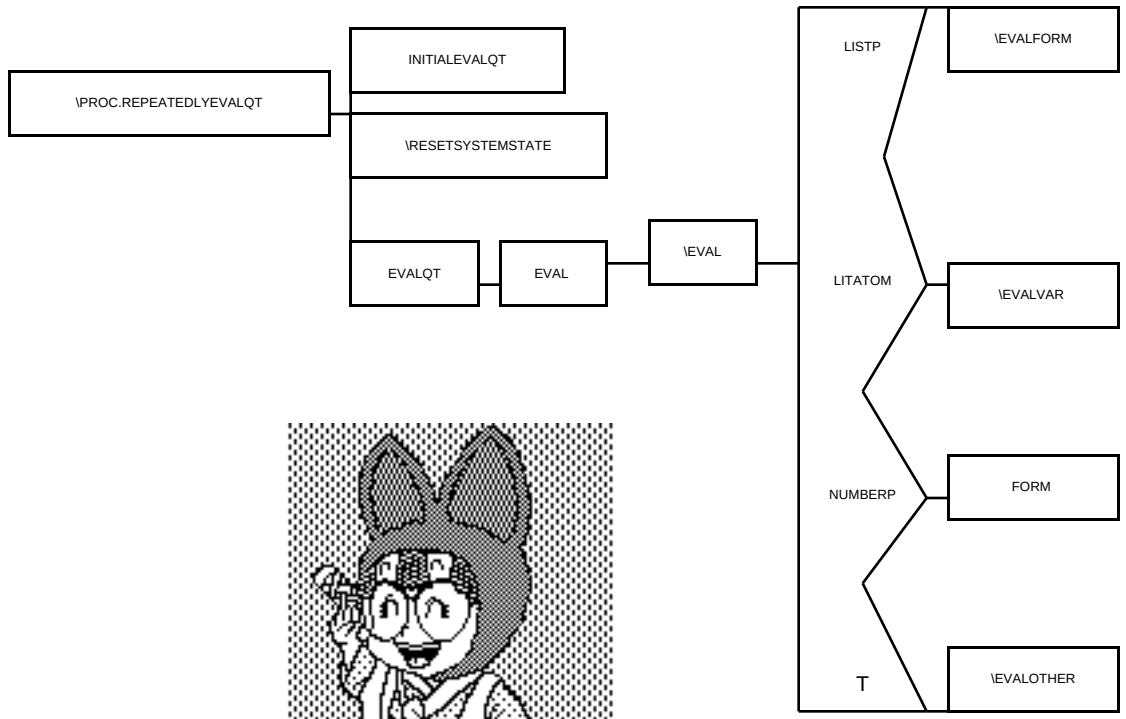
MSB

LSB







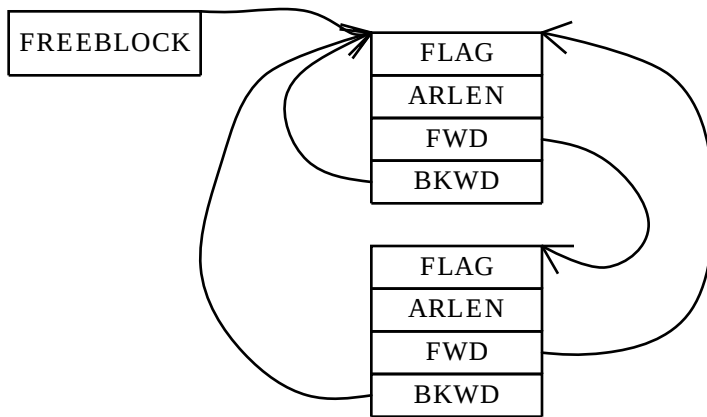


\LINKBLOCK

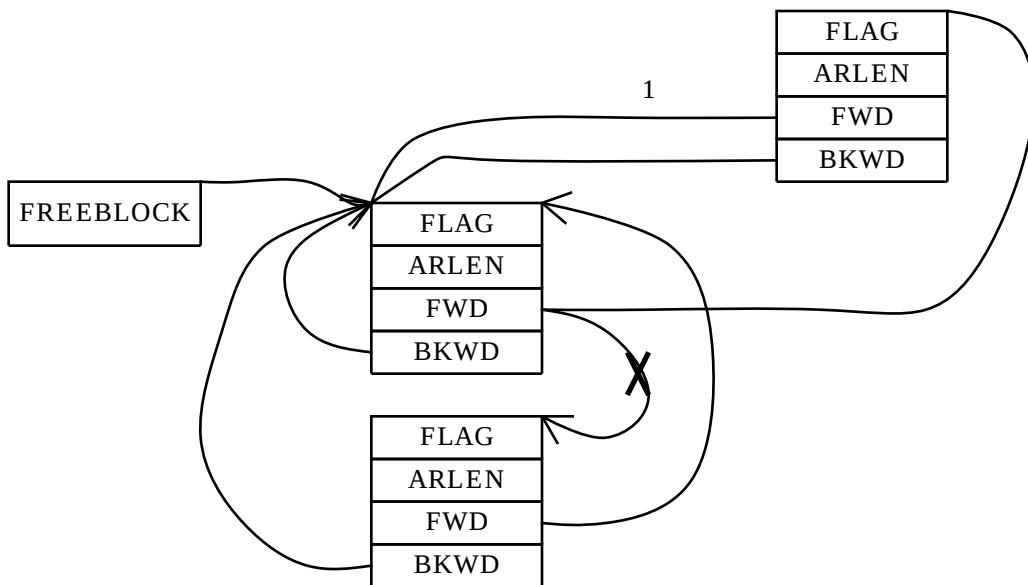
Case 1 (Record)

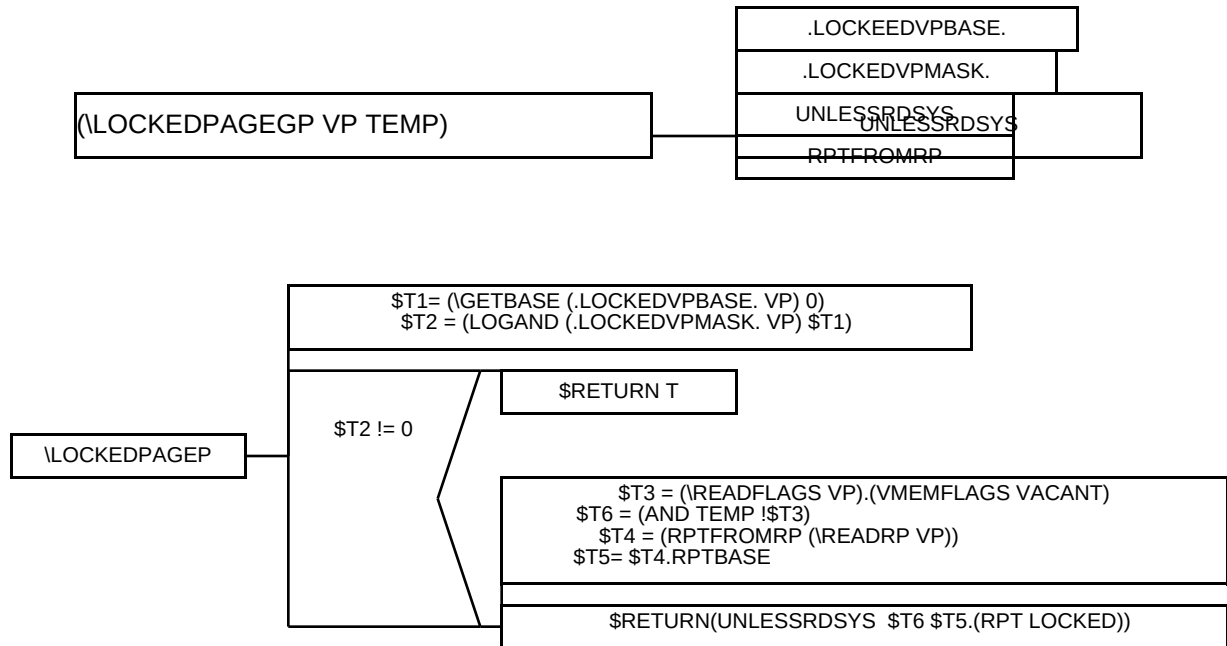


Case 2 LINK



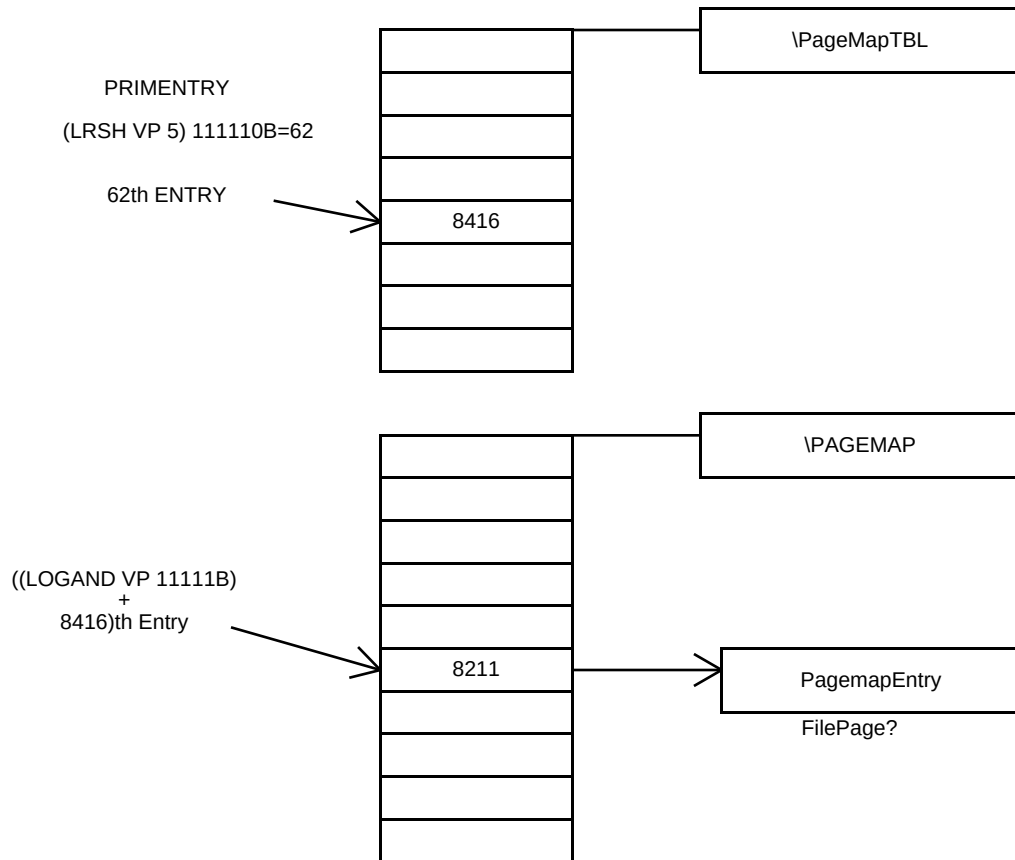
X



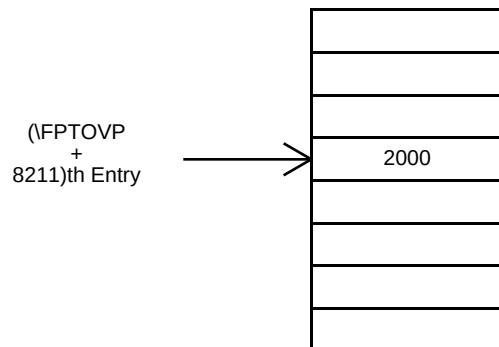


\LOOKUPPAGEMAP

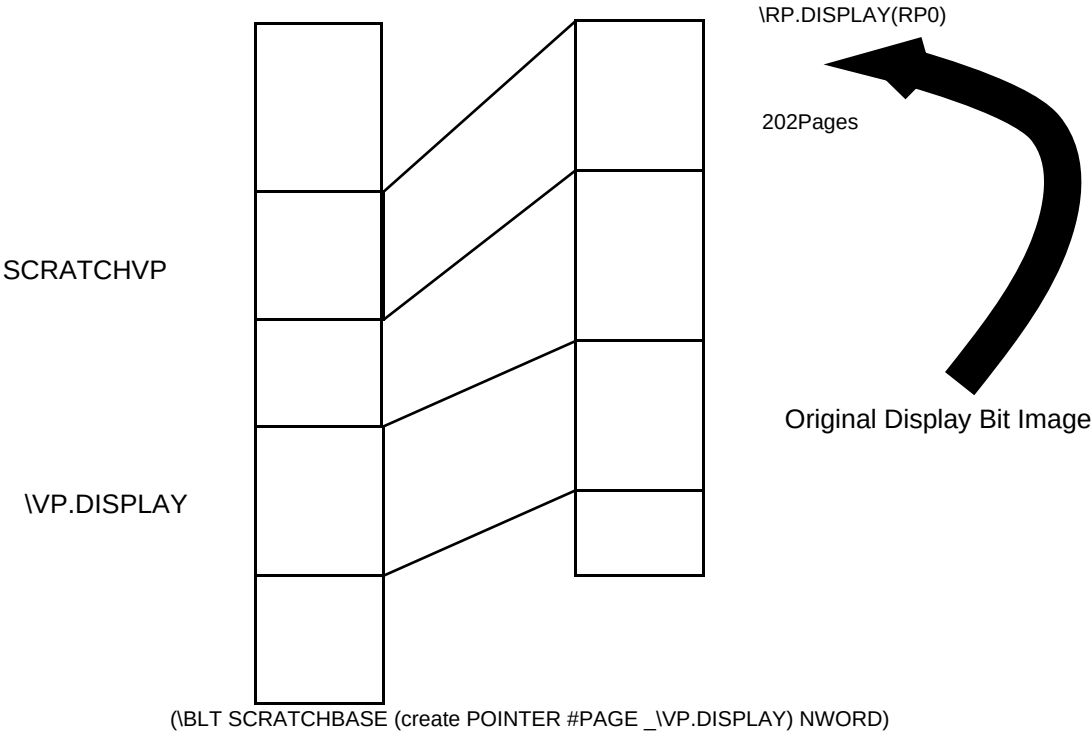
VP= 11111010000B=2000



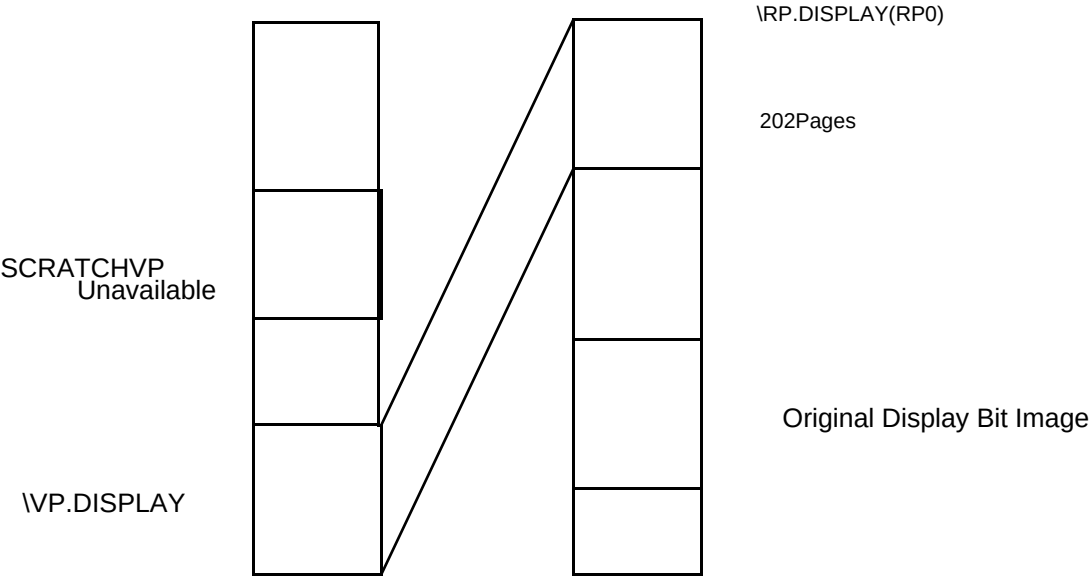
Operations by Using \FPTOVP

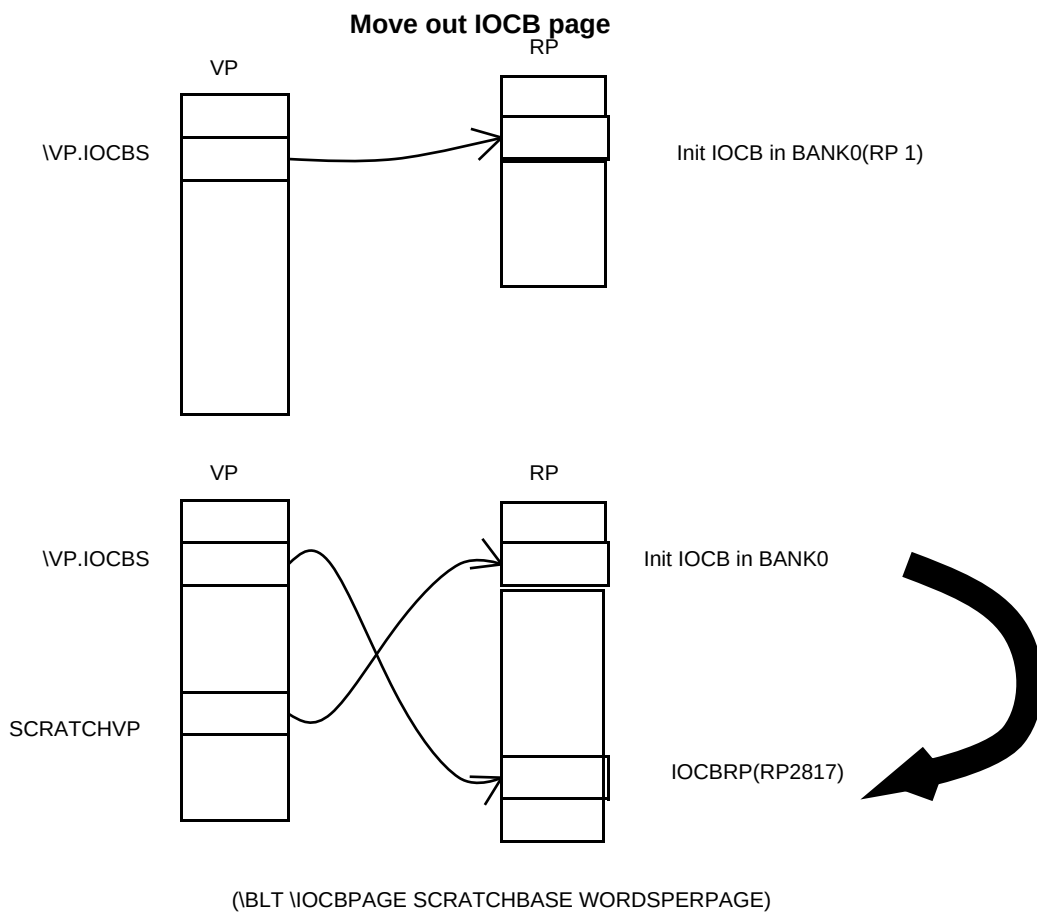


MOVE Display Region First



MOVE Display Region Second





2 bit
TAG

30 bit pointer

		CAR

00 : CDR is NIL
01 : CDR is PREV
10 : CDR is NEXT
11 : CDR INDIRECT

Next CONSPAGE ptr		
Next CELL ptr		
COUNT		

1000	1	1	1024
1004			
1008			
1016			
1020			
1024	1	0	CAR ptr
1028			CDR ptr
1032			

-X-
CAR
CDR
LISTP
NTYPX
TYPEP
DTEST
CDDR
PRLPTR.N
GCREF
ASSOC
RPLACA
RPLACD
CONS
FMEMB
LIST1

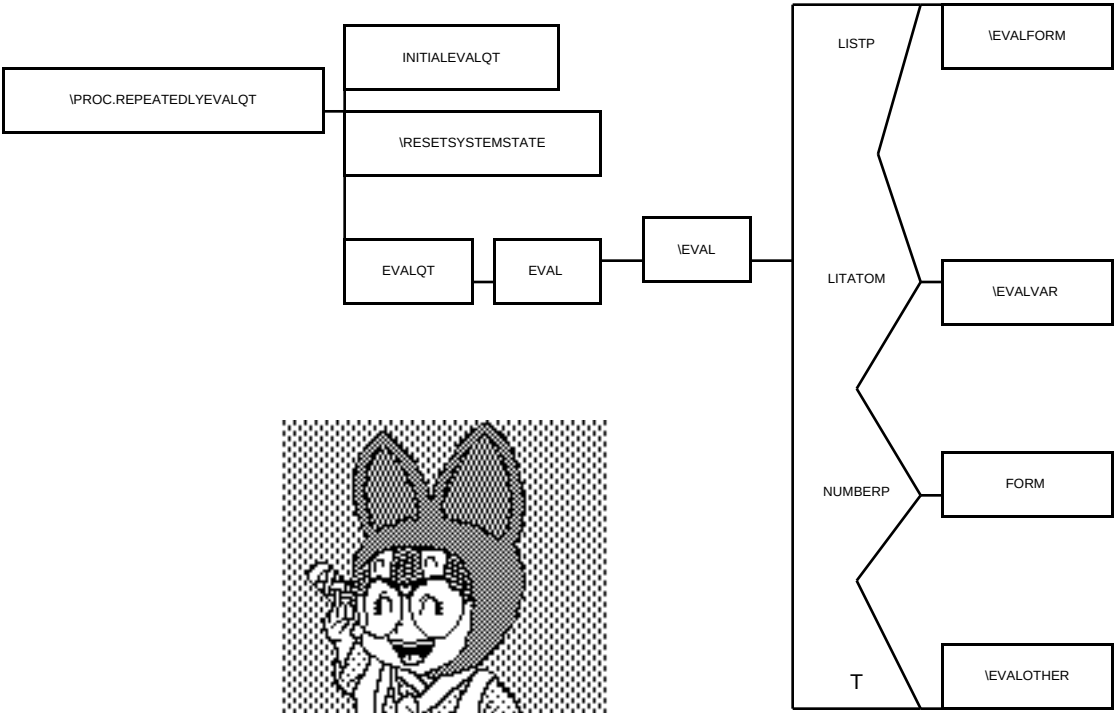
GETHASH
PUTHASH
CREATECELL

BIN
BOUT

FN0
FN1
FN2
FN3
FN4
FNX
APPLYFN
RETURN

CHECKAPPLY*

BIND
UNBIND
DUNBIND
GVAR_



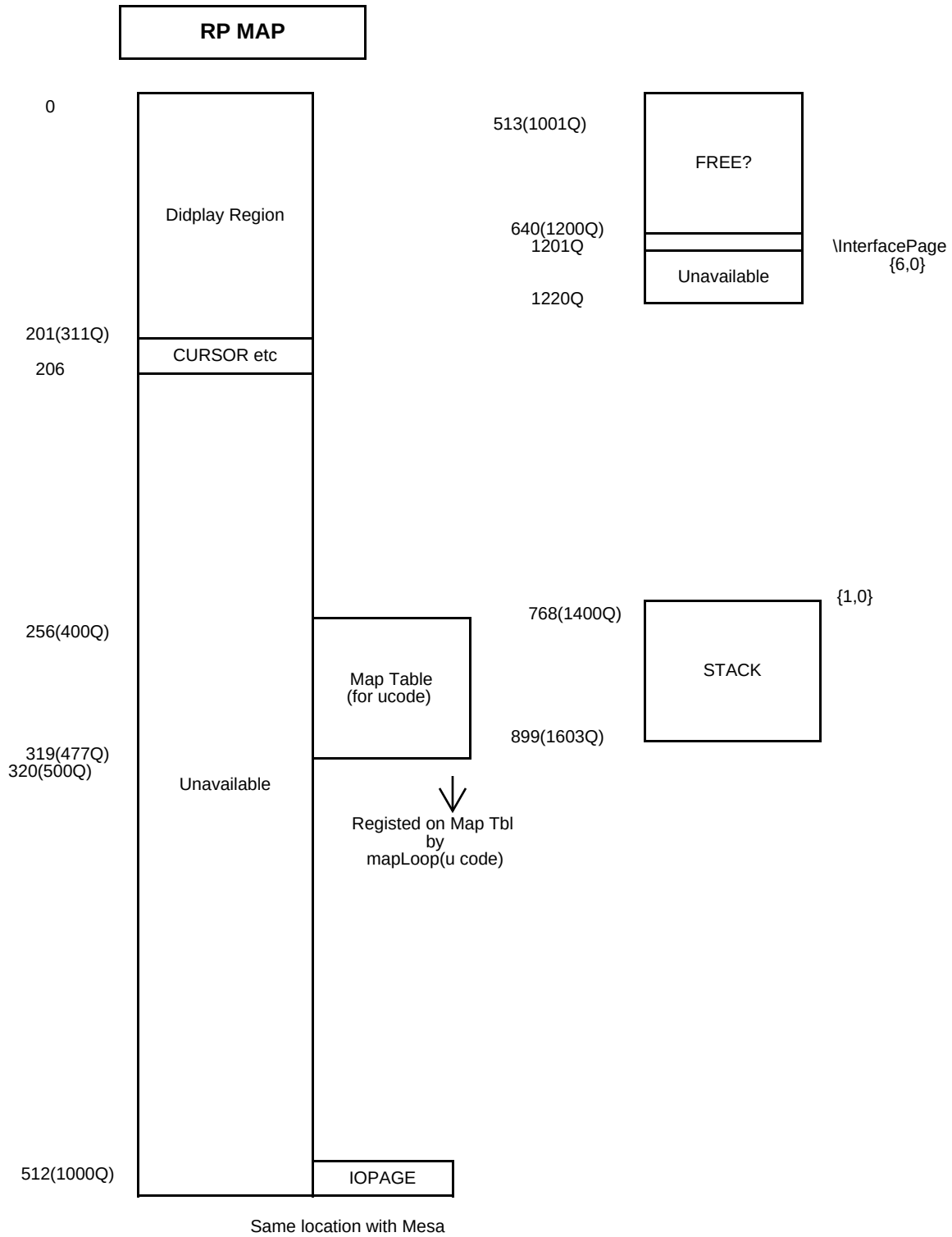
\RESETSTACK0

300H	
	\FAULTHANDLER
600H	
	\DOHARDRETURN
900H	
	\DOTELERAID
C00H	
	\KEYHANDLER
F00H	
	\DOGC
1200H	
	\DOSTACKOVERFLOW
1500H	
	\DOMISCAPPLY
1800H	
	\CODEFORTFRAME

\InterfacePage.StackBase

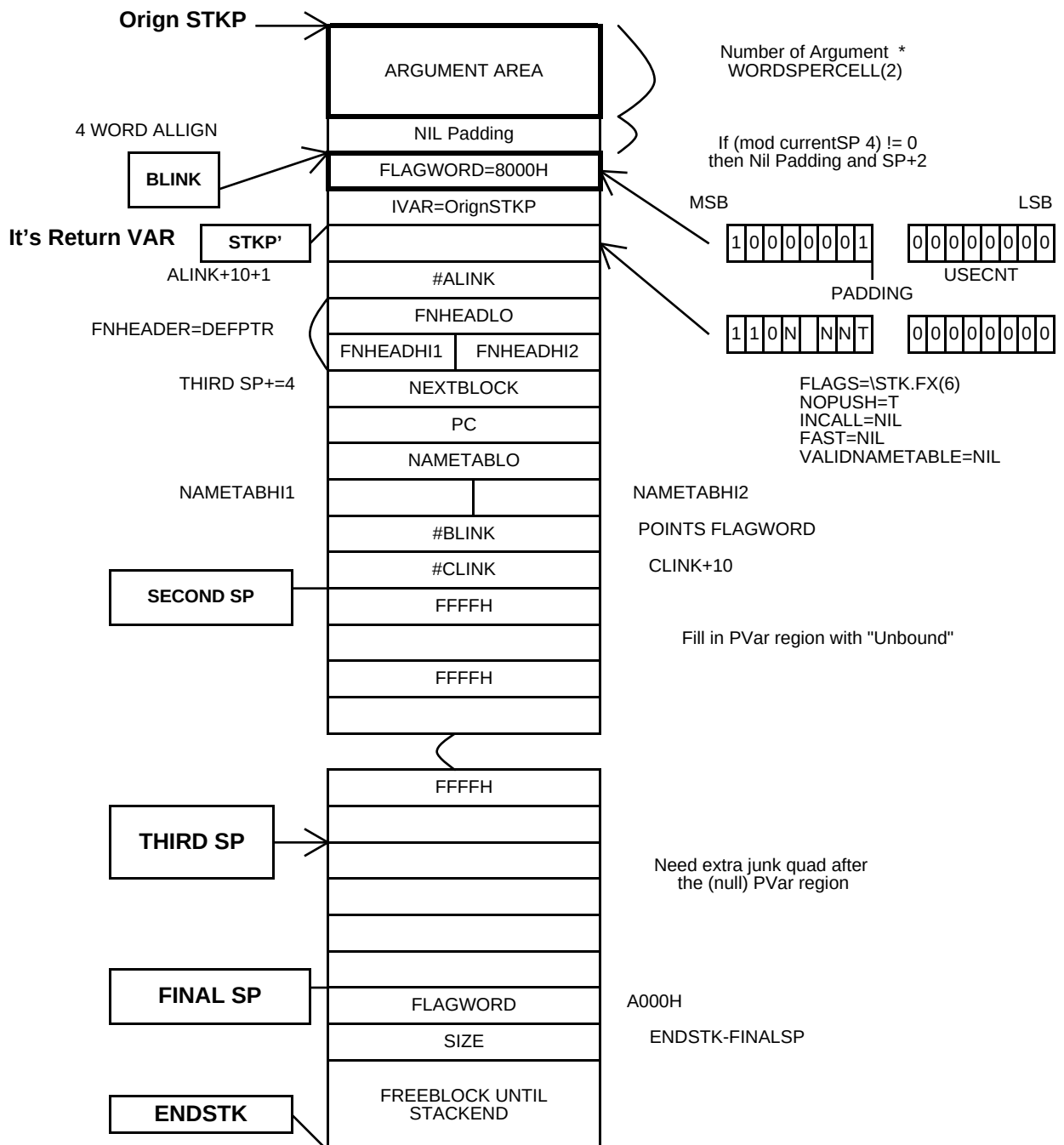
\InterfacePage.EndOfStack

\InterfacePage.FAULTFXP =302H
\InterfacePage.HardReturnFXP =602H
\InterfacePage.TELERAIDFXP =902H
\InterfacePage.KbdFXP =C02H
\InterfacePage.GCFXP =F02H
\InterfacePage.SubovFXP =1202H
\InterfacePage.MiscFXP =1502H
\InterfacePage.ResetFXP =1802H



1024(2000Q)	FPTOVP	{4,45}	4096(10000Q)	UnavailableREALPAGETABLE	
1061(2045Q)					
			4352(10400Q)		
1289(2400Q)	TYPETABLE GCTABLE GCOVERFLOW	{6,200}			
1408(2600Q)		{6,377}			
1536(3000Q)		{20,0}			
1780(3364Q)	\PagemapTBL	{6,2}	4225 2046 {4,46}	2045	Locked Dirty
1788(3374Q)		{6,11}	3346 2447 {5,47}	4060	Locked Dirty
	\PAGEMAP	{5,0}	4421 2052 {4,52}	2051	Locked Dirty
1799(3407Q)		{5,13}	4501 2453 {5,53}	4064	Locked Dirty
1815(3427Q)	{6,160}-{6,177}		5450 2454 {5,54}	4065	Locked Dirty
	{6,12}-{6,15}		11553 2450 {5,50}	4061	Locked Dirty
	{6,20}		11603 2047 {4,47}	2046	Locked Dirty
1820(3434Q)	\DEFSpace		13716 2050 {4,50}	2047	Locked Dirty
1821(3435Q)			13762 2451 {5,51}	4062	Locked Dirty
1832(3450Q)	\VALSPACE		15531 2051 {4,51}	2050	Locked Dirty
1840(3460Q)			15627 2452 {5,52}	4063	Locked Dirty
1841(3461Q)					
1952(3640Q)	{23,56}-{23,340}	{21,0}\HTCOLL			
	{24,44}-{24,50}, {24,53}-{24,56}	{24,75}-{24,123},{24,373}			
2081(4041Q)		{5,27}-{5,46}			
2096(4060Q)					
2817(5401Q)	Unavailable				

\IOCBPAGE(Depends on MEM size)



FUNC NAME	\LOOKUPPAGEMAP	ID	

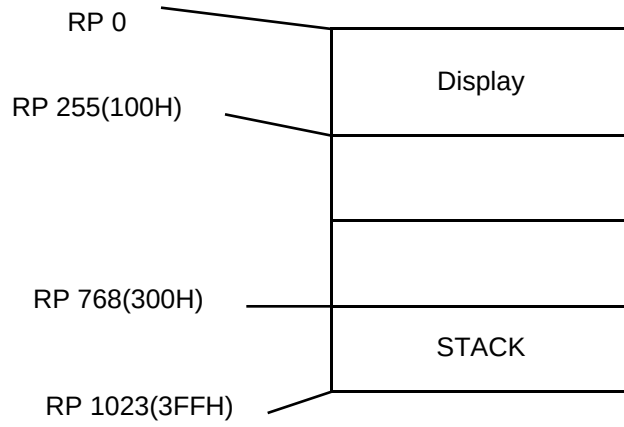
CALLING SEQUENCE	
(\LOOKUUPPAGEMAP VP)	

LOCAL VAL	
PRIMENTRY ** (\GETBASE \PageMapTBL (fetch (VP PRIMARYKEY) of VP]	

GLOOVAL VAL	
NAME	TYPE
\PageMapTBL (R) \EMPTYPMENTRY (R) \PAGEMAP (R) VP (R)	RECORD VP

CALL FUNCTIONS	
\GETBASE	

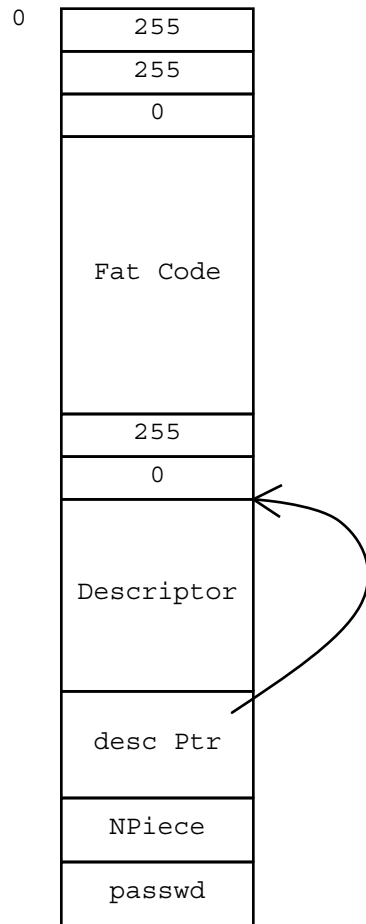
SPECIALRP

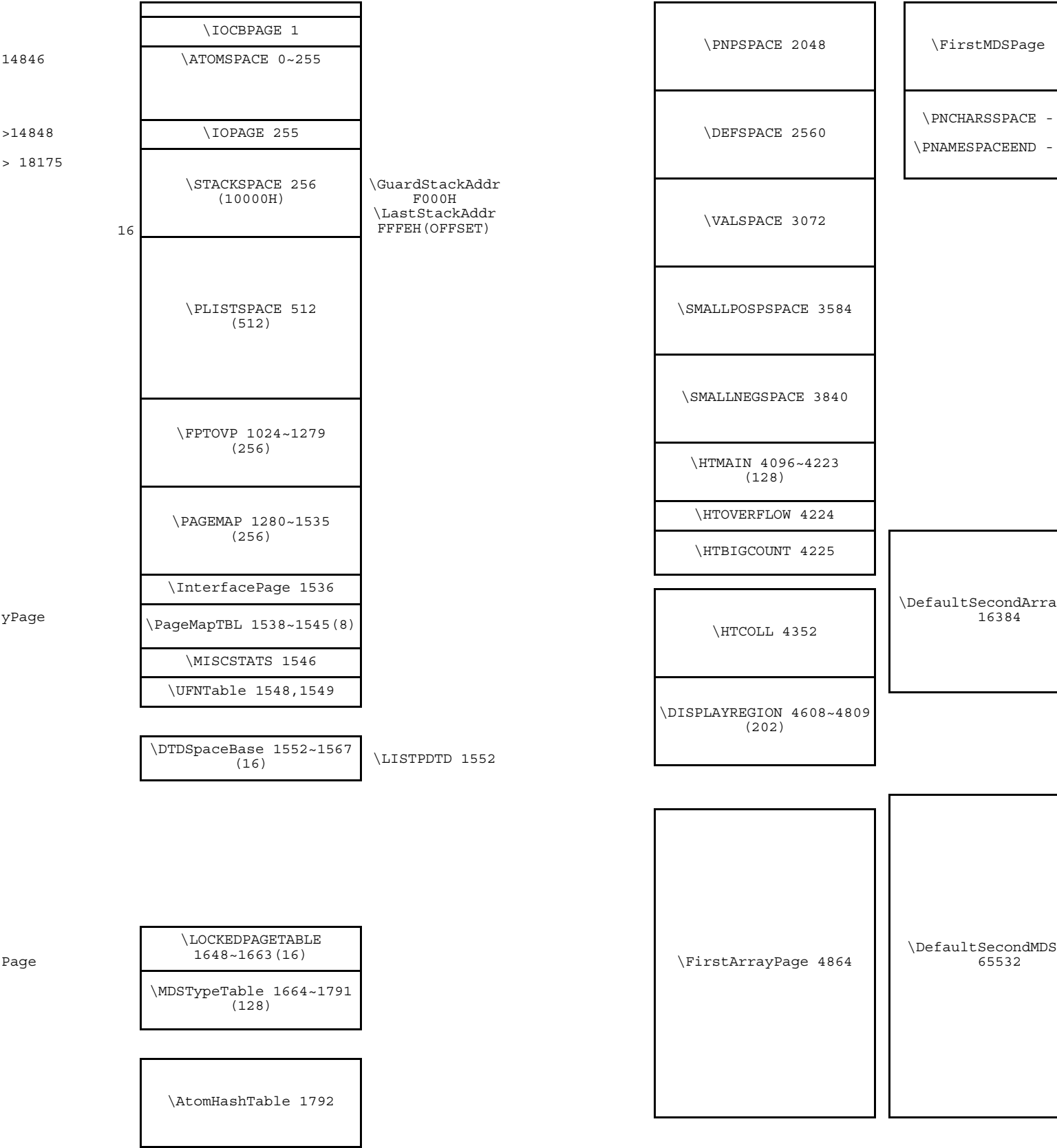


Memory Stats on STARTUP

0,100H	\IOCBPAGE
0,300H -0,23FEH	?STACKAREA?
0,FF00H	\IOPAGE
4,0000H	\FPTOVP
5,000H	\PAGEMAP 256 Pages
6,0000H	
6,0200H	\PageMapTBL
6,0A00H	\MISCSTATS
6,0C00H	\UFNTable
7,0000H	AtomHashTable
AH,0000	\VMEMPAGEP
12H,0000	\DISPLAYREGION







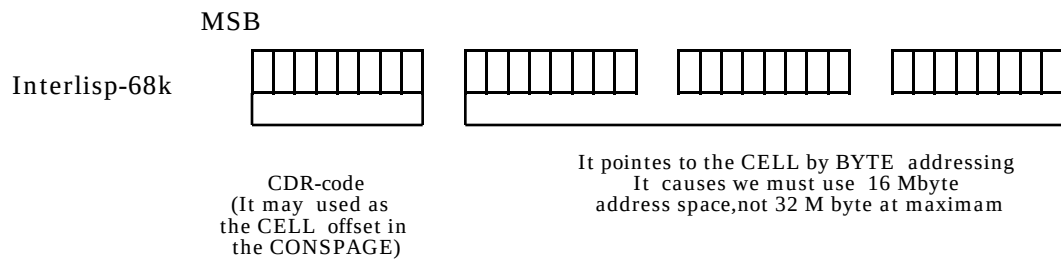
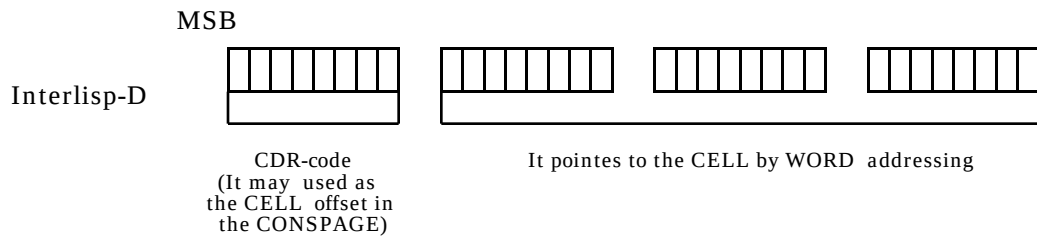
0~124	34820	14994~16382	MDS	64934~65536	MDS
126~1790	NOREF			65538	31315
1792~2046	32777	16384~17158	0		
2048~3582	NOREF	17160~64932	NOREF		
3584~4094	47105				
4096~4862	NOREF				
4864~4922	0				
4924~5886	NOREF				
5588~10036	0				
10038~10064	14338				
10066~10198	MDS				
10200~10234	6147				
10236~10256	MDS				
10258~10278	6147				
10280~10384	MDS				
10384~10388	14338				
10390~14846	MDS				
14848~14992	NOREF				

VMEM TYPE TABLE ON DLion
with 24000 pages
VMEMSIZE approximatery 16300

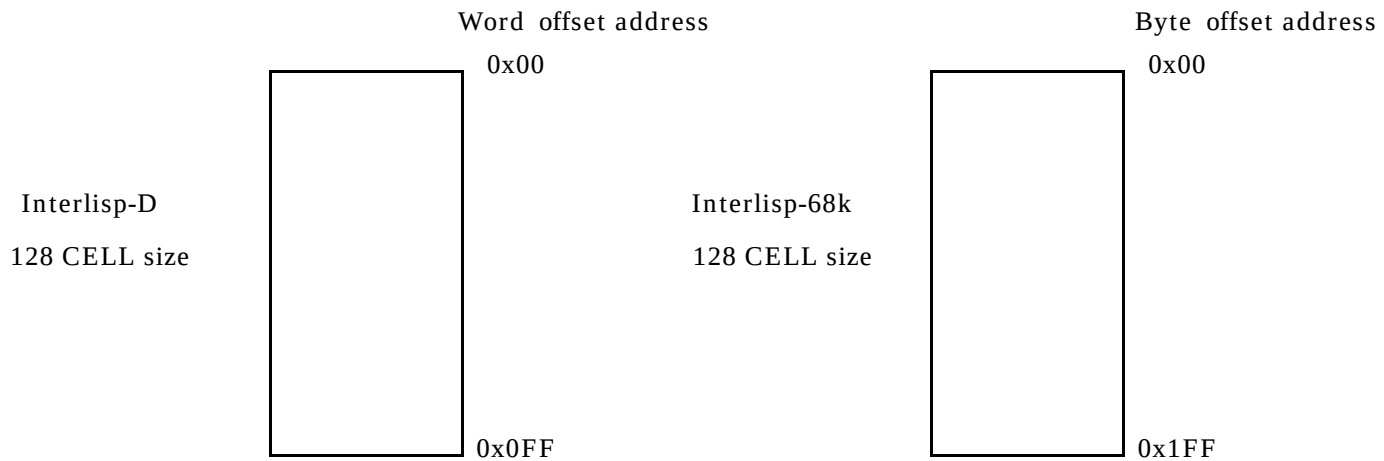
Spec of \ATOMSPACE

0	NIL ptr	\PTRHUNK16
1	NOBIND ptr	\PTRHUNK24
	ASCII code from 0 to 47, and 58 - 255	\PTRHUNK32
		\PTRHUNK42
		\PTRHUNK64
		\UNBOXEDHUNK1
367Q		
370Q	\EVALFORM	
	\GC.HANDLEOVERFLOW	
	\DTEST.UFN	
	\OVERFLOWMAKENUMBER	
	\MAKENUMBER	
	\SETGLOBAL.UFN	
	\SETFVAR.UFN	
	\GCMAPTABLE	
400Q	\INTERPRETER	
	SMALLP	
	FIXP	
	FLOATP	
	LITATOM	
	LISTP	
	ARRAYP	
	STRINGP	
	STACKP	
	CHARACTER	
	VMEMPAGEP	
	STREAM	
	BITMAP	
	COMPILED-CLOSURE	
	ONED-ARRAY	
	TWOD-ARRAY	
	GENERAL-ARRAY	
	\PTRHUNK2	
	\PTRHUNK4	
	\PTRHUNK5	
	\PTRHUNK6	
	\PTRHUNK8	
	\PTRHUNK10	
	\PTRHUNK12	

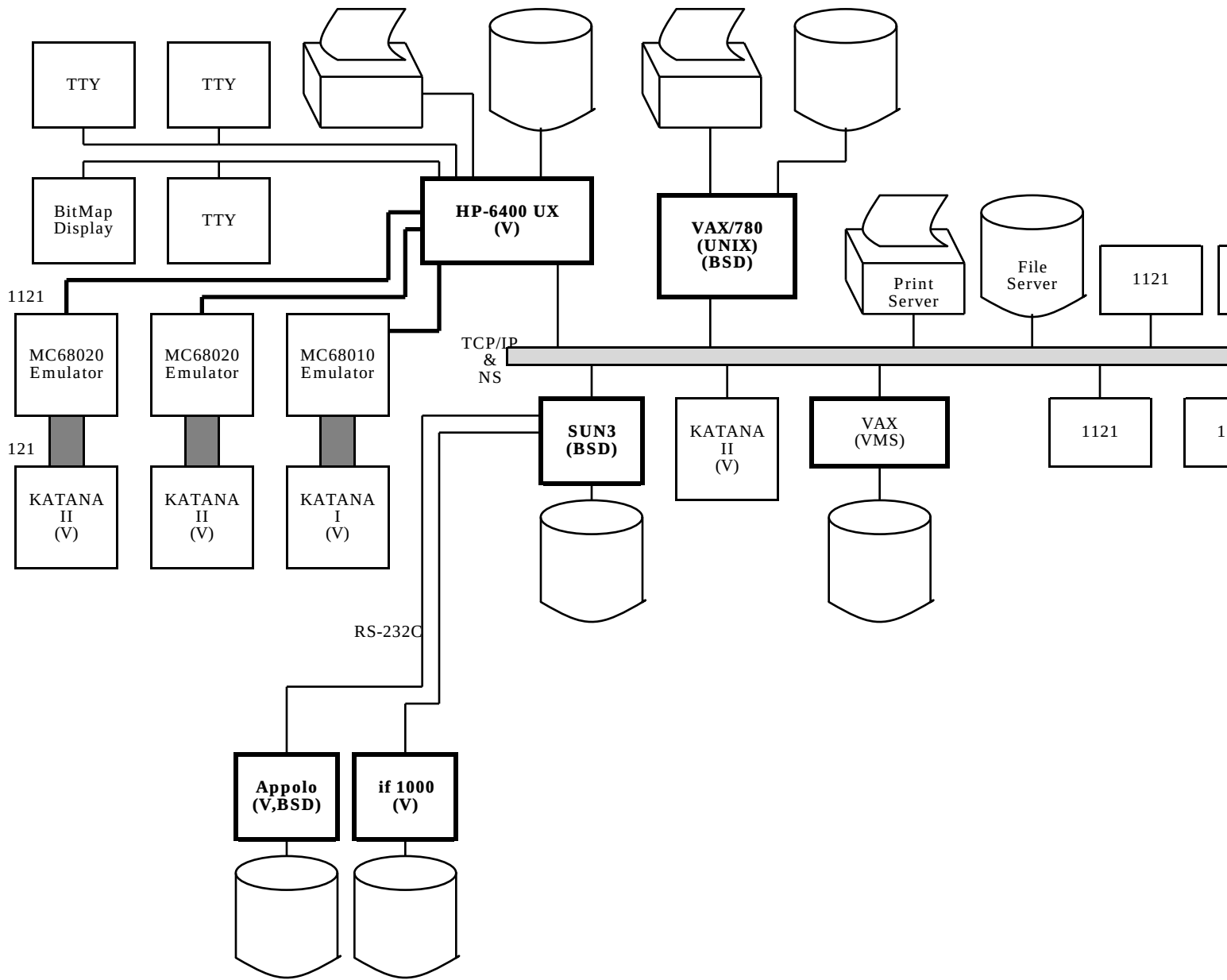
* CELL



* CONSPAGE

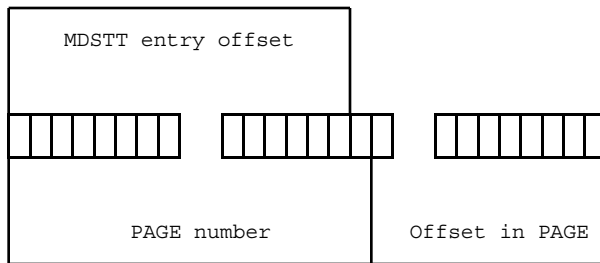


: 1 Cell 32 bit width, and !CONSPAGE includes 128 Cells



Data Type recognition on KATANA

by Takeshi Shimizu

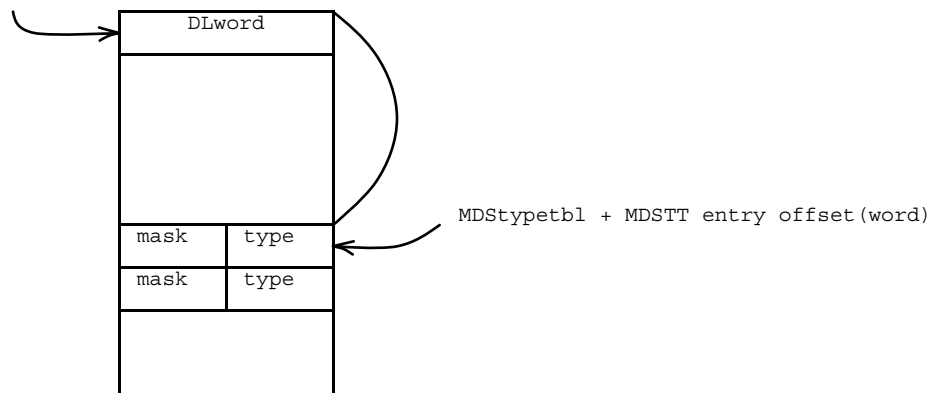


1 Page = 512 byte = 9 bit = 0x1FF

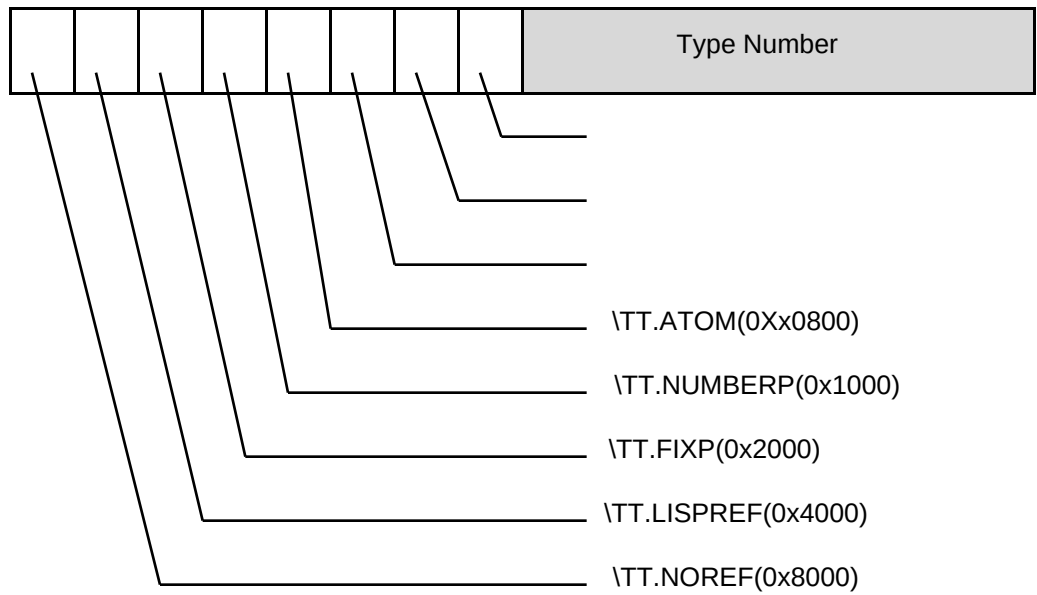
1 MDS entruy explains the type of 2 pages -- 10 bit

Method of accessing to MDSTT

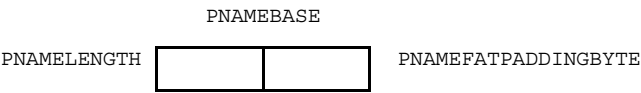
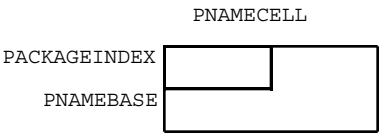
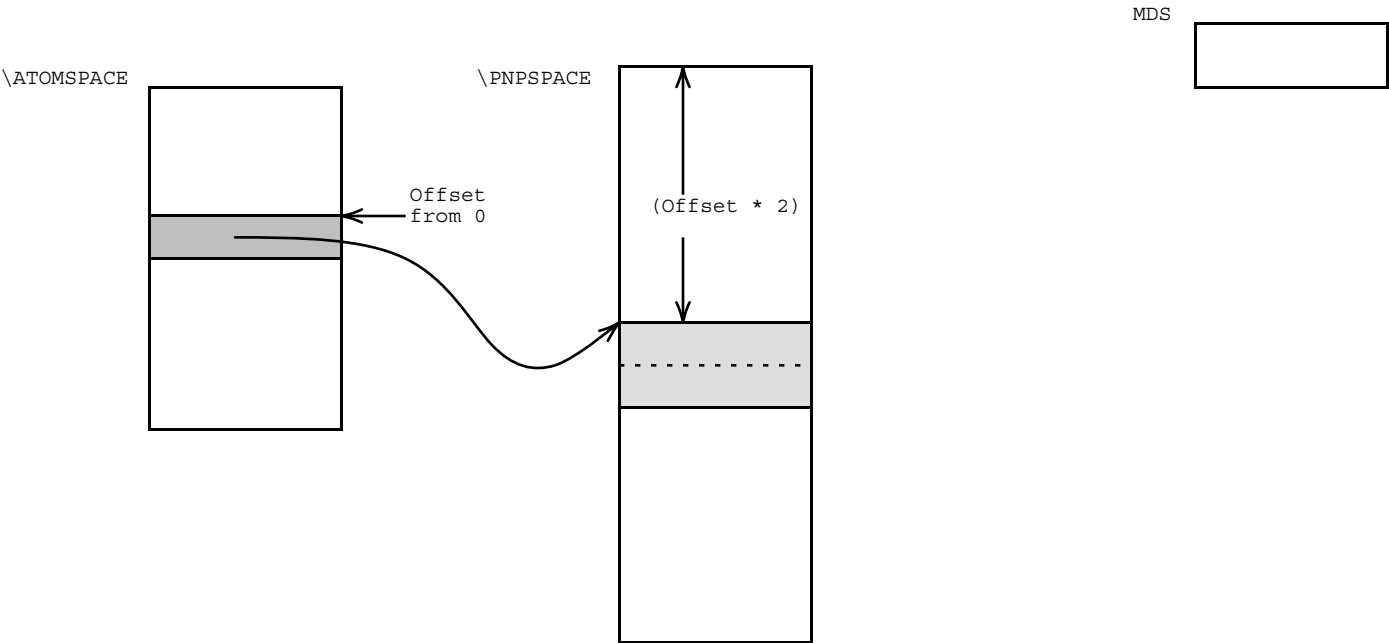
DLword *MDStypetbl



MDSTtpeTable Entry

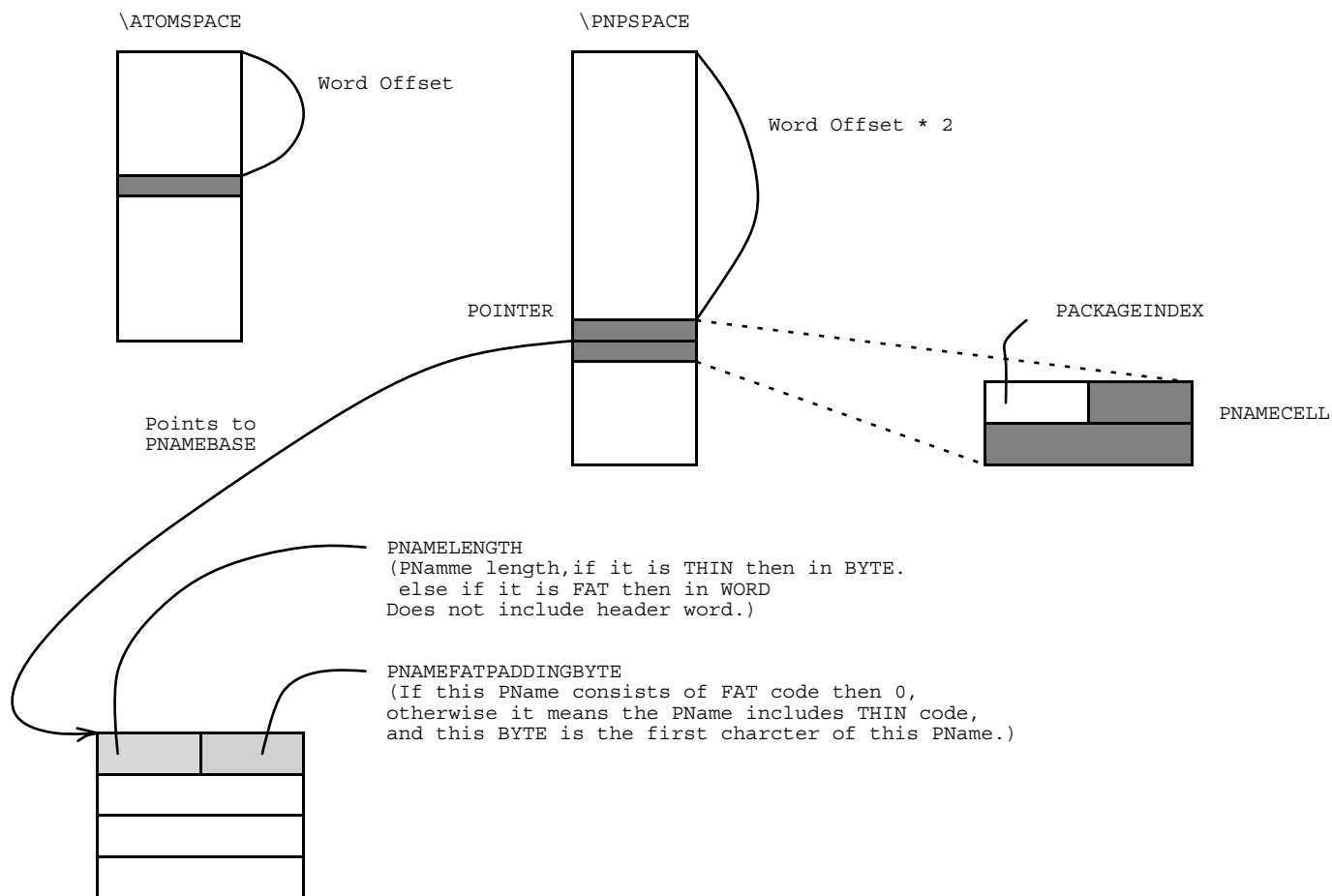


How to Access PName



How to Access PName

by Takeshi Shimizuu



Example: NIL

Its index number is 0.
And, Its Length is 3 bytes.

3	N
I	L

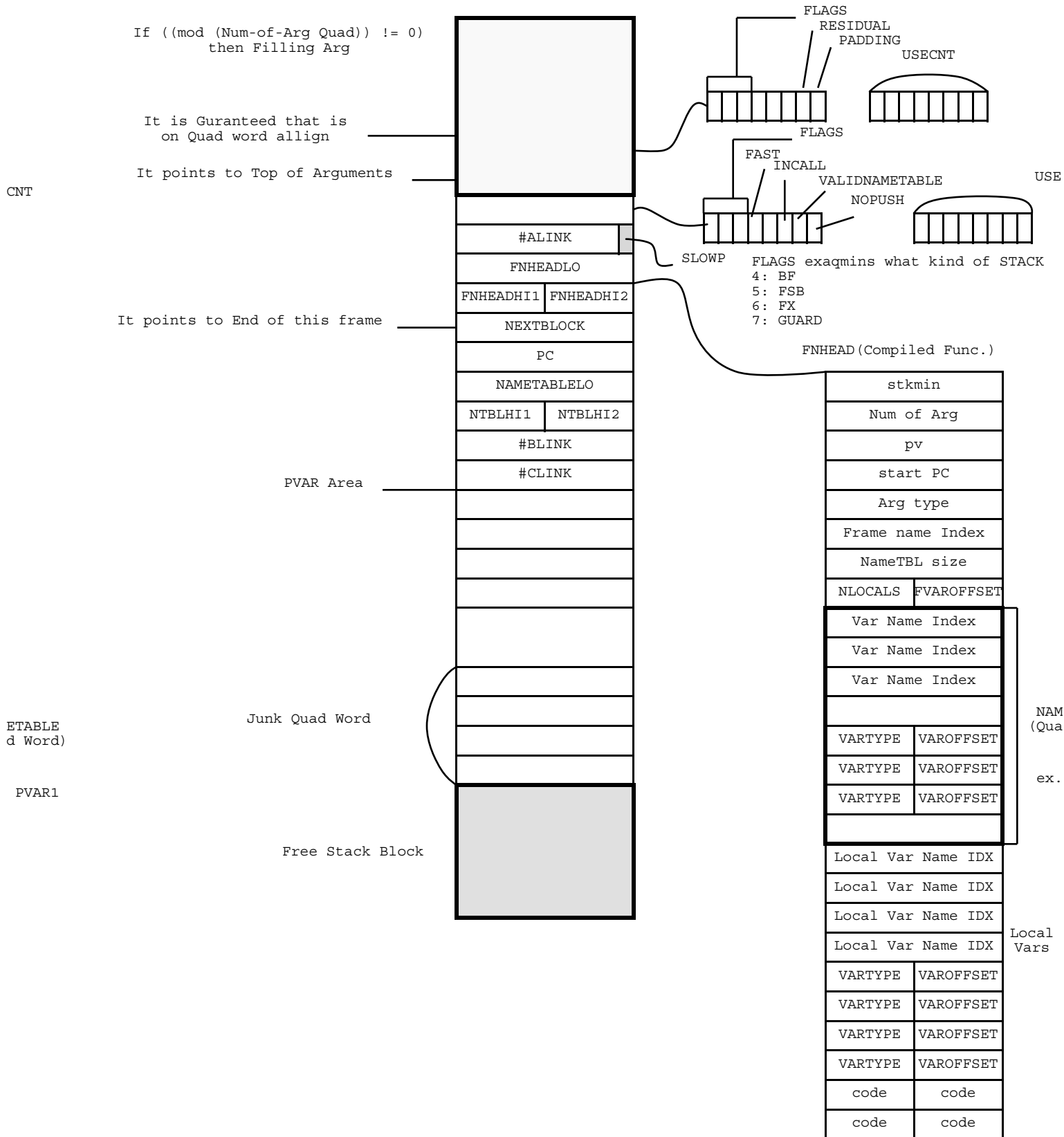
Example

Example: ■■■

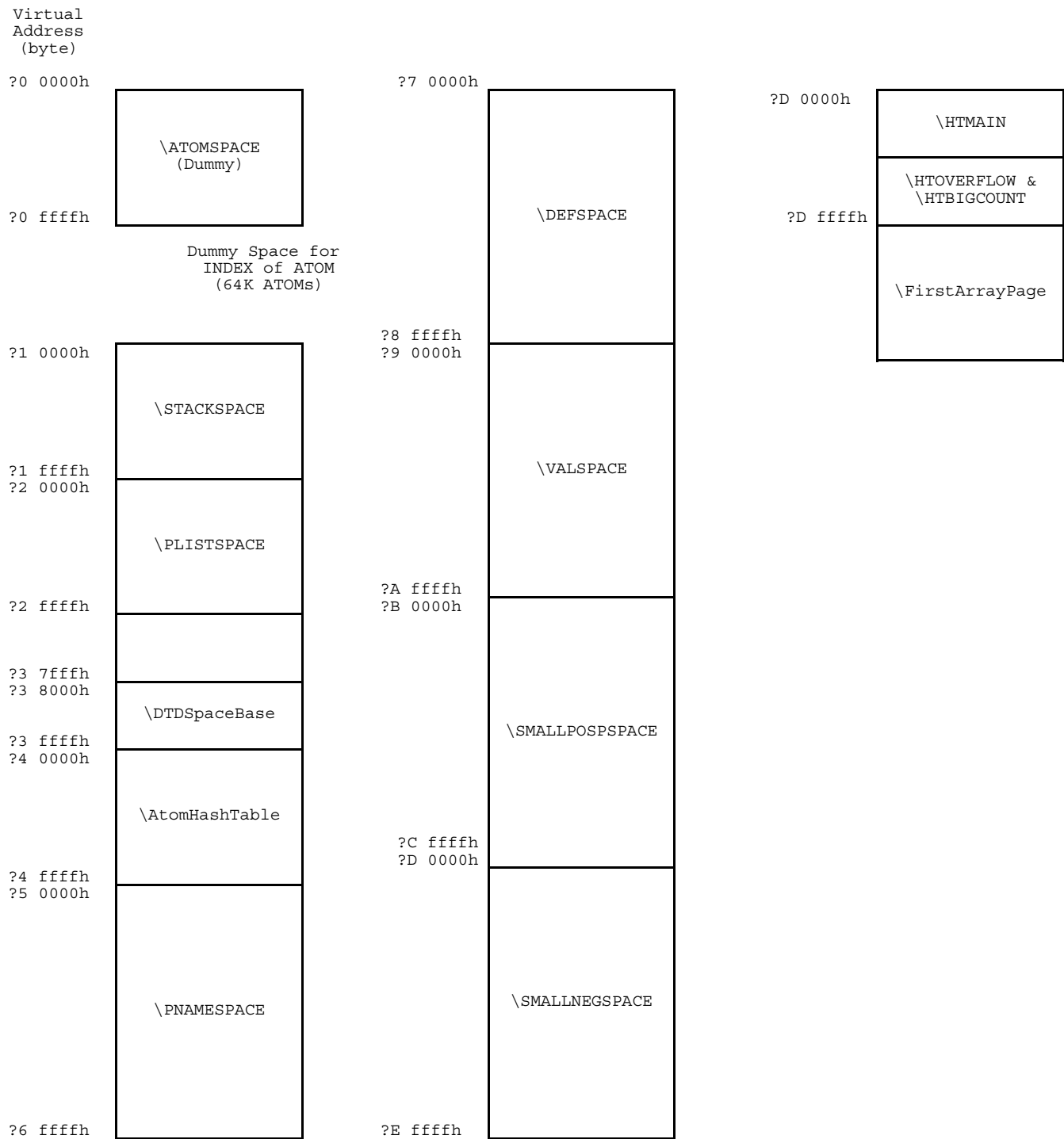
Its index number is X.
And, Its Length is 3 words.

3	0
■	
■	
■	

ME

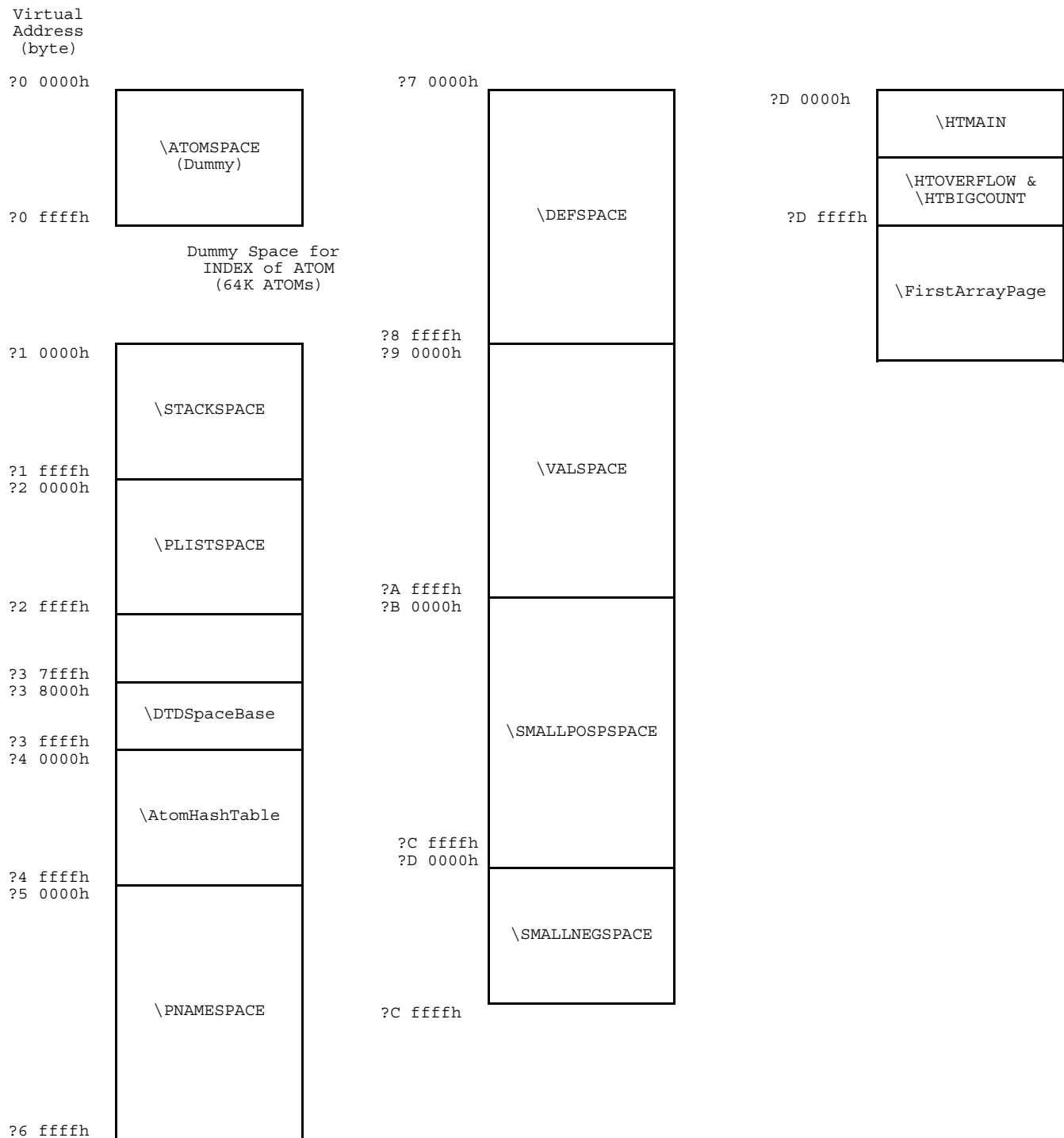


Memory Map (for TEST on SUN-3)



(ADDBASE (LLSH ATOMINEX 1) \PNAMESPACE)

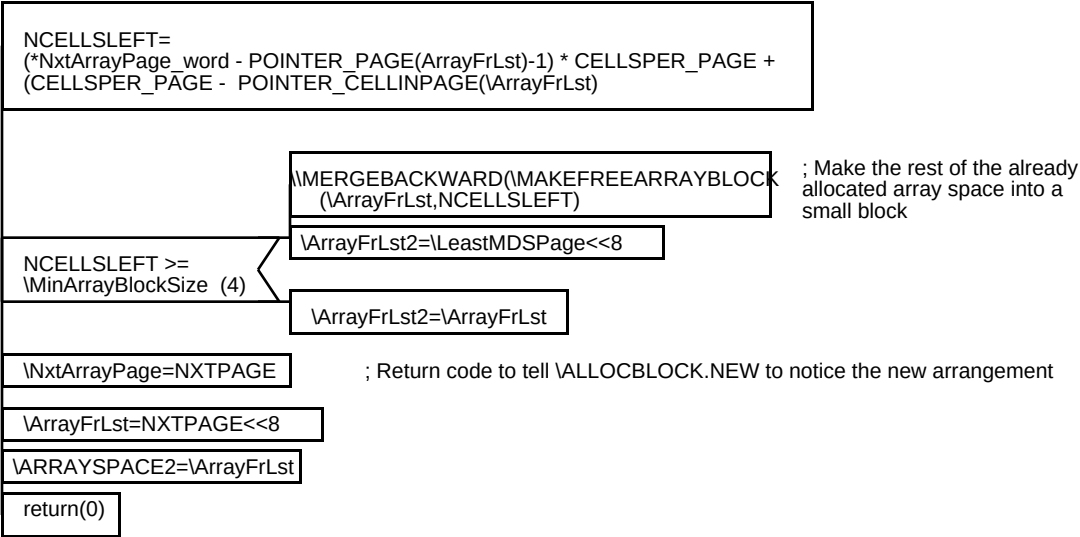
Memory Map (for TEST on SUN-3)



(ADDBASE (LLSH ATOMINEX 1) \PNAMESPACE)

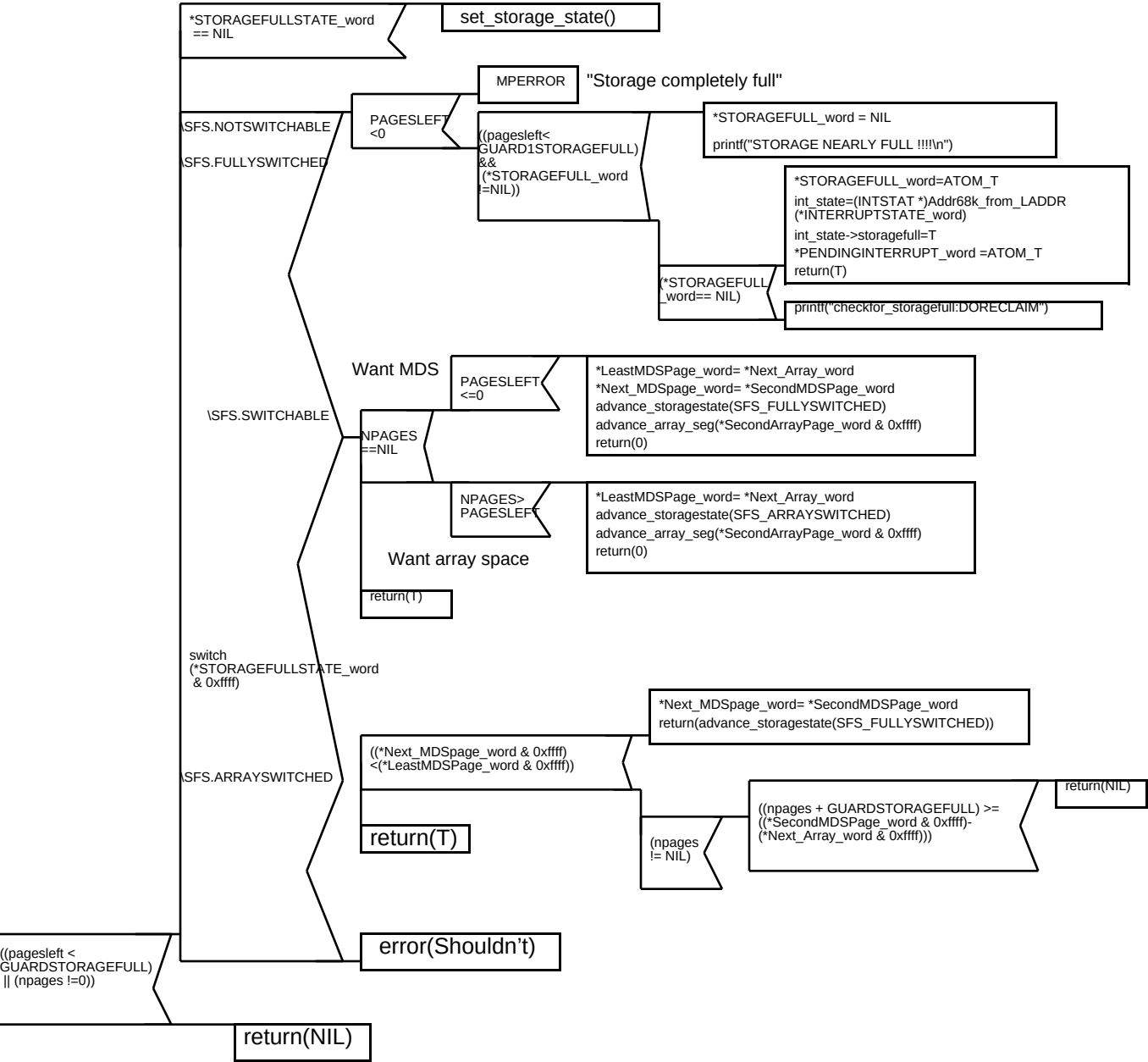
VADVANCE_ARRAY_SEG(NXTPAGE)

Called when the first 8mb are exhausted, and we want to switch array space into the next area, starting with page NXTPAGE -- have to first clean up what's left in the old area

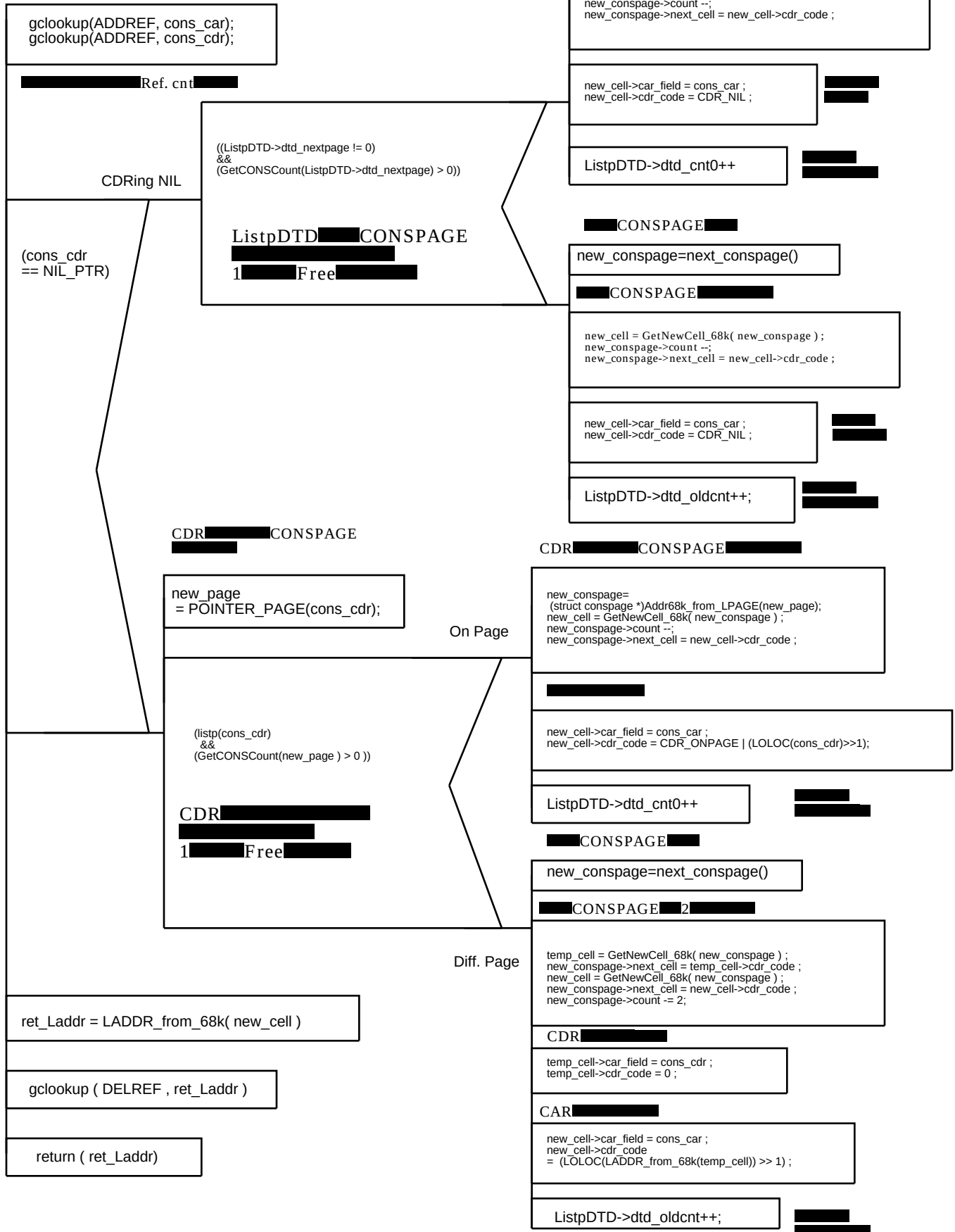


checkfor_storagefull(npages)

pagesleft = (*Next_MDSPage_word & 0xffff) - (*Next_Array_word & 0xffff) - PAGESPER_MDSUNIT



cons (cons_car , cons_cdr)



OP_contextswitch

PushCStack

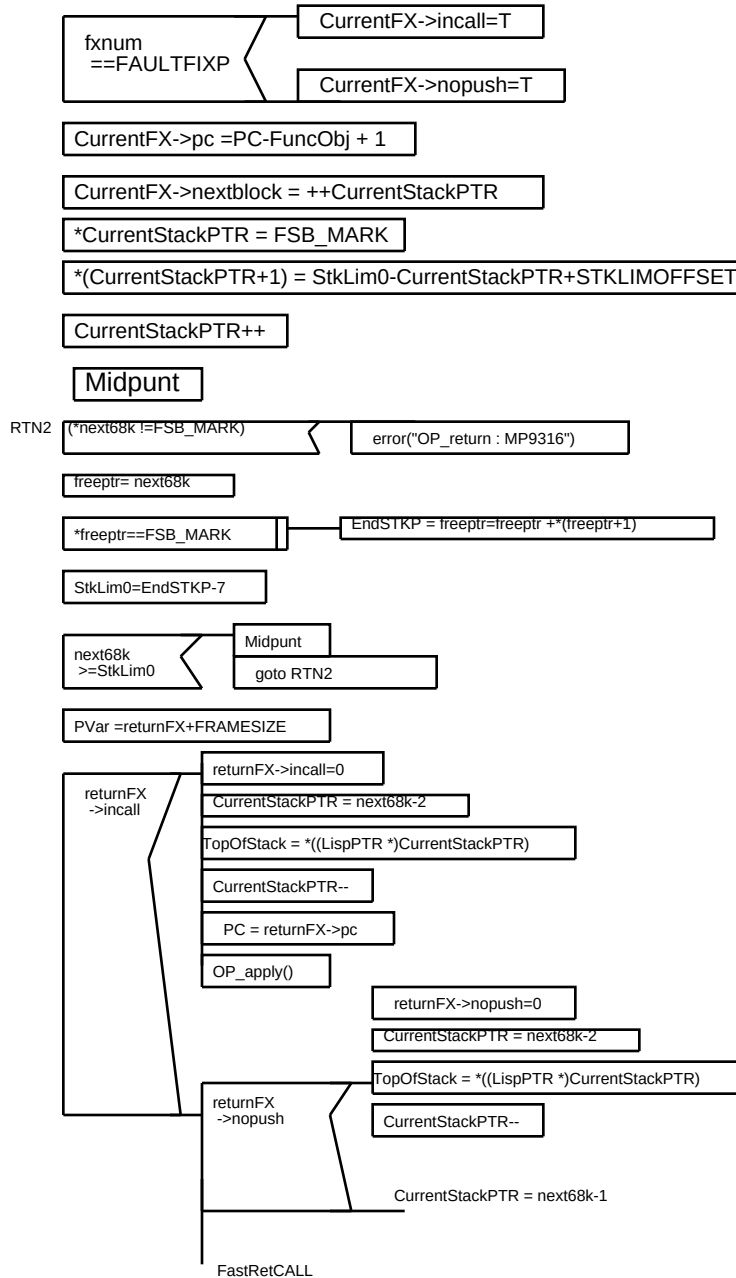
contextswitch(TopOfStack &0xffff)

CurrentFX = STK_OFFSET | Addr68k_from_LADDR(*(InterfacePage + fxnum))

*(InterfacePage + fxnum) = LOLOC(LADDR_from_68k(CurrentFX))

Midpunt

contextswitch(fxnum)



```
frame->usecount--  
return
```

```
frame->
usecount !=0
```

```
frame= clink
```

frame= clink

```
SETACLINK(fx,val) { if(!(fx->alink&1))fx->blink=fx-DLWORDSPER_CELL;
fx->clink=val+FRAMESIZE;
fx->alink=fx->clink+1
```

```
SETBLINK(fx,val) { fx->blink=val;
  if(!(fx->alink & 1))
    { fx->clink= fx->alink; fx->alink |= 1;}}
```

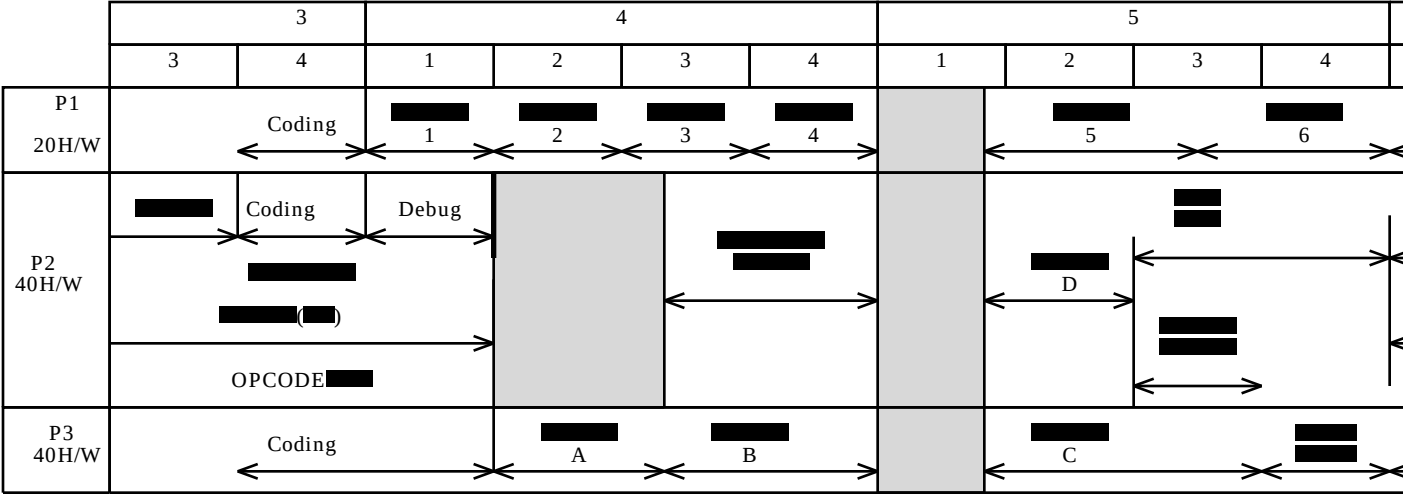
LAST

Replanning 24 March 1987

Takeshi Shimizu

By

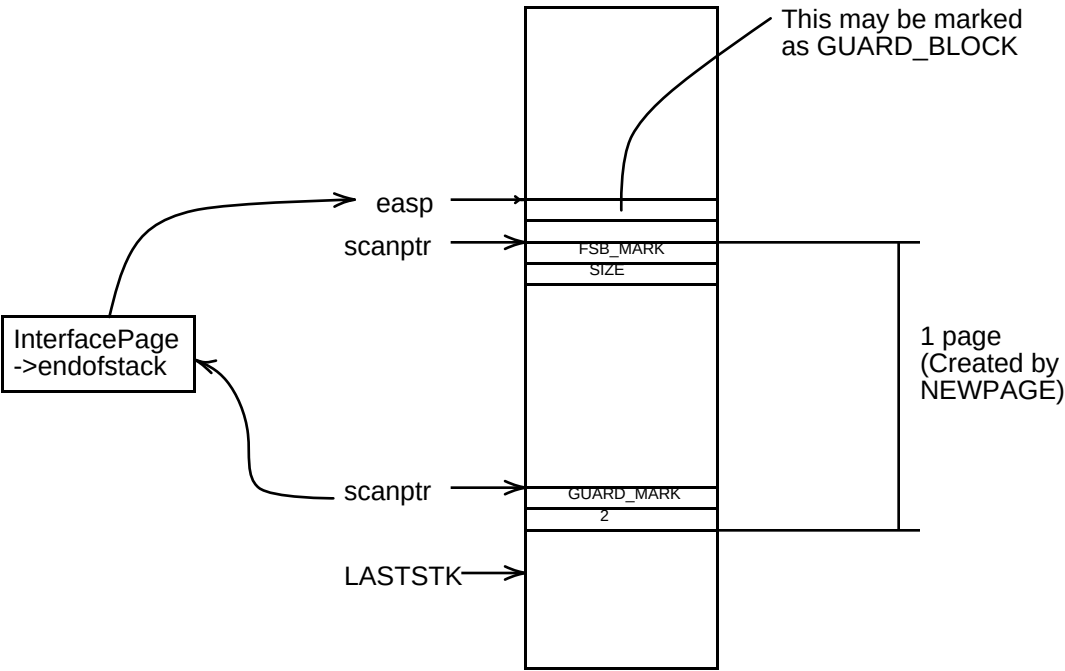
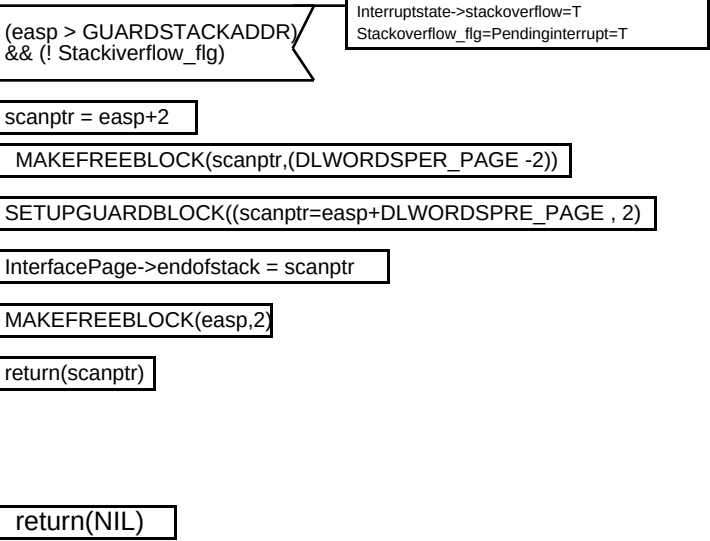
6
3
■■■■
■■■■
■■■■
■■■■



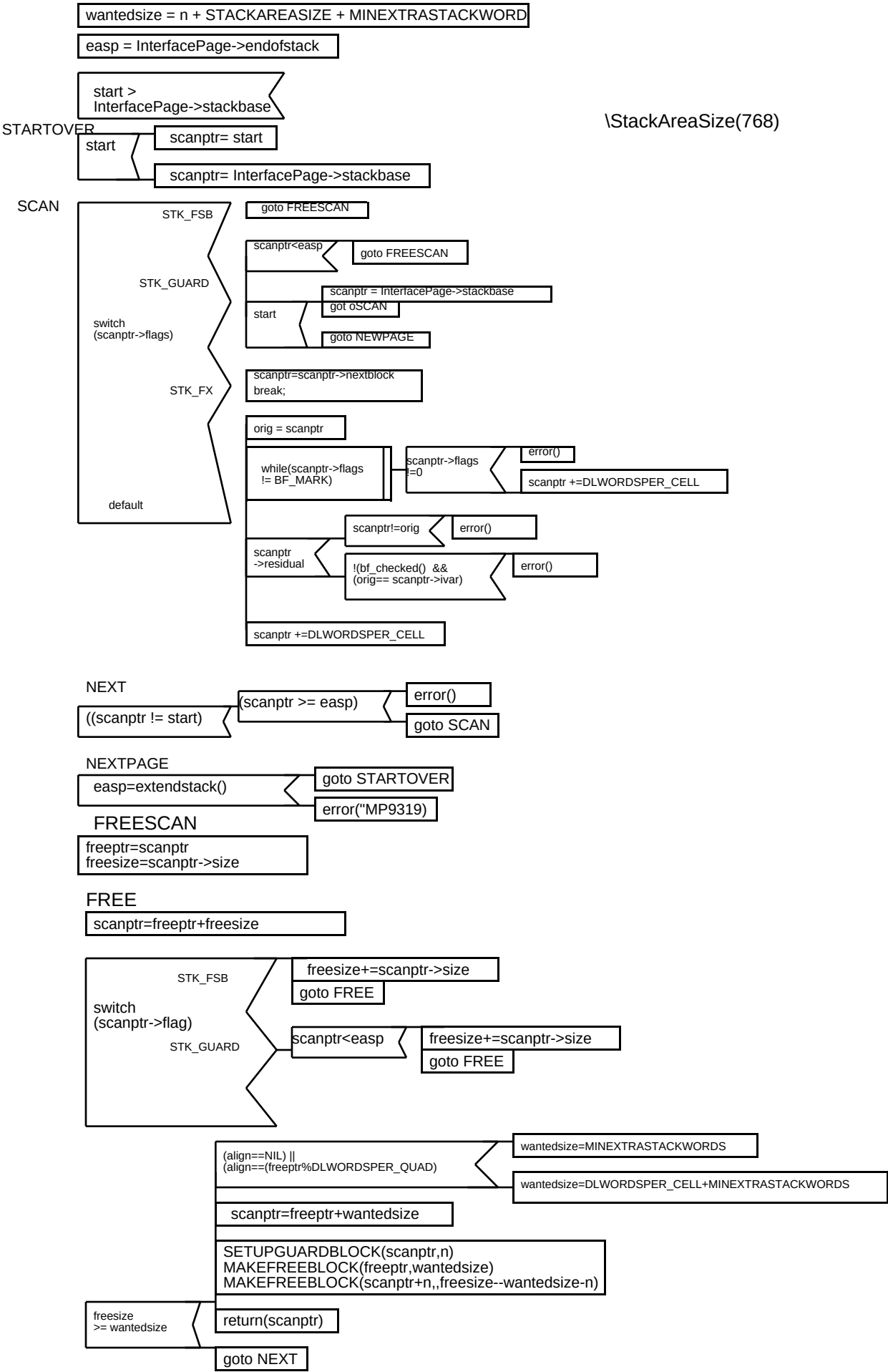
- 1 = { vars.c,vars2.c,binds.c,jump.c,arith.c }
- 2 = { lowlevel.c }
- 3 = { vars3.c }
- 4 = { gvar2.c }
- 5 = { fvar.c }
- 6 = { gc.c }

- = { initdatatype.c , makeatom.c , array.c }
- A = { typeof.c, constants.c, OP_eq, OP_copy , OP_atomcellN }
- B = { conspage.c , stack.c }
- C = { car-cdr.c }
- D = { funcall.c }
- = READ,PRINT,LOADER■■■■

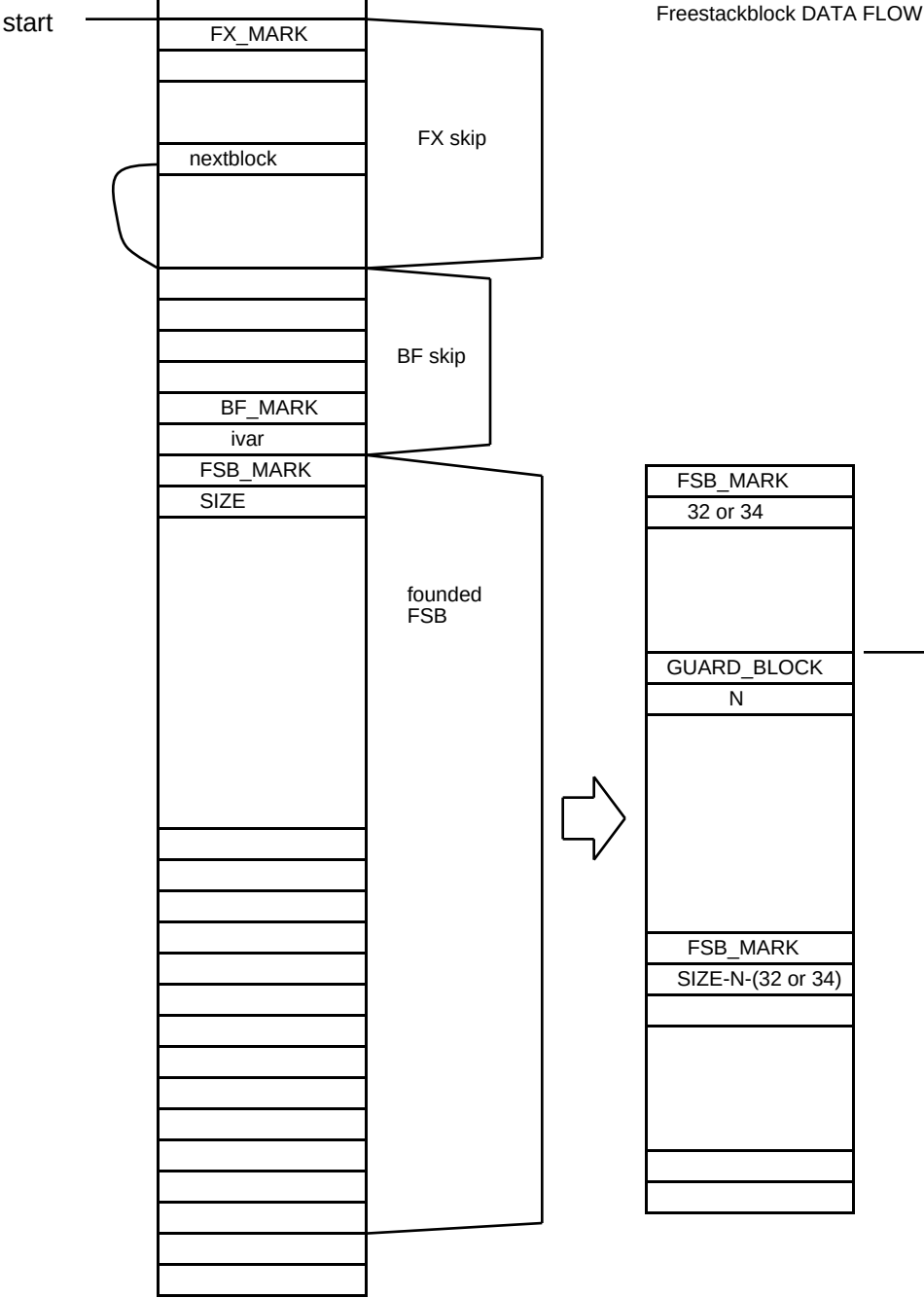
extendstack()



FREESTACKBLOCK(n,start,align)



\MinExtraStackWords(32)



```
\INCUSECOUNT (FRAME)
struct frameex1 *frame
```

INTERRUPTABLE
== NIL

frame->usecount ++
>MAXSAFEUSECOUNT

MPERROR

frame->alink
!=0

scanptr = frame->nextblock

\STK.NOTFLAG

scanptr->flags
 \STK.BF

default

do{ scanptr++;}
while(scanptr->flags != STKBF

break;

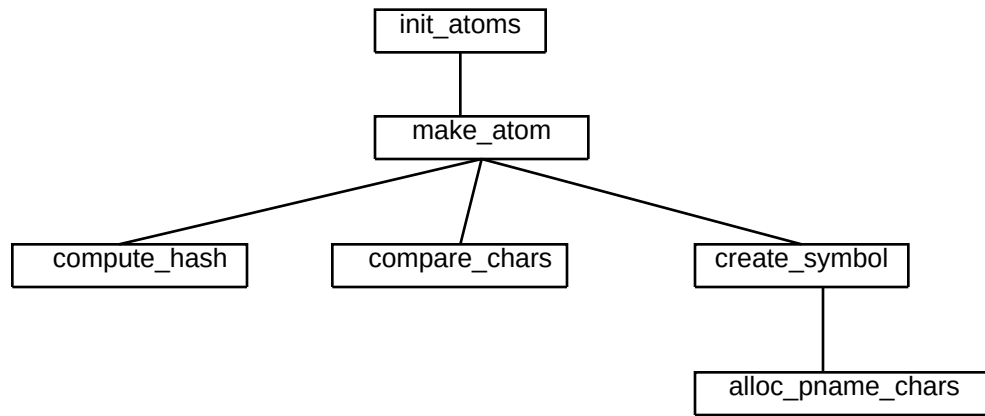
return(frame)

CHECK: whether
scanptr->residual || (scanptr ->ivar= frame->nextblock)

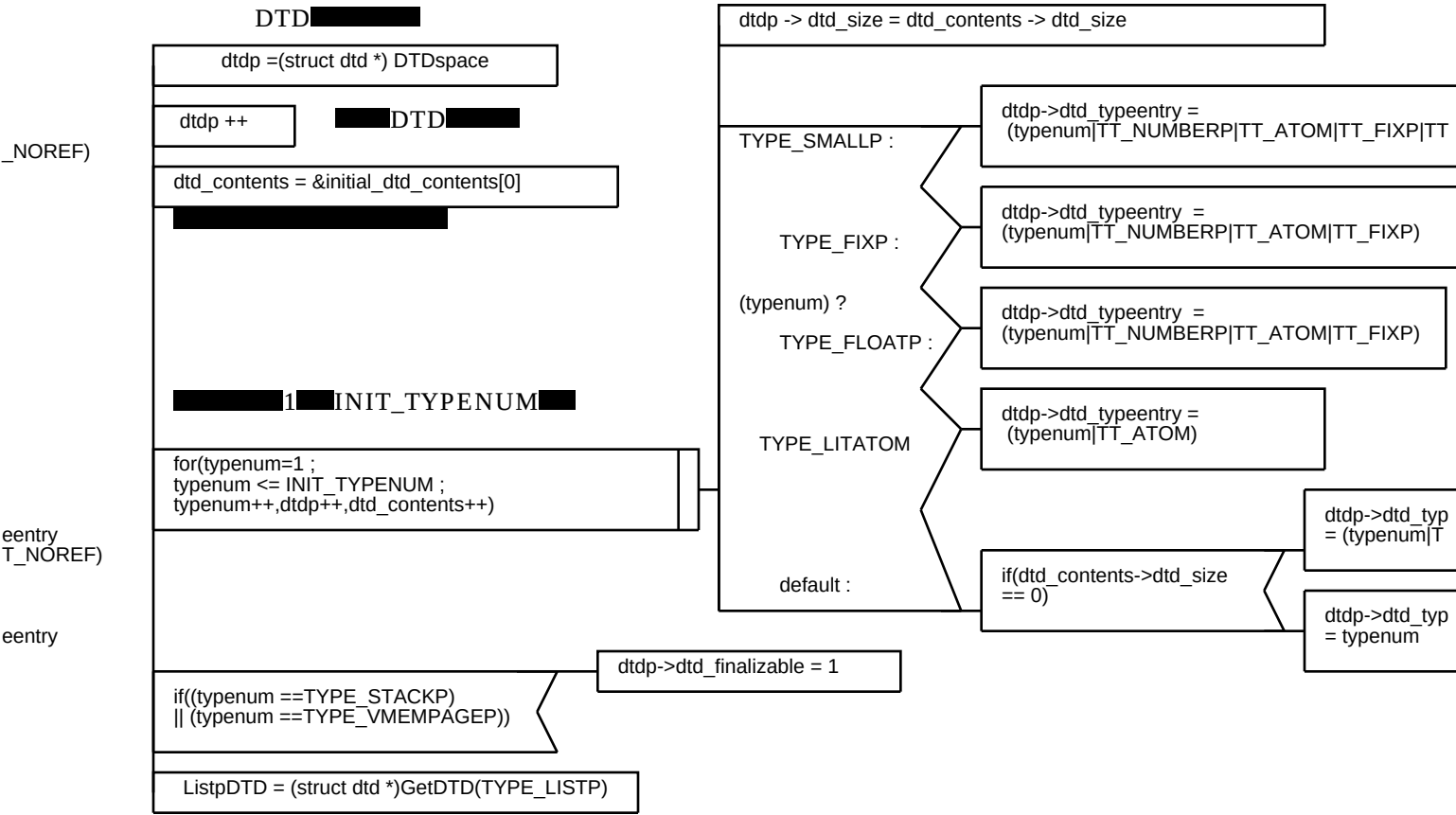
scanptr+=2

scanptr->flags
==STKFX

CHECK:whether
(scanptr-2 == scanptr->blink ||
((scanpre-2)->residual &&
(scanptr-2)->ivar == (scanptr->blink)->ivar



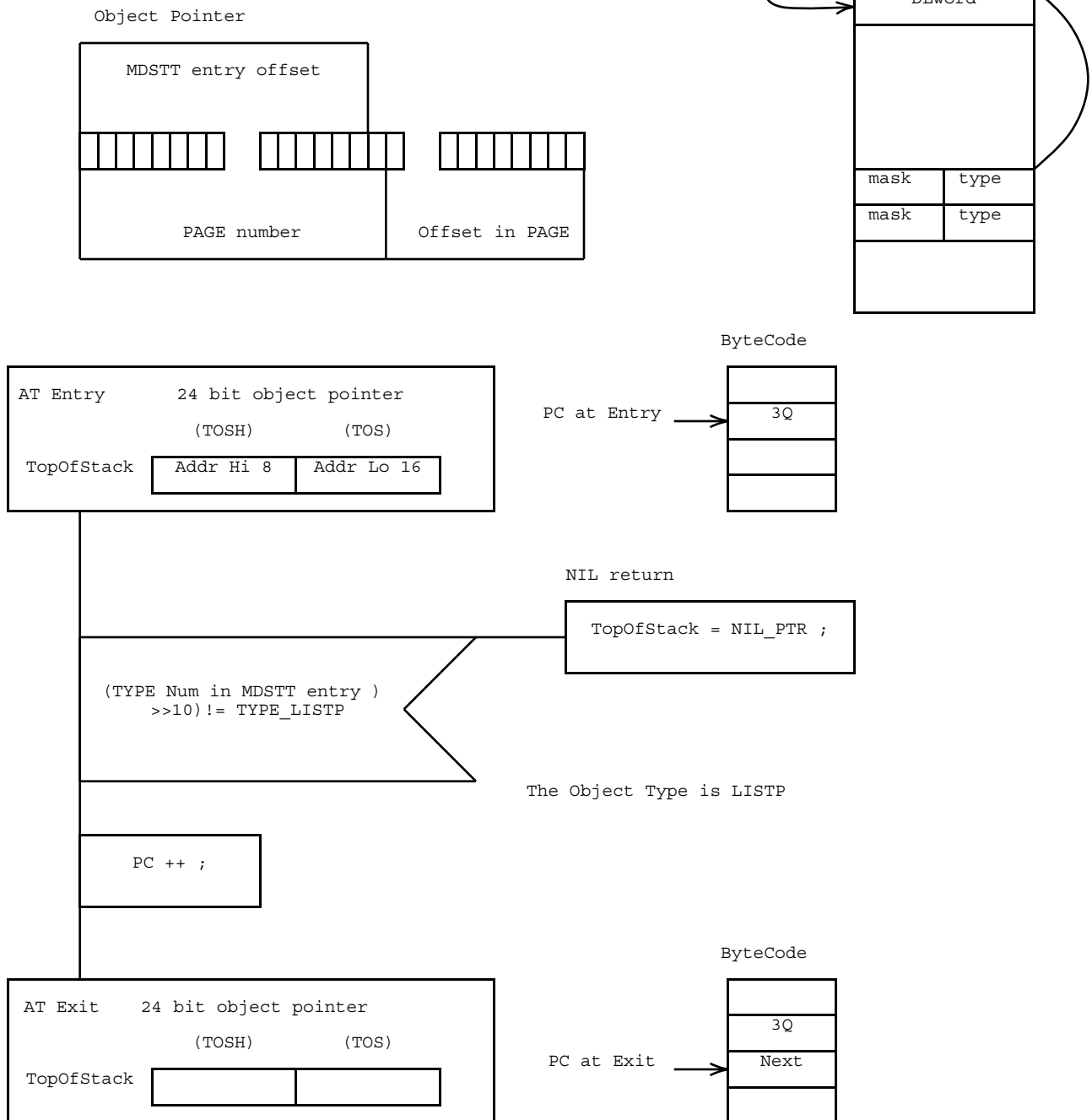
init_datatypes()



SPEC of LISTP(3Q)

OP_listp()

DLword *MDStypetbl



```
make_atom ( char_base, length, non_numericp )
char * char_base;
short length;
short non_numericp;
```

```
return(ATOMoffset
+ first_
char)
))
```

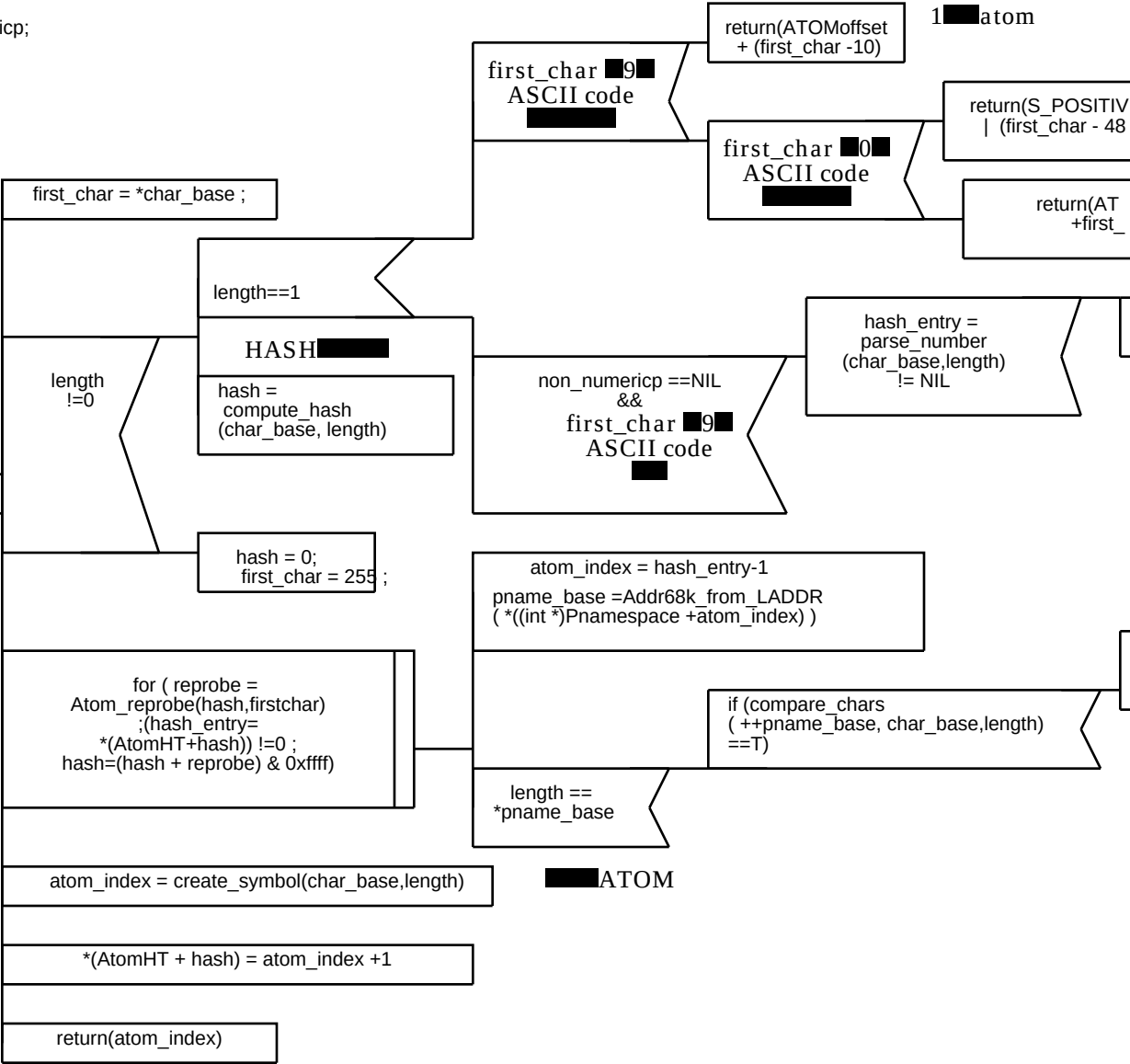
```
return(hash_entry)
```

```
10
```

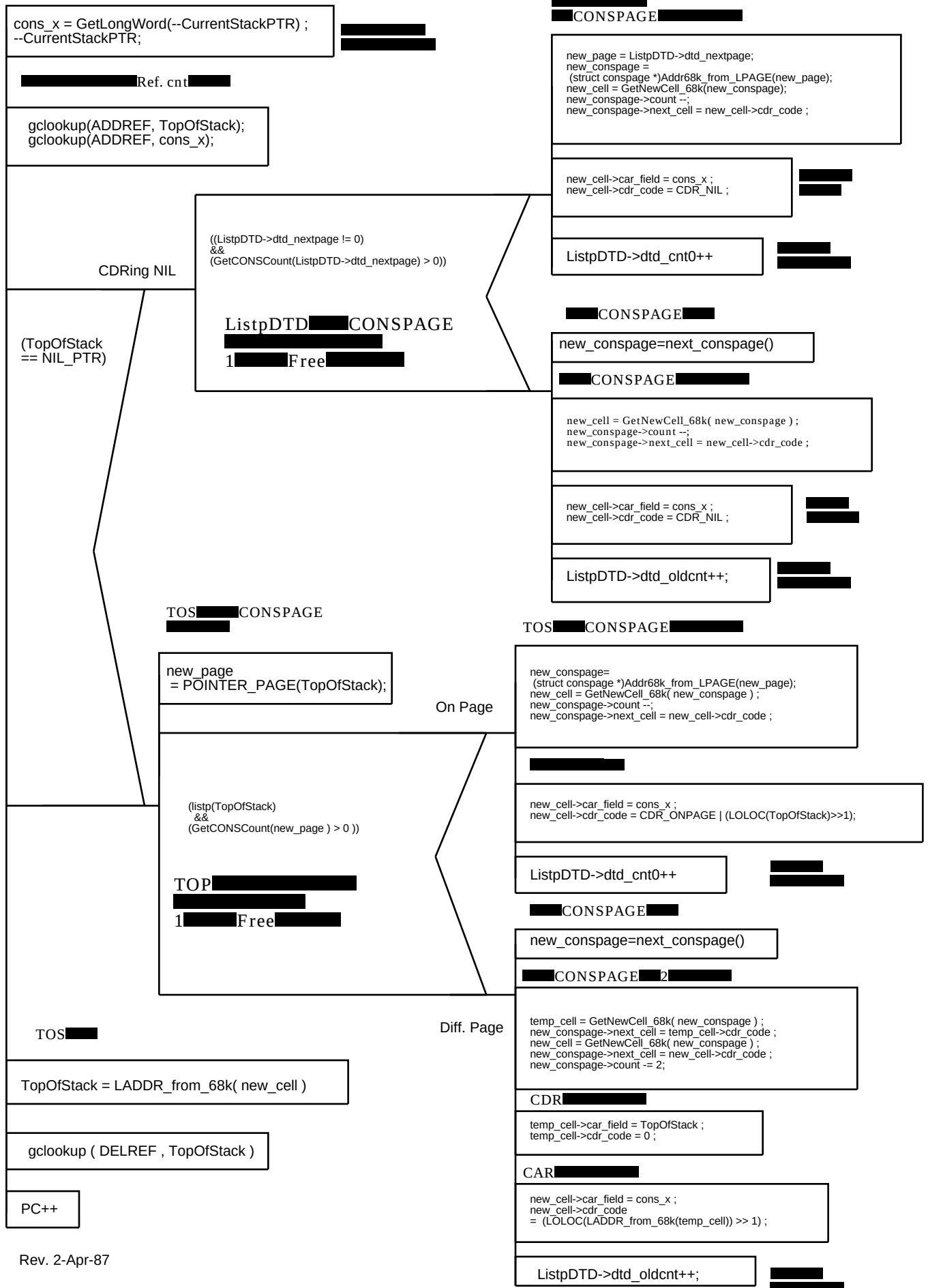
```
make_atom
```

```
ATOM
```

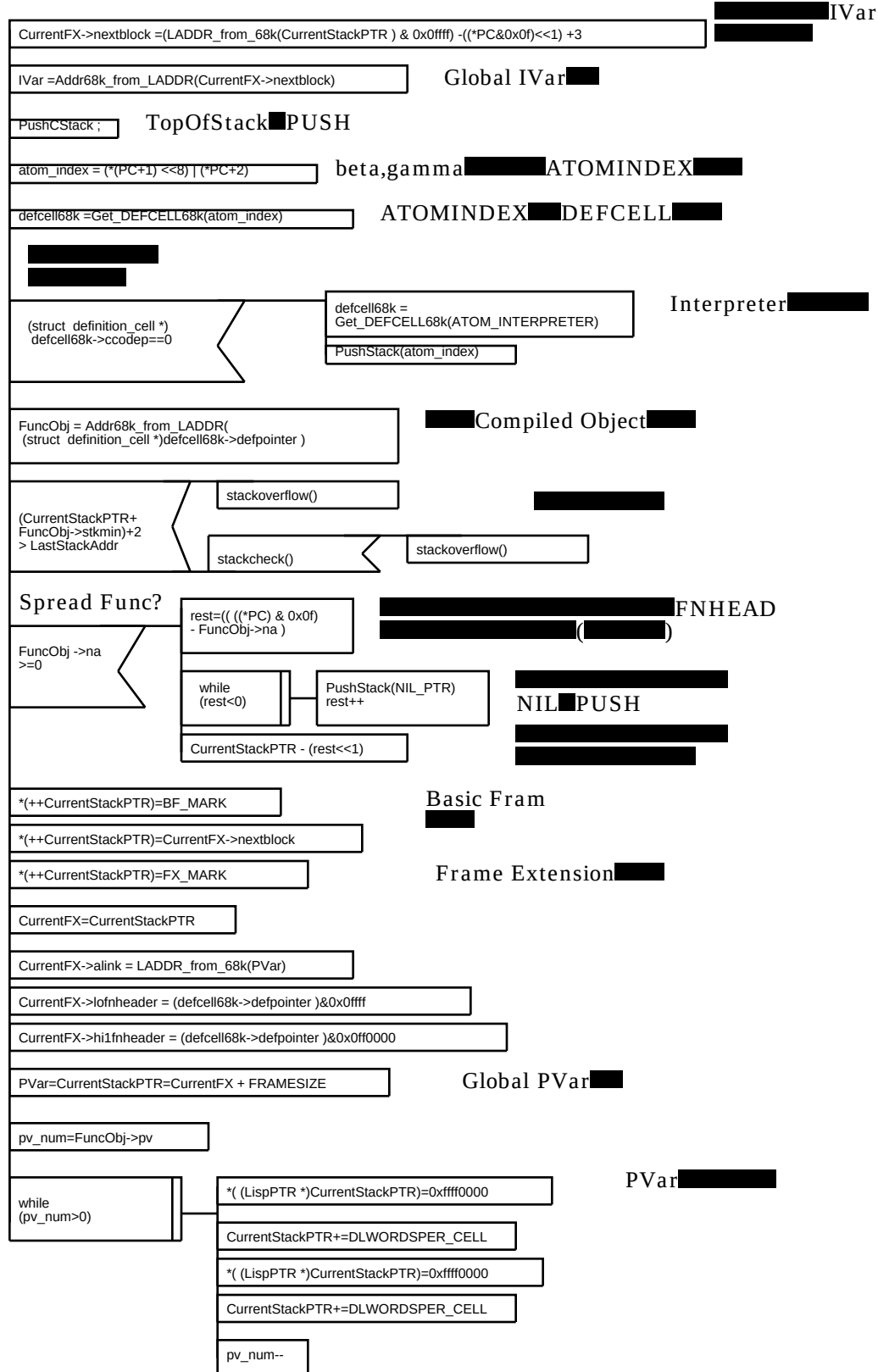
```
Already Exist
return(atom_index)
```



OP_cons ()



OP_fn



hardRet1

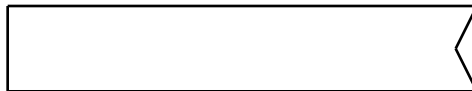
(CurrentFX->alink -1)
!= CurrentFX->clink

hardRet2

CurrentFX->
usecount !=0

hardRet3a

(IVar !=(next68k=Addr68k_from_LADDR
(STK_OFFSET | CurrentFX->nextblock)))
&&
next68k != IVar



CurrentFX->alink
& 1

Fast

CurrentStackPTR = IVar -1

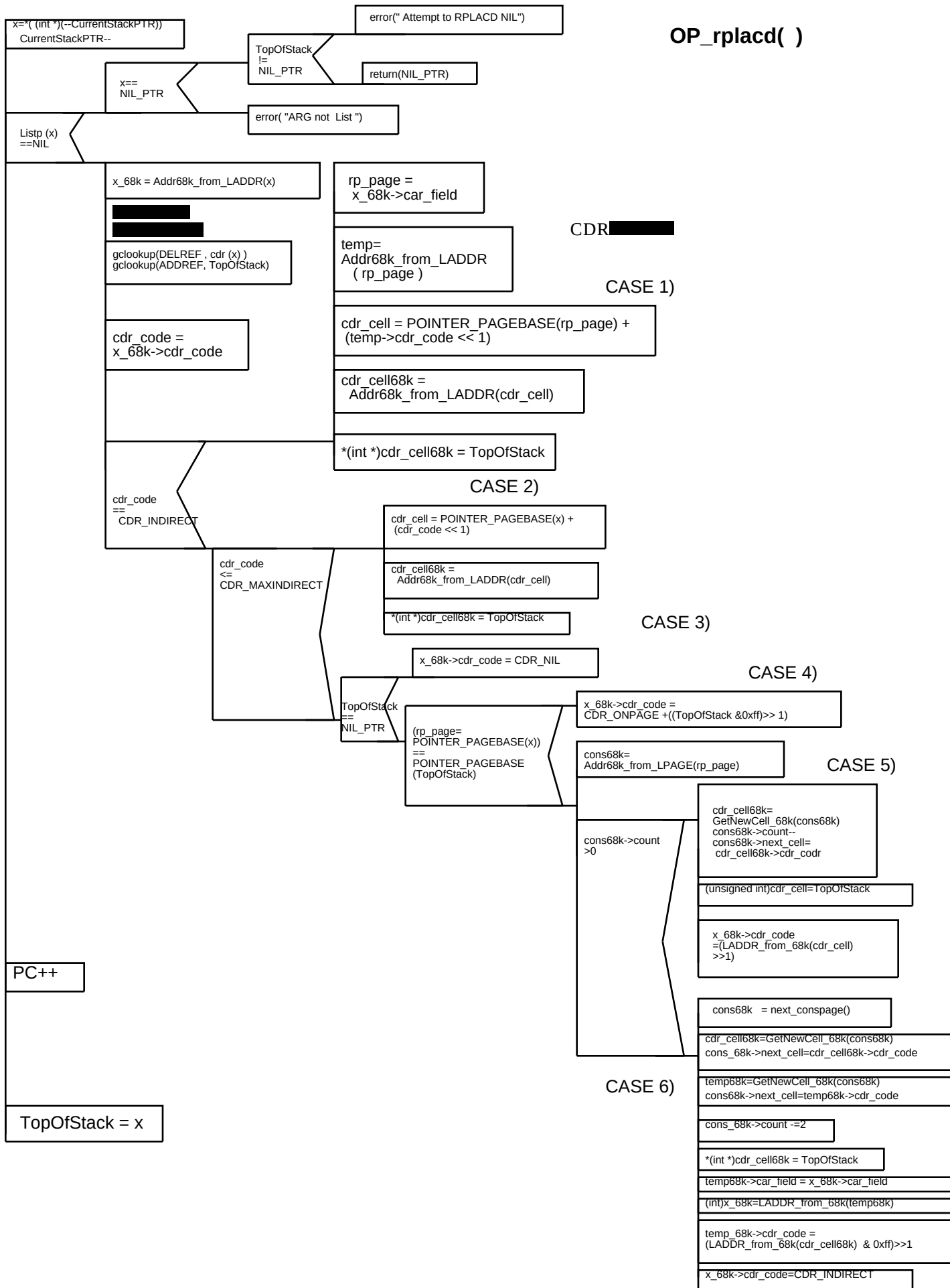
PVar =Addr68k_from_LADDR
(STK_OFFSET | CurrentFX->alink)

CurrentFX = PVar - FRAMESIZE

IVar = Addr68k_from_LADDR
(STK_OFFSET | *((DLword *)CurrentFX - 1))

FuncObj = Addr68k_from_LADDR
((CurrentFX->hi2fnheader <<16) | CurrentFX->lofnheader)

PC = (ByteCode *)FuncObj +CurrentFX->pc

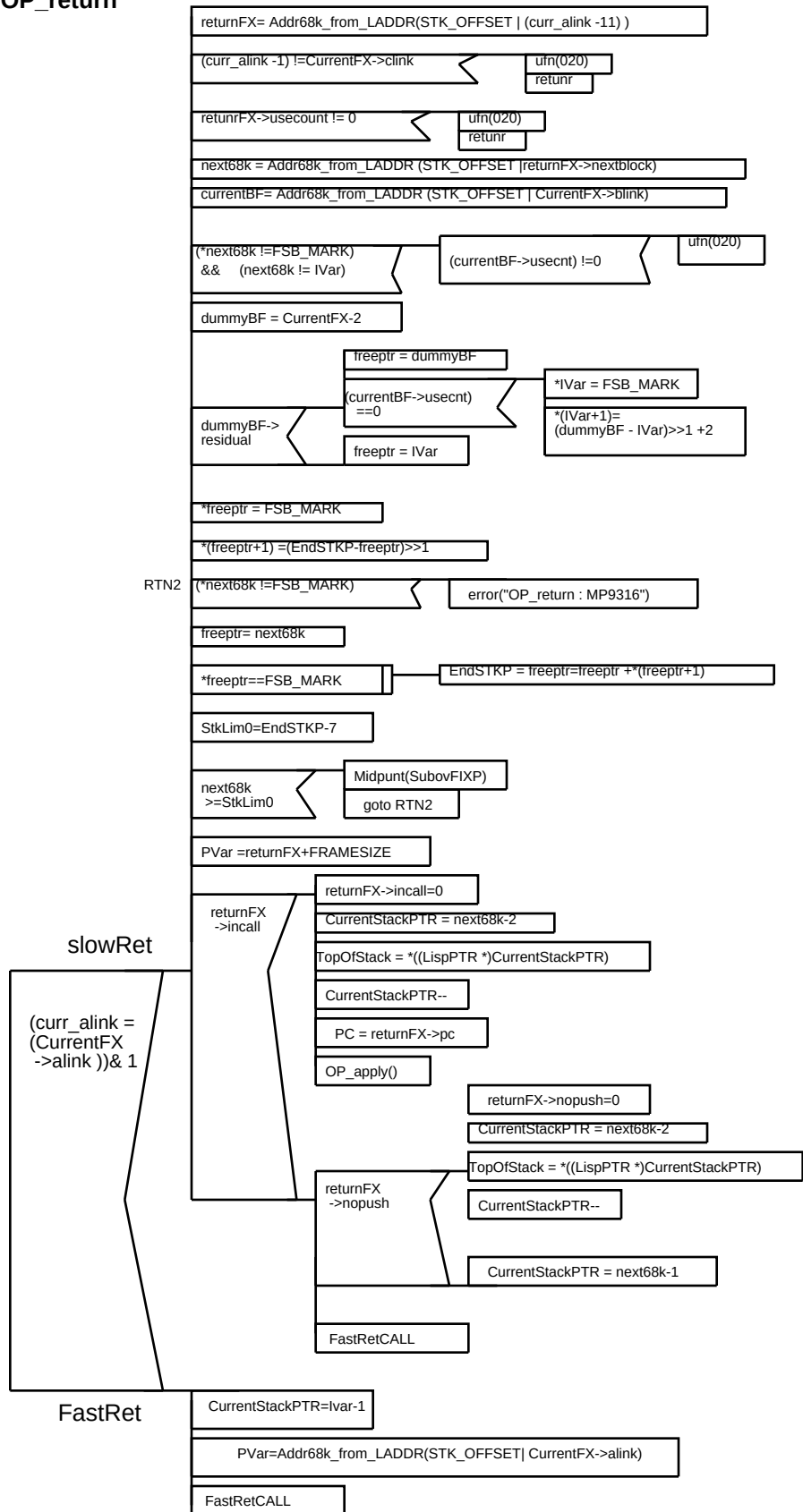


2

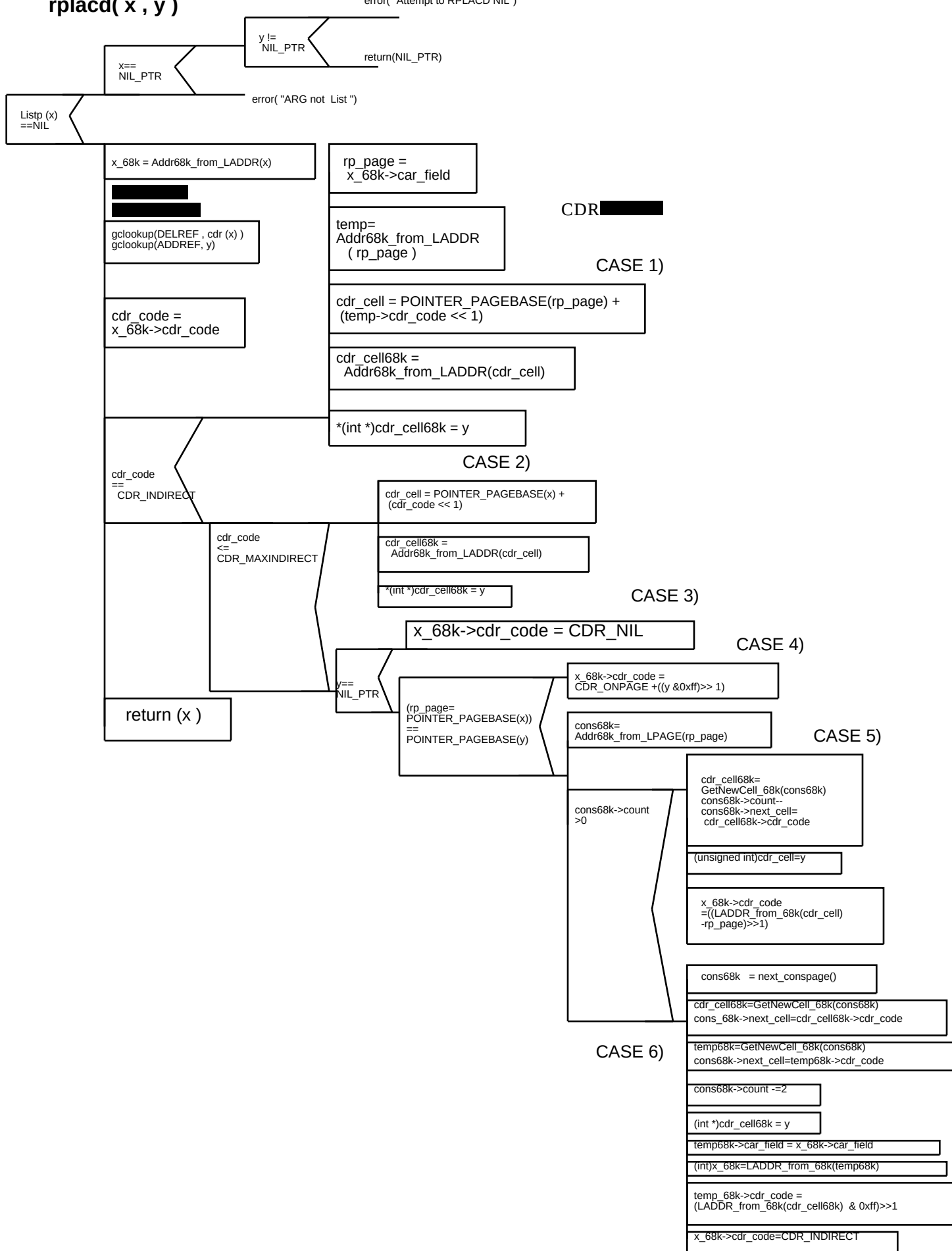
[illegible]



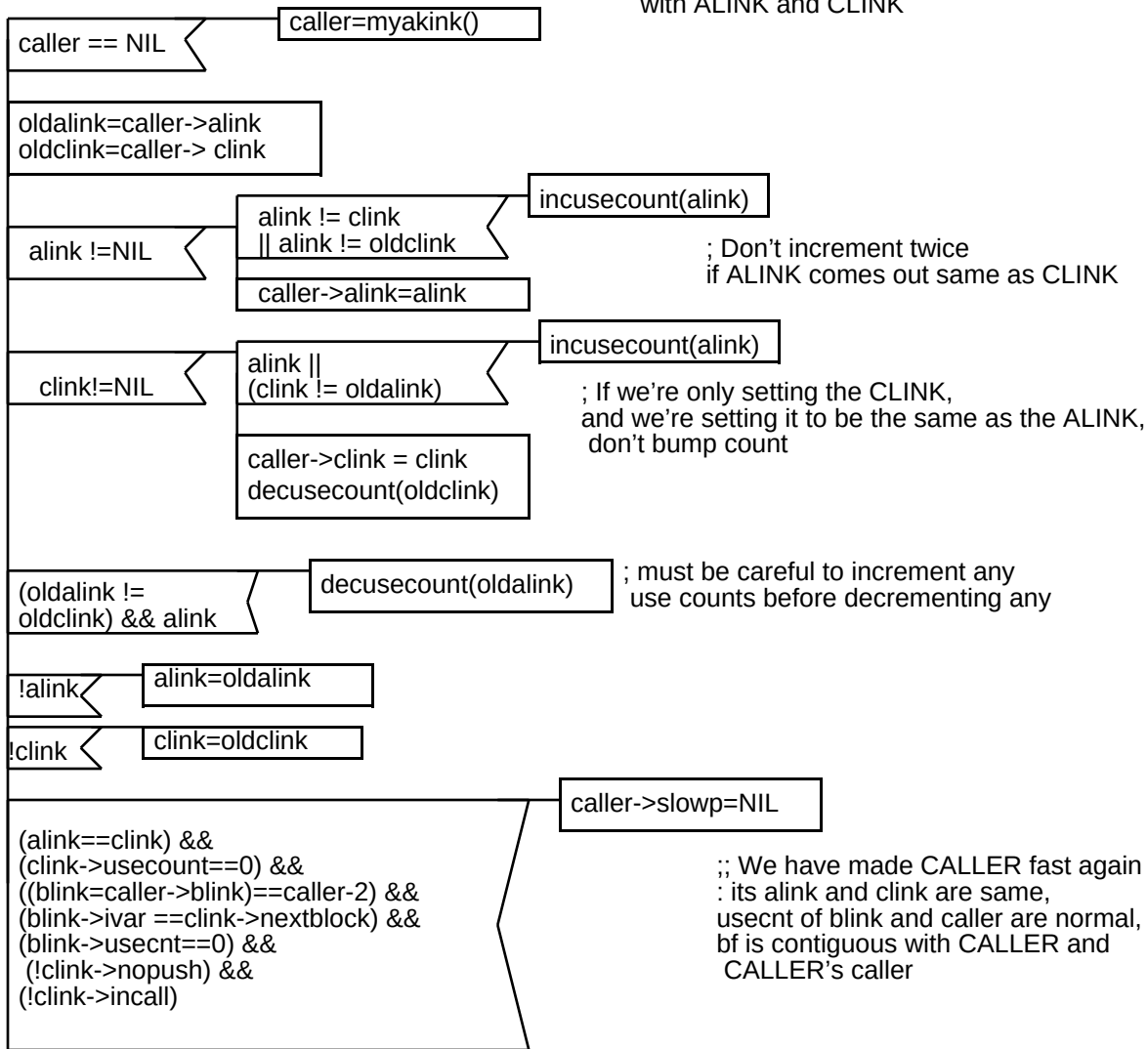
OP_return

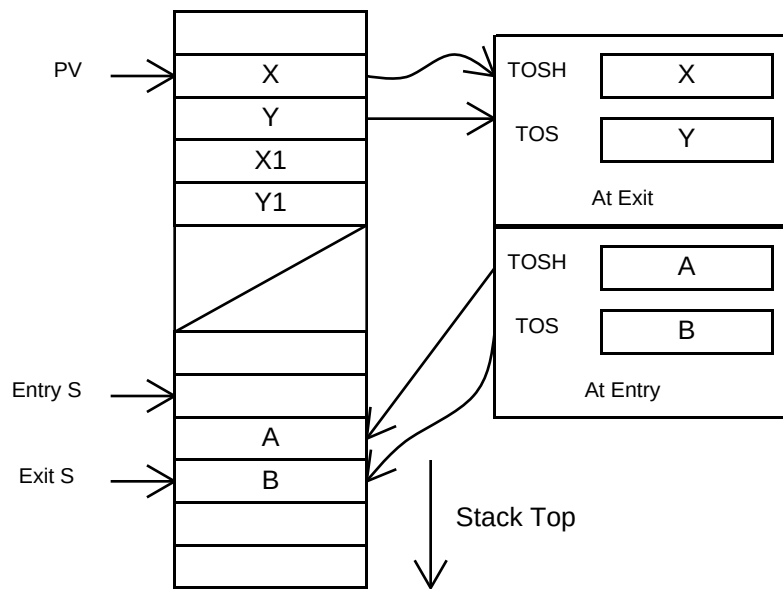


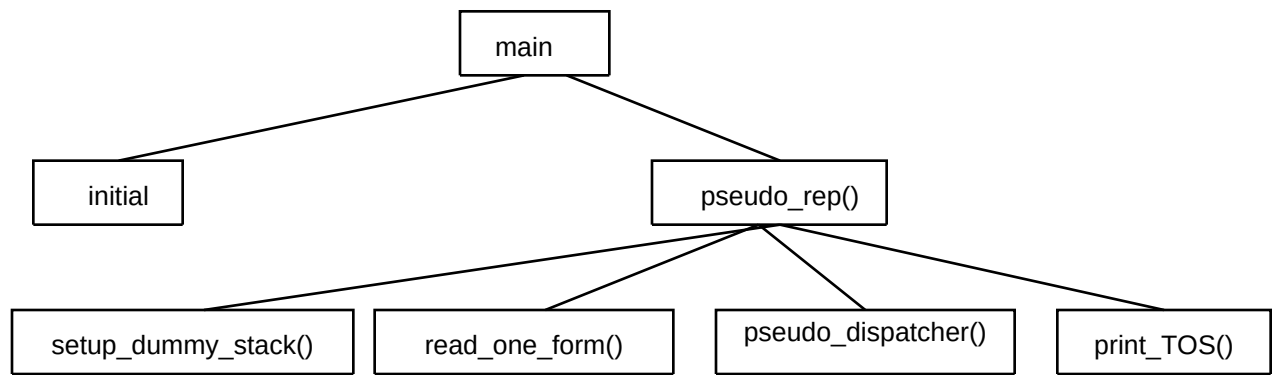
rplacd(x , y)



; Smashes caller's ALINK and/or CLINK
with ALINK and CLINK







SourceFiles.tedit

The following is the list of files in either the Carol sysout or the current lispcore sysout, showing those which we might not want to release the sources of. The notes "<removed>" and "<added>" indicate files that were removed or added to the lispcore loadup between the two versions.

Files in Lisp.sysout:

10MBDRIVER	
AARITH	
ABASIC	
ACODE	
ADDARITH	
ADIR	
ADISPLAY	
ADVISE	
AERROR	
AFONT	
AINERRUPT	
AOFD	
APRINT	
APS	<removed>
APUTDQ	
ASSIST	
ASTACK	
ATBL	
ATERM	
BOOTSTRAP	
BREAK	
BRKDOWN	
BSP	
BYTECOMPILER	
CHAT	
CLISP	
CLISPIFY	
CLEARINGHOUSE	<added> sensitive for now (bvm)
COMMENT	
COMPATIBILITY	<added>
COMPILE	
COREIO	
COURIER	sensitive protocols (Raim)
DEDIT	
DEXEC	
DFILE	
DISKDLION	
DLAP	
DLIONFS	
DMISC	
DPUPFTP	
DSPRINTDEF	
DTDECLARE	
DWIM	
DWIMIFY	
EDIT	
FILEIO	
FILEPKG	
FLOPPY	
FONT	
FPPATCH	<removed>
HELPDL	
HIST	


```

HLDISPLAY
HPRINT
IMAGEIO
INSPECT
INTERPRESS
IOCHAR
LEAF
LLARITH
LLARRAYELT
LLBASIC
LLBFS
LLCHAR
LLCODE
LLDATATYPE
LLDISPLAY
LLEETHER
LLFAULT
LLFCOMPILE
LLFLOAT
LLGC
LLINTERP
LLKEY
LLNEW
LLNS
LLREAD
LLSTK
LLSUBRS
LOADFNS
MACHINEINDEPENDENT
MACROAUX
MACROS
MASTERSCOPE
MATCH
MENU
MISC
MOD44IO
MSANALYZE
MSPARSE
NEWPRINTDEF
NSFILING          definitely proprietary (bvm)
NSPRINT           <added>
PASSWORDS
PCALLSTATS        <removed>
PMAP
POSTLOADUP
PRESS
PRETTY
PROC
PUP
RECORD
SPELL
SPP
TRSERVER
TTYIN
UNDO
VOLUMEALLOCATIONMAP
VOLUMEFILEMAP
WBREAK
WEDIT
WINDOW
WTFIX

```

Additional files in Full.sysout:

```

AREDIT
ATTACHEDWINDOW
EDITBITMAP
FILEBROWSER
GRAPEVINE      potentially sensitive (bvm)
GRAPHER        <added>
HASH           <added>
ICONW
IMAGEOBJ
LAFITE         potentially sensitive (bvm)
LAFITEBROWSE   <added> potentially sensitive (bvm)
LAFITEMAIL     <added> potentially sensitive (bvm)
LAFITESEND     <added> potentially sensitive (bvm)
MAILCLIENT    potentially sensitive (bvm)
MTP            potentially sensitive (bvm)
RDSYS
READNUMBER
READSYS
REMOTEVMEM
SAMEDIR        <added>
SINGLEFILEINDEX <added>
SPY            <added>
TEDIT
TEDITABBREV
TEDITCOMMAND
TEDITFILE
TEDITFIND
TEDITHCPY
TEDITHISTORY
TEDITLOOKS
TEDITMENU
TEDITSCREEN
TEDITSELECTION
TEDITWINDOW
TEXTOFD
TFBRAVO
VMEM
WHEREIS        <added>

```

Other files:

```

MAINTAIN      potentially sensitive (bvm)

```

=====

The following are the messages which lead to the creation of the above list:

Date: 11 Jun 84 15:42 PDT
From: Sannella.pa
Subject: Restricted Sources Files?
To: LispCore↑.pa
cc: Sannella.pa
Reply-To: Sannella.pa

Beau has asked me to collect a list of the source files that we want to give to customers. Please send me (Sannella) the names of any source files you are responsible for which contain sensitive or proprietary information that should not be let out to the public at large. For each file, I would also like a few words of description of what it contains --- a sentence would be fine. I will compile the

results, and send out another message. Thanks.

----- Next Message -----

Date: 12 Jun 84 10:58 PDT

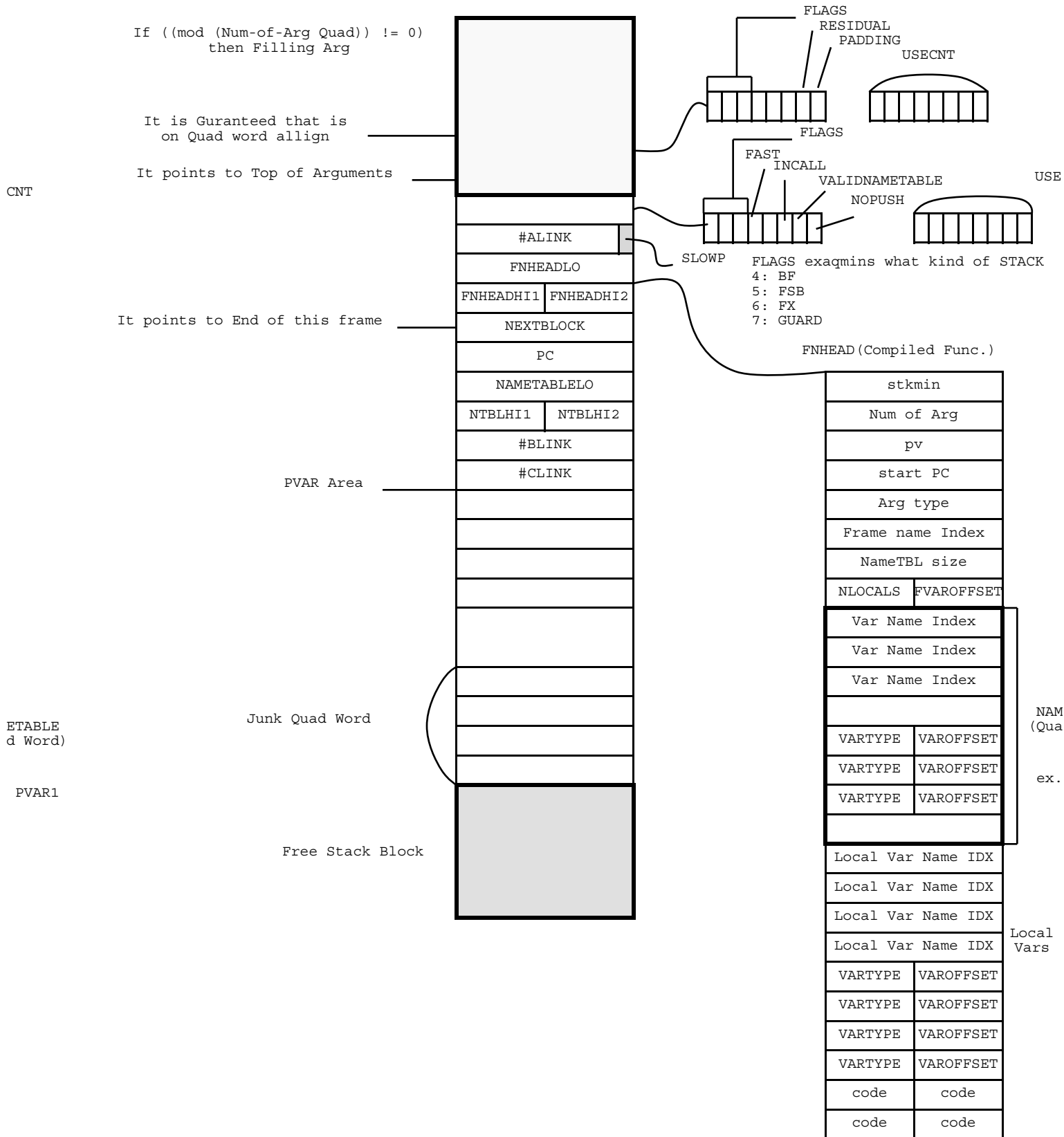
From: vanMelle.pa

NSFILING is definitely proprietary (implementation of Filing protocol).

I would also consider CLEARINGHOUSE sensitive for now. Even though Clearinghouse protocol is now public, I think there may still be some things in that file that aren't.

All the LAFITE* files, as well as GRAPEVINE, MAILCLIENT, MTP and MAINTAIN are potentially sensitive. Of course, no customers are even getting the dcom's of those, so there would be no reason to give them sources.

Bill



Stack and Function Header Format

original: August 15, 1981 by Bill van Melle, Larry Masinter
reformatted for implementor's manual: July 2, 1985 by Ron Fischer
revised: August 1, 1985 by Bill van Melle
revised: July 16, 1988 by Frank Shih (minor note re: Maiko)

Stack structure

The stack segment in the Interlisp-D virtual memory consists of a series of stack *blocks*, which represent frames for active invocations of Lisp functions and free space left over. There are actually four specific kinds of blocks that can appear in the stack space: basic frames (abbreviated BF), frame extensions (abbreviated FX), free blocks (FSB), and guard blocks (GUARD). Each BF holds the arguments of a particular function call, while the FX holds other locally-bound variables and temporary storage for the function. A BF and FX are created at each call or entry, and released upon return or exit. A FX and its associated BF will simply be referred to as a *frame*.

Basic frame

A Basic Frame consists of the *n* arguments to the function being called, possibly followed by a cell of padding, followed by the BF word containing flags and a pointer to the first argument. The BF word is quadword-aligned. Every frame extension is immediately preceeded by a BF, either the actual BF, or by a "dummy" BF pointer in which the R bit is on and the IVAR pointer is valid. Only "slow" FX's can be preceded by a dummyBF.

```
IVAR:  *+--+--+--+--+--+*+--+--+--+--+--+*+--+--+--+--+--+*+--+--+--+--+--+*
        |          0          |          V a l u e 1          |
        |          ...         |          V a l u e n          |
        |          0          |          |
BF:     *+--+--+--+--+--+*+--+--+--+--+--+*+--+--+--+--+--+*+--+--+--+--+--+*
        | 1 0 0 0 0 0 | R | P |   Ext. cnt.   |          IVAR          |
        *+--+--+--+--+--+*+--+--+--+--+--+*+--+--+--+--+--+*+--+--+--+--+--+*
```

The fields are interpreted as follows:

- IVAR The 2nd word of the BF cell points back at the IVAR.
- R "Residual". This bit is on in "dummy" basic frames which are actually only 2-word quantities. Normal function call leaves the bit off. Dummy BF's are created when an FX has to move or be copied (on stack overflow or returning to a frame with non-zero use count). It's not clear these are really necessary—one could move/copy the entire frame (BF+FX).
- P "Padded". If 1, the actual number of arguments in the frame is 1 less than the size would indicate.
- USECNT Number of frame extensions (less 1) pointing at this BF. Initially 0.

Frame extension

A Frame extension consists of 10 words of control information, followed by storage for locally bound variables (PVARs), then binding slots for any variable referenced freely by the function, followed by dynamic storage ("the stack").

```
*+--+--+--+--+--+*+--+--+--+--+--+*+--+--+--+--+--+*+--+--+--+--+--+*
```


	table of names is inside the function header); otherwise this field points at something that has the same format as a function header (see below).
BLink	When X is set, points at the BF for this frame. When X is not set, BF=FX-2.
CLink	When X is set, points at the returnee for this frame. When X is not set, CLink=ALink. CLink, like ALink, points at the PVAR area of the FX rather than at the beginning of the FX.
PVAR	This area contains PROG/local variable values and free variable binding pointers. Initialized with all ones in the left half, which denotes {variable not bound} for PROG variables, and {variable not looked up} for free variables (see below). PVAR is quad-aligned (hence so are FX+1 and BF), and contains an even number of cells (so that TEMP is also quad-word aligned).
TEMP	contains the temporaries (dynamic stack space) of the function. The first two cells of this region are garbage, for the benefit of the Dolphin hardware stack reloading. TEMP is quadword aligned, if necessary by padding out the PVAR area.

When a function is called, its arguments appear at the end of the caller's FX. The implementation is designed so that the bookkeeping for a BF appears at its high end, and for a FX at its low end, so that the arguments can be made into a new BF without having to copy them. After thus fabricating a new BF and shortening the old FX, the new FX is created. Upon return, the function value ("top of stack", which appears at the end of the returner's FX) is preserved while the returner's frame is deleted; then the value is pushed onto the caller's FX and execution resumes in the caller.

PVAR Slots and FVAR (Binding) Slots

When a variable is bound, via the BIND opcode, the corresponding slot in the PVAR area is filled in with the value:

```
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
| 0 |               | value |
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
```

When a free variable is found during free variable lookup, the variable's binding slot is filled in with:

```
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
| binding pointer (even) | 0 | bind-addr-hi | bind-addr-hi |
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
```

This two word quantity is the (swapped) address of the location where the binding of the variable actually occurs. If the variable is not bound on the stack, this is the location of the variable's top level value, or possibly some other piece of storage. Note that all legitimate pointers are double-word aligned, and so have the low order bit turned off.

Note: on Maiko, the bind-addr-hi is **not** duplicated, i.e.:

```
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
| binding pointer (even) | 0 | 0 | bind-addr-hi |
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
```

The only way to tell between the PVAR and the FVAR region is to look in the name table (or function header if V is off): (PV+1)*2 is the number of cells (doublewords) in the PVAR region; NLOCALS is the number of those which are PVAR slots; the rest are FVAR slots or padding.

FVAR_ (the free variable assignment opcode), tests whether the binding address of the variable is in stack space. If so, it can do the assignment directly. If not, it must do the assignment using RPLPTR, i.e., decrement the reference count of the old value and increment the reference count of the new. This can be done by punting to the ufn if desired: the arguments to the ufn are the new value and the address (in normal order) of the binding location.

Note that this architecture does not specify anything about where non-stack bindings can be. Ordinarily, if the binding pointer is not in stack space, it is the top-level value cell of the variable, but it can also be used for closures, etc. to allow data structure manipulation to look like variable reference. The only current instance of this use is that the binding of RESETVARSLST at the top level of each process is actually a pointer into the process handle, for the benefit of cleaning up after a HARDRESET.

Free blocks

The other two stack block types are used to manage free space on the stack and as markers for book-keeping purposes. They are:

Ordinary Free block

```
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
|1 0 1 0 0 0 0 0|      0      |      Size      |
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
```

Guard block

```
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
|1 1 1 0 0 0 0 0|      0      |      Size      |
*-+-+-+-----*-+-+-+-----*-+-+-+-----*-+-+-+-----*
```

The only difference between Free blocks and Guard blocks is that microcode is guaranteed not to "use" Guard blocks. They are used at the end of stack regions. Size is in words.

Stack blocks may be discriminated by selecting the leftmost three bits of the flags into:

000b	Ordinary pointer (not a flag word)
100b	Basic frame
101b	Free block
110b	Frame extension
111b	Guard block

Function format

1. Function definition cells

Function definition cells are either:

Interpreted:

```
*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-
|      0      | ptr to definition (list/NIL) |
*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-
```

Compiled:

```
*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-
|1|F|Aty| 0 | ptr to definition (array block) |
*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-
```

Compiled definitions have 1 in the high order bit. The Aty field contains the 'argtype' of the function, and is used only by the CHECKAPPLY* opcode. The F bit is set if the corresponding NTSIZE of the definition is zero, i.e., the function has an empty name table. Undefined functions appear as an 'Interpreted' function with definition = NIL.

2. Function header

A compiled function begins with a function header and a name table, which contains variable names; function names and other constants are stored in-line in the code. The first 8 words of the function are called the function header. The function header is quadword aligned. Format:

```
*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-
|      STK      |      NA      |
*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-
|      PV      |      START    |
*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-
|-|-|Aty| 0 | function name |
*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-
|      NTSIZE   |      NLOCALS  | FVAROFFSET |
*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-+-+-+*-
```

NA The number of arguments this function expects. For LAMBDA* functions (arbitrary number of args, no adjustment) this field is -1.

PV Number of quadwords required for the PVAR region, less 1, i.e., (#PVars + #Fvars + 1)/2 - 1.

STK Amount of stack space required to call this function:

$$2 * (\max(\text{NA}, 0) + (\text{PV} + 1) * 2) + \text{FrameOverhead} + \text{MinExtraStackWords}.$$

max(NA, 0) is maximum number of extra (doubleword) pushes to fill in unsupplied arguments; FrameOverhead=12 (includes BF overhead); MinExtraStackWords is the number of words microcode requires before punting, currently 32 (for D0 HStack microcode).

START Initial PC of the function.

Aty Argtype of the function: 0 = LAMBDA, 2 = LAMBDA*, 1 = NLAMBDA, 3 = NLAMBDA*). ignored by microcode, but transcribed to the definition cell, where CHECKAPPLY* tests it.

NTSIZE Name table size (in words). This is a multiple of 4.

NLOCALS Number of PROG variables.

FVAROFFSET Offset (from start of header) of first FVAR in name table.

This is then followed by the specvar Name Table (not to be confused with the NAMETABLE field of the FX, which always points at something looking like a function header), which is actually two parallel tables.

The first table contains atom numbers ("value index"), terminated with enough zeros to fill out a quadword, but always at least one word of zero. Thus the size of the table in words is NTSIZE+1 rounded up to a multiple of 4. Note that in the special case where NTSIZE is 0, there is still a quad-word of zeros. The name table is arranged with PVAR names in reverse order of binding, IVAR names, and then FVAR names. Thus, when free variable lookup scans the name table in order, it will find the most recent bindings first.

The second table (which starts NTSIZE words beyond) contains entries of the form:

```
*-+-+-+---*--+-+-+---*
|vty|      0      |      offset      |
*--+-+-+---*--+-+-+---*
```

vty codes are as follows:

00	This name corresponds to an IVAR.
10	This name corresponds to a PVAR.
11	This name corresponds to a FVAR.

Offset is a zero-based doubleword offset from the start of the corresponding section of the BF or FX. Both PVAR and FVAR offsets are relative to PVAR, i.e., PVAR *n* is followed by FVAR *n*+1.

Free variable lookup scans the first table for a match of the "looked up variable". If a match is found, the word at the same offset in the second table is fetched to determine whether the value is in the IVAR section (vty=0), in the PVAR section (vty=2) or pointed to by the FVAR section (vty=3).

The regular name table is followed by the "localvar argument" name table, which is not visible to the microcode. It is in the same format as the regular name table, but lists names of arguments to the function that are declared LOCALVARS and hence would otherwise be invisible. This table is for the benefit of ARGLIST and the debugger.

Streams and File Devices

Edited 30 Nov 84, van Melle
Edited 19 June 86, Jellinek
Edited 22 September 86, Jellinek
Edited 3 October 86, Jellinek

An Interlisp Stream is an object of datatype `STREAM` that is capable of performing, at the least, sequential input and/or output of bytes. Some streams can do much more. Streams are used for access to open files, for writing to the display, for chatting to remote hosts, and whatever other uses people come up with. This document describes how one goes about defining a new device, the meanings of the record fields of the `STREAM` and `FDEV` datatypes, and anything else that seemed relevant at the time.

The implementation of Streams is strongly object-oriented. Every `STREAM` has a pointer to a device (the datatype `FDEV`), which contains a vector of functions to be called when certain operations are required of the stream. There can be many streams with the same device. In the object-oriented terms of `LOOPS`, one can think of the device as a *class*, which provides a set of *methods* that implement class operations, and the streams as *instances*. Devices and streams also have local state, which might be thought of as class and instance variables. Declarations for `STREAM` and `FDEV` can be obtained by loading `EXPORTS .ALL`.

`OPENSTREAM`, `CLOSEF`, `FORCEOUTPUT`, `READP`, `EOFP`, `GETEOFPTR`, `GETFILEINFO`, `SETFILEINFO`, `DIRECTORY`, `COPYBYTES`, `DELFILE`, `RENAMEFILE`, `FULLNAME` are some of the Lisp functions called by the programmer that ultimately turn into operations at the device level. The descriptions that follow sometimes allude to these functions, and knowledge of how they operate may occasionally give the reader additional clues as to how the device operations work.

Typically, some part of the operation is handled by the "generic" file system code, which then calls on the device to handle that part of the operation that is device-specific. For example, the function `OPENSTREAM` takes the name of the file it is to open and fills in host and directory defaults, and decides which device handles such a file. It then calls on the particular device to actually open the file. After the file is opened, the generic file system code registers the file on (`OPENP`). As another example, operations involving open streams first coerce non-streams (e.g. filenames) to open streams before calling the device-specific operation. With the advent of multiple streams per file, the last two examples will soon no longer obtain.

Devices

A device is an object of type `FDEV` (so named for historical reasons: "File Device"). The standard way to define a new device is to create such an object, by performing (**create** `FDEV` `--`), and then pass the newly created `FDEV` to the function `\DEFINEDEVICE`. `\DEFINEDEVICE` is the way a device "announces" itself to the generic file system.

(`\DEFINEDEVICE` *NAME* *DEV*)

[Function]

Installs device *DEV*, giving it the name *NAME*. *NAME* must be an uppercase litatom. The generic file system code makes use of the name to locate the device that is willing to deal with files whose full name begins *{NAME}*. It is permissible to have more than one name map to the same device; this effectively provides device synonyms. Devices are encouraged, however, to always create file names using the canonical device name, independent of what name was passed in.

NAME can be *NIL*, in which case the generic file system code never consults the device directly. However, its *EVENTFN* is still run around Lisp exits, and it can be used as the device for a stream created by nonstandard methods.

If a device never wants to be invoked by name, and has no interesting *EVENTFN* or *HOSTNAMEP* methods, then there is no need to ever register it with *\DEFINEDEVICE*.

(*\REMOVEDEVICE DEV*) [Function]

Removes device *DEV* from the list of known devices, as well as any name that maps to that device.

(*\GETDEVICEFROMNAME NAME NOERROR DONTCREATE*) [Function]

Returns the device associated with *NAME*. *NAME* can be a litatom or string; it is coerced to uppercase, and if it begins with an open brace, is assumed to be a file name, from which the host name is extracted. If no such device is known, attempts to find one by polling the *HOSTNAMEP* methods of all known devices (see below); if a device is still not found, causes a *FILE NOT FOUND* error unless *NOERROR* is true. If *DONTCREATE* is true, it never attempts to create a device, just returns an existing device if there is one, *NIL* otherwise.

The fields of an *FDEV* are divided up into informational fields and "methods".

<i>DEVICENAME</i>	A pointer field, the name of the device, standardly a litatom. Use of this field is largely up to the device, but it is usually selected to be the name that appears inside braces in filenames opened on this device. For devices that do not support the notion of named files, <i>DEVICENAME</i> can be anything that the implementor cares to use for debugging assistance.
<i>RESETABLE</i>	A flag, true if (<i>SETFILEPTR</i> stream 0) can be performed. Currently unused.
<i>RANDOMACCESSP</i>	True if the stream is randomly accessible, i.e., if <i>SETFILEPTR</i> works on this kind of stream.
<i>NODIRECTORIES</i>	True if files opened on this device do not (usually) have a directory as part of their name. The principal use for this is by the <i>CONN</i> command, which will not try to connect to the user's home directory if given a host only, e.g., <i>CONN {DSK}</i> .
<i>BUFFERED</i>	True if streams of this sort are buffered in a manner compatible with the microcoded versions of <i>BIN</i> and <i>BOUT</i> . More specifically, <i>BUFFERED</i> implies that the device implements the <i>GETNEXTBUFFER</i> method. See description of buffered streams.
<i>PAGEMAPPED</i>	True if this stream is implemented by the pagemapped functions. All pagemapped streams are also buffered, so if this flag is true, so should be <i>BUFFERED</i> . See description of pagemapped streams.
<i>FDBINABLE</i>	True if streams on this device obey the rules for microcoded <i>BIN</i> whenever such stream is open for input access.
<i>FDBOUTABLE</i>	True if streams on this device obey the rules for microcoded <i>BOUT</i> whenever such stream is open for output access. Currently unused, as the spec needs revision.
<i>FDEXTENDABLE</i>	Special kind of <i>FDBOUTABLE</i> . Currently unused, as the spec needs revision.

DEVICEINFO	A pointer to arbitrary device-specific information. The standard use for this is to hold local state specific to one of several similar devices that share methods. For example, the Dolphin disk provides a separate FDEV for each partition of the machine; the DEVICEINFO field of each has pointers to the partition's directory and other information specific to files on that partition only.
OPENFILELST	A list of all streams open on this device. Not required; it's provided for convenience only.

The following fields are all pointer fields, and contain functions for implementing various device operations. Not all devices need have all fields filled in; the required ones are listed first and so indicated. Some "required" fields have defaults specified in the FDEV (or STREAM) record declaration, so the implementor need not explicitly fill those fields if the default is reasonable. Each field is presented with its arguments, in the style of a function definition; of course, it is the contents of the field, not the field name, that is the function. Using object-oriented terminology, the occupants of these fields are referred to as "methods". For example, "the BIN method" means "the function that occupies the BIN field".

One of the arguments to each method is usually either the device itself, or a stream open on the device, so that the device (and hence its DEVICEINFO) is usually accessible to all these functions. Arguments that are file names or patterns or pieces of file names can be either litatoms or strings, and already have their host and/or directory parts appropriately filled in from the connected directory defaults. The device may assume that the host field of the file name is indeed a name that the device has said it implements (see HOSTNAMEP). "Full" file names returned by these functions (or stored in the FULLFILENAME field of a stream) should be litatoms, and at least in the current implementation should be all uppercase.

Fields required of every device:

(HOSTNAMEP *HOSTNAME* *DEVICE*)

Called by the generic file system code when presented with a host name for which there is as yet no device defined. The function should return non-NIL if it "recognizes" *HOSTNAME*. There are two ways in which this method is invoked:

- (1) To obtain a device for *HOSTNAME*, for example, so that a file can be opened on it. In this case, *DEVICE* is an already defined device (the one whose HOSTNAMEP method is being called), and the function should return either a new device, or T, meaning it is willing to take responsibility for this host name as well as any previous name under which the device was registered. In either case, the caller will install the returned device, or *DEVICE* if value was T, as the device to which *HOSTNAME* maps.
- (2) As a pure predicate. In this case, *DEVICE* is NIL, and the function need only return T or NIL, indicating whether it believes that *HOSTNAME* is the name of a host.

In practice, the HOSTNAMEP method need only take care of the first case, since that also takes care of the second case. The second case is provided so that the device need not be created until there is an actual use for it, should the device wish to avoid unnecessary work. In practice it is rare that anyone tests a host name without subsequently needing to have the device created in full.

There are basically three kinds of devices in the system as distinguished by their HOSTNAMEP methods.

- (1) Predefined devices with exactly one name, or strictly internal devices with no notion of name. For example, the CORE device always exists, and has exactly one name; the SPP device (a network byte stream) has no name (it supports no files directly). Such devices have a

HOSTNAMEP method of NIL—the only name they ever go by is the one they gave to \DEFINEDEVICE, if any. This is the default.

(2) Devices that don't know ahead of time what their name will be, but for which there might be many incarnations. This is the model for remote file servers. The standard way of handling this case is to define a dummy device that has only a HOSTNAMEP method, and no name. When the HOSTNAMEP method gets called with a name that the device knows it can service, it creates a device by that name. If given a name that is a synonym of another name, it might just return the existing device of the canonical name (using \GETDEVICEFROMNAME to find the right device). In either case, the HOSTNAMEP method of the new device is usually NIL—the original device is the only one that worries about creating new instances of this class of device.

(3) Like (2), but all the different names are handled by a single device, which takes care internally of the multiplexing among, say, different remote hosts. HOSTNAMEP returns T in this case. This is usually clumsier than (3), so discouraged.

(EVENTFN DEVICE EVENT)

Called around Lisp exits, to allow the device to do any necessary cleaning up, clearing of caches, disconnects with remote hosts, revalidation of files, etc. *EVENT* is one of the following litatoms: BEFORELOGOUT, BEFORESYSOUT, BEFOREMAKESYS, BEFORESAVEVM, AFTERLOGOUT, AFTERSYSOUT, AFTERMAKESYS, AFTERSAVEVM, AFTERDOSYSOUT, AFTERDOMAKESYS, AFTERDOSAVEVM. The AFTERxxx events are all run when Lisp is booted from a memory image that resulted from a LOGOUT, SYSOUT, etc. The AFTERDOxxx events run when continuing Lisp in the same incarnation following the SYSOUT, etc. (there is no such event for LOGOUT, of course). The "after" events are called in the same order in which the devices were defined; the "before" events in the reverse order.

For example, the BEFORELOGOUT event for the Leaf remote file server devices performs a FORCEOUTPUT on all its open files and then breaks the connection with the file server. The AFTERxxx events for the Leaf devices calls \REMOVEDEVICE on itself to flush any connection between the name and the server (since names and addresses can change over exit), and then revalidates all of the device's open files. The AFTERxxx events for the Dorado disk device rebuilds its cache of the disk's directory.

There are a few devices in the system that exist only for their EVENTFN. In most cases, a simpler way to tell the system you want something performed around exit is to add your event function to the list AROUNDEXITFNS instead of going to the expense of defining a device for it. There is yet another list, \SYSTEMCACHEVARS, for handling a more specialized "around exit" operation: every time Lisp is booted, each of the variables in the list \SYSTEMCACHEVARS is set to NIL.

The following are required of all named devices, that is, devices that map from some hostname to the device, upon which files might be opened or otherwise manipulated:

(DIRECTORYNAMEP HOST/DIR DEVICE)

True if *HOST/DIR* is a valid directory name on *DEVICE*. Function should ideally perform recognition as well, and return the "true" name. For example, given "{PHYLEX:<LISP>}" as argument, it might return {Phylex:PARC:Xerox}<Lisp>. *HOST/DIR* might include a subdirectory name. The device should attempt to tell the truth about whether the subdirectory exists or not, though this may not be possible for devices with fake subdirectories. Defaults to NIL, i.e., device supports no directories. Used by the command CONN and the function DIRECTORYNAMEP.

(OPENFILE NAME ACCESS RECOG PARAMETERS DEVICE)

Used to implement the `OPENFILE` and `OPENSTREAM` functions. Opens the file named *NAME* on this device for access *ACCESS*, returning a `STREAM`. The stream is usually on *DEVICE* (its `DEVICE` field is *DEVICE*), but is not required to be. The arguments *ACCESS*, *RECOG*, *PARAMETERS* are as with the `OPENFILE` function in the manual. Thus, if *NAME* does not include a version number, recognition is according to *RECOG*, which should be appropriately defaulted per *ACCESS* (INPUT implies OLD, OUTPUT implies NEW, BOTH implies OLD/NEW).

The argument *NAME* can also be a `STREAM`, which must be a closed stream. `OPENFILE` should "reopen" the stream. The value returned in this case may be a new stream (with the same name as the old), or the old stream (*NAME*) itself. It is likely that the specification will be changed at some point to require that the old stream itself be returned, suitably reopened.

The argument *PARAMETERS* is a list of pairs (*OPTION VALUE*). The most interesting *OPTIONS* are as follows:

TYPE	For new files, the type of the file (TEXT or BINARY). If this parameter is not specified, the value of the global variable <code>DEFAULTFILETYPE</code> (initially TEXT) should be used.
CREATIONDATE	For new files, the date of its creation. The device should use this if at all possible instead of letting the creation date default to the current date and time.
LENGTH	The intended length of the file, in bytes. This need not be accurate—it is only a hint that may allow smarter allocation. For example, if the device knows that it does not have room for a file of the specified length, it should immediately cause a <code>FILE SYSTEM RESOURCES EXCEEDED</code> error for the intended file.
DON'T.CHANGE.DATE	For old files being opened for access BOTH, don't change the creation date of the file. <i>ACCESS</i> = BOTH would normally imply that the content of the file is to change, and thus its creation date should be updated. Use of this parameter is a form of "cheating" to make it look as though the file had not changed. For example, the code that rewrites filemaps uses this parameter, since rewriting the filemap does not logically change the file's content.
SEQUENTIAL	If T, is a hint that the file will, or need, only be accessed sequentially, which may allow the device to open the file in a more efficient mode.

Any parameters that the device does not understand should be ignored, rather than be cause for an error. All devices are encouraged to support at least `TYPE` and `CREATIONDATE`.

The additional options `ENDOFSTREAMOP` and `BUFFERS` are handled by the generic file system code; specifying them is equivalent to calling `SETFILEINFO` (q.v.) immediately after the open.

Fine point about *ACCESS* = OUTPUT: this operation always produces a new, empty file, independent of whether its name is exactly the name of an existing file. That is, it replaces any old file by the same name. On opening, such a file has an end of file of zero. Of course, since *RECOG* defaults to NEW in this case, the name can only clash with an old file name if a version was explicitly specified, or *RECOG* is OLD or OLD/NEW. To open an old file for output but preserve its contents, i.e., only write over part of the file, one should open for *ACCESS* = BOTH (since to preserve the old contents one implicitly reads them).

Exception handling: If the desired file is not found, the `OPENFILE` method should return `NIL` rather than cause a `FILE NOT FOUND` error. This is so that the generic file system code can cause the error using the original file name, not the one packed with host and directory passed in to the `OPENFILE` method. The device should feel free to signal any other errors itself on failing to open the file, e.g., `FILE WON'T OPEN` for a busy file, `PROTECTION ERROR`, or `FILE SYSTEM RESOURCES EXCEEDED`. Ideally, this error should be signaled in a way that is resumable, i.e., so that a user could, in the break, take some action to remedy the condition and then type `OK` to continue. In most cases it suffices that all the internal functions below the `OPENFILE` be named with backslashes, so that the error code will choose to resume by reverting to the `OPENFILE` and trying again.

The device must keep track of the set of files open on it, for use in access-conflict detection, and as an assist to the user in closing "dropped" streams. This is done by the generic file system, employing the `REGISTERFILE` method, below.

(`CLOSEFILE STREAM`)

Closes `STREAM`. Performs all necessary "finalization" on `STREAM`, including doing a `FORCEOUTPUT` or equivalent if `STREAM` was open for output. De-registers `STREAM` from its device's list of open streams.

(`REOPENFILE NAME ACCESS RECOG PARAMETERS DEVICE OLDSTREAM`)

This is exactly like `OPENFILE`, except that it is called after `LOGOUT` (or other "after" events) by the device's `EVENTFN` on the name of any stream that was left open over exit. The idea is to maintain the illusion that the file really was open over `LOGOUT`, but check and make sure nothing changed. The generic file system code uses the `VALIDATION` field to test whether the file changed behind your back.

`OLDSTREAM` is the stream that was open before exit, and is supplied for the benefit of devices where there is no possibility that the file changed (e.g., `{CORE}`), so that they can just return `OLDSTREAM` directly. `OLDSTREAM` is also of use for those devices that have to cheat in order to maintain the illusion.

(`GETFILENAME NAME RECOG DEVICE`)

Performs "recognition" on `NAME`. That is, it returns the full name of the file that would be opened by `OPENFILE` in the indicated recognition mode, or `NIL` if the file is not found. It is not necessary that `OPENFILE` actually be capable of opening the file (there is no need to check protection, for example). Used by `INFILEP`, `OUTFILEP`, `FULLNAME`.

(`DELETEFILE NAME DEVICE`)

Deletes the file named `NAME`, returning its full name on success, `NIL` on failure. Recognition mode is implicitly `OLDEST`. Local devices, after recognizing the file, should make sure that it is not among the device's open files (open files cannot be deleted). This, `OPENFILE`, and `RENAMEFILE` are usually the only device methods that need to know anything about what files are open.

(`GENERATEFILES DEVICE PATTERN DESIREDPROPS OPTIONS`)

Enumerates files matching `PATTERN`. Returns a "file generator object" of the form (`NEXTFILEFN INFOFN . ArbitraryState`). This is described in more gory detail under **Directory Enumeration**.

(`RENAMEFILE OLD-DEVICE OLD-NAME NEW-DEVICE NEW-NAME`)

(A method of `OLD-DEVICE`.) Renames the file named `OLDNAME` on device `OLD-DEVICE` to have name `NEWNAME` on `NEW-DEVICE`. Returns the full name of the new file if successful, `NIL` if not. Recognition

mode is implicitly OLD for *OLDNAME*, NEW for *NEWNAME*. The generic file system code always invokes this method to implement the function RENAMEFILE. If *OLD-DEVICE* and *NEW-DEVICE* are not EQ, the RENAMEFILE method can call \GENERIC.RENAMEFILE to do the job; \GENERIC.RENAMEFILE is defined to copy *OLDNAME* to *NEWNAME* and then delete *OLDNAME*. Defaults to \GENERIC.RENAMEFILE. RENAMEFILE needs to check if OLD-NAME is open; only closed files can be renamed.

(OPENP *FILENAME ACCESS DEVICE*)

Returns all of *DEVICE*'s open streams with name *FILENAME* that are open in mode *ACCESS*. *FILENAME* and/or *ACCESS* may be NIL, which matches all names and access modes. If supplied, *FILENAME* should be a complete and exact filename, including a version number. The device field OPENFILELST provides a convenient place to store the open streams. \GENERIC.OPENP is a sample OPENP method that can be used in conjunction with \ADD-OPEN-STREAM and \DELETE-OPEN-STREAM; it consults the device's OPENFILELST.

(REGISTERFILE *DEVICE STREAM*)

Invoked by \OPENFILE, the REGISTERFILE method places the newly-opened stream on *DEVICE*'s list of open streams. Note that this need not be an actual list, but might be contained in the device's directory data structure or other convenient place. The function \ADD-OPEN-STREAM, though not the default, is a simple function that adds STREAM to *DEVICE*'s OPENFILELST.

(UNREGISTERFILE *DEVICE STREAM*)

Invoked by \CLOSEFILE, the UNREGISTERFILE method removes *STREAM* from *DEVICE*'s list of open streams. The revalidation code also uses this method to silently remove a device's invalid streams. Note that this need not be an actual list, but might be contained in the device's directory data structure.

The function \GENERIC-UNREGISTER-STREAM, though not the default, is a simple function that removes STREAM from *DEVICE*'s OPENFILELST. The revalidation code also uses this method to silently remove a device's invalid streams.

The following methods are invoked for open streams. They are all required:

(BIN *STREAM*)

Returns the next byte of input from *STREAM*, or takes the appropriate action if at end of file. Unless a device has a good reason not to, it should call (\EOF.ACTION *STREAM*) at end of file/stream.

The device BIN method is actually not used directly. Rather, every stream has a STRMBINFN field, which is the function actually applied to do the input. The STRMBINFN field could thus be used to fake a specialization of the device differing only in the BIN method. However, the typical use of STRMBINFN is to temporarily override the device default. In particular, setting a stream's access to INPUT or BOTH automatically sets the stream's STRMBINFN to be the device's BIN method; setting access to NIL or OUTPUT sets the STRMBINFN to be an error. This relieves the device's BIN method of any need to check the stream's access on every call to BIN. Some network streams temporarily set their STRMBINFN to be an input eater when they receive a "clear output" command.

Currently, all Interlisp-D streams have bytesize 8, so BIN always returns an 8-bit integer.

Calls to the function BIN are compiled into the BIN opcode, which runs in microcode on some machines if the requirements for it are met. More on this later.

(BOUT *STREAM BYTE*)

Outputs *BYTE* to *STREAM*. As with BIN, this method is not used directly. Rather, every stream has a STRMBOUTFN field, which is the function actually applied to do the output. Setting a stream's access to OUTPUT or BOTH automatically sets the stream's STRMBOUTFN to be the device's BOUT method.

There exists a BOUT opcode, but the design is incomplete.

(PEEKBIN *STREAM* *NOERRORFLG*)

Returns the next input byte from *STREAM*, but does not advance the stream pointer. Thus a subsequent PEEKBIN or BIN will return the same byte. At end of stream, the device should take eof action as with BIN, unless *NOERRORFLG* is true, in which case it should return NIL.

(READP *STREAM* *FLG*)

Returns true if input is available from *STREAM*, that is, if a BIN right now would succeed without waiting. Defaults to `\GENERIC.READP`, which uses EOF and PEEKBIN.

Roughly speaking, READP is the complement of EOF for streams that are not arriving in real time. It is interestingly different for network streams, or the keyboard.

FLG is a bit of cruft that not everyone pays attention to, and may be flushed at some point: if *FLG* is NIL, then READP should return NIL if the only input waiting is an end of line character.

(EOF *STREAM*)

Returns true when *STREAM* is "at end of file", i.e., a BIN would cause an end of file action to occur. Note that for a network stream, it is possible for both EOF and READP to be false simultaneously, viz. when there is no input waiting (buffered locally), but the remote end of the stream has not indicated that there is no more input.

There are some who call EOF on streams open only for output. This is a crock; output streams are always at end of file. But to avoid complaints, a device could return T for EOF on an output stream.

(BLOCKIN *STREAM* *BUFFER* *BYTEOFFSET* *NBYTES*)

Performs bulk input transfer: retrieves the next *NBYTES* bytes from *STREAM* and stores them in successive byte positions in *BUFFER* starting at *BYTEOFFSET*. Defaults to `\GENERIC.BINS`, which repeatedly calls BIN and `\PUTBASEBYTE`.

It is almost always the case that a device with a non-trivial BLOCKIN method can be made to be a Buffered device, thereby benefiting from other Buffered operations as well.

(BLOCKOUT *STREAM* *BUFFER* *BYTEOFFSET* *NBYTES*)

Performs bulk output transfer: outputs *NBYTES* bytes to *STREAM*, taking the bytes from *BUFFER* starting at *BYTEOFFSET*. Defaults to `\GENERIC.BOUTS`, which repeatedly calls `\GETBASEBYTE` and BOUT.

(FORCEOUTPUT *STREAM* *WAITFORFINISH*)

Forces to its ultimate destination any output buffered on *STREAM* but not yet sent. *WAITFORFINISH* means that the function should not return until it is confident that the output has reached its destination and been committed. Defaults to NIL, which is reasonable for unbuffered streams.

For example, for a network stream, FORCEOUTPUT sends the current packet being buffered up. For a buffered stream to the disk, FORCEOUTPUT writes out to the disk any "dirty" pages, and makes sure the file is in such a state that if the machine were booted after FORCEOUTPUT returns, that the file could be successfully reopened with no information lost.

(GETFILEINFO *NAME/STREAM* *ATTRIBUTE* *DEVICE*)

Returns the value of the specified *ATTRIBUTE* of *NAME/STREAM*, which can be an open Stream or the name of a (closed) file. Returns *NIL* for attributes it doesn't know about. It is considered good citizenship, though not absolutely required, to know about the following attributes:

LENGTH	Length of the stream/file in bytes. If the device's method returns <i>NIL</i> , but the stream is random access, the generic <i>GETFILEINFO</i> code tries the device's <i>GETEOFPTR</i> method instead.
SIZE	Length in pages, i.e., (<i>FOLDHI</i> length <i>BYTESPERPAGE</i>).
CREATIONDATE	Date when the file's contents were created, as a string. The creationdate does not change when a file is copied or renamed, only when it is changed.
WRITEDATE	Date when the file was written to its current place of storage.
READDATE	Date when the file was last read.
ICREATIONDATE, IWRTEDATE, IREADDATE	The creation, write and read dates as integers, such as from the function <i>IDATE</i> .
TYPE	Type of the contents: <i>TEXT</i> for files that contain only "text" (generally meaning 7-bit <i>ascii</i>), <i>BINARY</i> for all others. <i>NIL</i> means unknown.
AUTHOR	Name of the user who created the file (a string).

The following "generic" attributes are generally handled by the generic side of *GETFILEINFO* if the device's *GETFILEINFO* method returns *NIL*:

EOL	The end of line convention of the stream (<i>CR</i> , <i>CRLF</i> or <i>LF</i>).
BUFFERS	The number of pagemap buffers for use by the stream (see description of <i>MAXBUFFERS</i> field of pagemapped streams).
ENDOFSTREAMOP	Action to take on any attempt to read beyond the end of file. This is a function of one argument, the stream. The function can cause an error, or return a value, which is interpreted as a value to return from <i>BIN</i> . The default <i>ENDOFSTREAMOP</i> causes an <i>END OF FILE</i> error.
ACCESS	An atom describing the access mode of the stream (<i>INPUT</i> , <i>OUTPUT</i> , etc). This is so generic that it is handled before the device's method ever sees it.
BYTESIZE, OPENBYTESIZE	The size of bytes transmitted on the stream. Always 8 these days.

(*SETFILEINFO NAME/STREAM ATTRIBUTE VALUE DEVICE*)

Sets the value of the specified *ATTRIBUTE* of *NAME/STREAM* to be *VALUE*. Returns *T* if successful, *NIL* if unsuccessful, or for attributes it doesn't know about.

It is not generally required that *SETFILEINFO* recognize any attributes at all—*NILL* is a perfectly good filler for this slot. Most devices recognize no more than *TYPE* and *CREATIONDATE* (*ICREATIONDATE*), and even those are not very important, as most applications set those attributes in the *PARAMETERS* argument to *OPENFILE* when creating a file.

ATTRIBUTE = *LENGTH* implies actually truncating (or lengthening) the file; however, the *SETFILEINFO* need not handle this itself—if it returns *NIL*, then the generic file system will attempt to use the *SETEOFPTR* method instead.

The following operations are only required of random access streams. They default to the function `\IS.NOT.RANDACCESSP`, which causes a "Stream is not randaccessp" error when called.

`(GETFILEPTR STREAM)`

Returns the current file pointer (byte position) in *STREAM*. The file pointer is zero when the stream is opened (except for *ACCESS = APPEND*), and is incremented by one for each byte read.

Although this operation is only absolutely required for random access streams, it is desirable to supply it for other streams where possible. For example, when reading a file sequentially through PupFtp, the stream can count the bytes as they go by and thus give an accurate value for `GETFILEPTR`. If a stream has no idea at all of position, it can make its `GETFILEPTR` be the function `ZERO` and thereby at least avoid breaks from code that calls `GETFILEPTR` carelessly.

`(GETEOFPTR STREAM)`

Returns the file pointer of the end of *STREAM*, i.e., the file pointer that `GETFILEPTR` would return after the last byte of *STREAM* is read. Same as the `LENGTH` attribute for a stream that represents a file. Of course, non-random access streams may have no idea where the end is, and causing a non-randaccessp error is perfectly acceptable.

`(SETFILEPTR STREAM BYTENUMBER)`

Sets the file pointer of *STREAM* to be *BYTENUMBER*. The special value *BYTENUMBER* = -1 means the end of the stream; other negative values are illegal.

`SETFILEPTR` beyond the end of the stream is permissible, but it has no immediate effect beyond changing the logical file pointer. Attempting to then `BIN` causes an EOF error. Attempting to `BOUT` (for a file open for write) should extend the file, so that its eof is immediately beyond the newly `BOUT`d byte.

As with `GETFILEPTR`, there is no requirement that this work on non-random access streams, and it may be completely impossible on some of them. However, for those non-random access streams that perform `GETFILEPTR`, it is possible to fake `SETFILEPTR` for values larger than the current file pointer by skipping some number of bytes in the file, e.g., by performing `(RPTQ (DIFFERENCE BYTENUMBER (GETFILEPTR STREAM)) (BIN STREAM))`. There are some applications for which forward `SETFILEPTR` is all the random access that is actually required, so it is nice to be able to accommodate such applications.

`(BACKFILEPTR STREAM)`

Backs up the file pointer in *STREAM* by one byte. Functionally the same as `(SETFILEPTR STREAM (SUB1 (GETFILEPTR STREAM)))`, but may be possible on non-random access streams by maintaining a one-character buffer, which is all the backing up this operation is formally required to perform. I believe the main use for this is in `READ`, which needs to back up the stream one character when, for example, it reads a break character terminating an atom.

`(SETEOFPTR STREAM LENGTH)`

Changes the length of *STREAM* to be *LENGTH*, i.e., "sets" its end of file pointer. This may require lengthening or truncating the file. Used by the function `\SETEOFPTR` and by `SETFILEINFO` for attribute `LENGTH` when the device's `SETFILEINFO` method doesn't handle it.

The following three fields are place holders for possible future extensions. These fields are not currently used at all:

(LASTC *STREAM*)

Returns the last character read from *STREAM*, i.e., the last byte that was BINed, as a character. LASTC is currently implemented via BACKFILEPTR.

(FREEPAGECOUNT *HOST/DIR DEVICE*)

Intended use is to return the number of free pages on *HOST/DIR*. May be folded into a general GET/SET device/directory info operation.

(MAKEDIRECTORY *HOST/DIR DEVICE*)

Intended use is to create a new directory *HOST/DIR*.

The remaining fields in the FDEV are for buffered and page-mapped streams, and are ignored for non-buffered devices. These fields are described in separate sections.

Streams

The following fields are used by all streams:

DEVICE	Pointer to this stream's FDEV.
FULLFILENAME	"Full" name by which this file is known to the user. Should be an uppercase litatom, fully qualified so that giving the same name back to the file system should produce the same file (to the extent that the device can support such uniqueness). Is NIL for unnamed streams.
FULLNAME	Access field. Is the same as FULLFILENAME, unless that is NIL, in which case it is the stream itself. This avoids the circularity that would result if the FULLFILENAME field contained the stream datum.
NAMEDP	Access field. Is T if the streams is named, i.e., its FULLFILENAME is non-NIL.
ACCESSBITS	Contains a numeric code describing what access mode the file is open for: there are read, write and append bits. This field is usually accessed indirectly via the ACCESS field. However, there are macros for referring to particular types of access using more efficient bit test operations:
	(OPENED <i>STREAM</i>) ACCESS is not NIL.
	(READABLE <i>STREAM</i>) Read bit is on: ACCESS is INPUT or BOTH.
	(READONLY <i>STREAM</i>) Only the read bit is on: ACCESS is INPUT.
	(APPENDABLE <i>STREAM</i>) Append bit is on: ACCESS is OUTPUT, BOTH or APPEND.
	(APPENDONLY <i>STREAM</i>) Only the append bit is on: ACCESS is APPEND.
	(DIRTYABLE <i>STREAM</i>) Append or write bit is on: ACCESS is OUTPUT, BOTH or APPEND. Yes, this is operationally the same as APPENDABLE, given the four possible values of ACCESS.
	(OVERWRITEABLE <i>STREAM</i>) Write bit is on: ACCESS is OUTPUT or BOTH.

	(WRITEABLE <i>STREAM</i>)	Write bit is on, or append bit is on and file is at EOF. Avoid using this one, it's a little strange.
ACCESS		Access field for referring to the ACCESSBITS field symbolically. Its value is one of the legal values of the <i>ACCESS</i> argument to <i>OPENFILE</i> : INPUT, OUTPUT, BOTH, APPEND; or NIL when the stream is closed. Replacing this field has the side effect of setting the BINABLE, BOUTABLE, STRMBINFN and STRMBOUTFN fields appropriately (from the corresponding device fields, or to values consistent with no access).
USERCLOSEABLE		Flag, true if the stream can be closed by <i>CLOSEF</i> . Default is T, but is NIL for such things as dribble files and the terminal.
USERVISIBLE		Flag, true if the stream is to be listed in the result of (<i>OPENP</i>) . Default is T, but is NIL for such things as dribble files and the terminal.
BINABLE		True if BIN microcode can be used. Normally set automatically from FDBINABLE when input access is set.
BOUTABLE		True if BOUT microcode can be used. Normally set automatically from FDBOUTABLE when output access is set.
EXTENDABLE		True if BOUT can extend the buffer when <i>COFFSET</i> reaches <i>CBUFSIZE</i> . Obsolete.
STRMBINFN		Function called by BIN. This is normally set indirectly as a side effect of setting the <i>ACCESS</i> field. Setting <i>ACCESS</i> to an input access (INPUT or BOTH) sets the STRMBINFN to be the stream's device's BIN method. Setting to any other access sets the STRMBINFN to be a "file not open" trap.
STRMBOUTFN		Function called by BOUT. As with STRMBINFN, this is normally set indirectly (from the device's BOUT method) as a side effect of setting the <i>ACCESS</i> field.
OUTCHARFN		Function called to output a single byte. This is like STRMBOUTFN, except for being one level higher: it is intended for text output. Hence, this function should convert (<i>CHARCODE EOL</i>) into the stream's actual end of line sequence, and should adjust <i>CHARPOSITION</i> appropriately before invoking the stream's STRMBOUTFN to actually put the character. Defaults to <i>\FILEOUTCHARFN</i> . The <i>OUTCHARFN</i> for the display additionally worries about such things as <i>ECHOCONTROL</i> .
CHARPOSITION		Current horizontal character position in the stream. Incremented (and reset to zero) by <i>OUTCHARFN</i> . Used by the function <i>POSITION</i> .
LINELENGTH		Maximum line length of the stream, in characters. Used by the function <i>LINELENGTH</i> . Defaults (at creation time) to the value of the global variable <i>FILELINELENGTH</i> .
EOLCONVENTION		The stream's end of line convention: the manner in which "end of line" is encoded on this stream. That is, output of an end of line (function <i>TERPRI</i>) produces the stream's end of line sequence, and on input, the stream's end of line sequence is converted to (<i>CHARCODE EOL</i>) by <i>READC</i> . This is not necessarily the same as the way that end of line is encoded in the actual file written by, say, a file server. For example, Lisp might open a stream to a Tenex file server with <i>EOLCONVENTION</i> of CR, while the server

might choose to take each of the CRs in the stream and actually store a CR, LF sequence in the physical file.

The convention is encoded as a two-bit field; the constants CR.EOLC, LF.EOLC, CRLF.EOLC can be used to refer to the currently known values symbolically. Default in Interlisp-D is CR.EOLC.

ENDOFSTREAMOP	Function of one argument (the stream) called when an attempt to read beyond the end of file occurs. If this function returns something, it should be interpreted as a value to return from BIN (the value T is currently prohibited). Defaults to \EOSERROR, which causes an END OF FILE error.
VALIDATION	Pointer field, some compact encoding of the state of the file such that if the file's content changes, the VALIDATION changes. The file's ICREATIONDATE attribute usually works well enough. The only use for this field is to check whether the file changed over LOGOUT, etc.—if the VALIDATION of the stream returned from REOPENFILE is EQUAL to the VALIDATION of the stream open before LOGOUT, the stream is assumed to be unchanged. This will probably be the sole concern of the device when we go to multiple streams per file.
BYTESIZE	Byte size of the file, i.e., what BIN and BOUT traffic in. Defaults to 8. This field is not used by many; there are probably a lot of things that won't work if the byte size is not 8.
OTHERPROPS	List in property list format used by the function STREAMPROP. Analogous to WINDOWPROP, etc.
IMAGEOPS	Image operations vector (object of type IMAGEOPS) for use of device-independent graphics operations, such as DSPXPOSITION, DSPFONT. Defaults to \NOIMAGEOPS, a vector suitably defined for non-display devices. See the implementors' manual chapter Device-Independent Graphics .
IMAGEDATA	Device-dependent data for use by IMAGEOPS.
REVALIDATEFLG	Flag. The standard use of this flag is to solve a problem with correctly maintaining the creation date. The problem is that the definition of "creation date" is that the creation date changes whenever the contents of the file change. If followed literally, this would mean, for example, that ever time you wrote out a page of a {DSK} file, you would also have to rewrite its leader page with a new creation date. However, it suffices in practice to only change the creation date when it would matter, i.e., when there would be any possibility of some agent other than the currently running Lisp to see the change. Usually, this means the only time to worry about is when the Lisp vmem is saved and a file that was open before the save is written to again afterwards. Thus, the use of this flag (for those devices that care) is as follows: the device's BEFORExxx events set this flag true for any streams open on the device. Then, whenever the device is about to do something that would change the file's content, e.g., write out a new page, it first tests REVALIDATEFLG. If the flag is true, it updates the file's creation date and clears the flag.
NONDEFAULTDATEFLG	Flag. Standard use is in conjunction with REVALIDATEFLG, to mark a file that was opened in a way that the user constrained the creation date of the file (e.g., the PARAMETERS argument to OPENFILE included an explicit creation date, or the option DON'T.CHANGE.DATE).

F1, F2, F3, F4, F5	Pointer fields for private use by the stream, to maintain stream-specific state of concern only to the device. Stream clients that wish to hang information on a stream without regard to what kind of stream it is should use the function <code>STREAMPROP</code> .
FW6, FW7, FW8, FW9	16-bit word fields for private use by the stream.
DIRTYBITS	Obsolete.
EXTRASTREAMOP	?

Buffered Streams

Buffered streams are ones that constrain themselves to obey a set of conventions that make it easy for an agent (e.g., microcode) to perform input or output on the stream without knowing about the details of the stream's physical i/o. The stream maintains a "current buffer" and two indices into that buffer, the offset of the next byte, and the offset of the end of the buffer. As long as the former index is less than the latter, the stream guarantees that the bytes in the buffer between those indices are the true contents of the file/stream starting at the current file pointer. Advancing the first index effectively advances the file pointer. When it reaches the second index, a stream-specific operation is called to "refill" the buffer.

The following fields are used by buffered streams:

COFFSET	Byte offset in the buffer <code>CBUFPTR</code> of the next <code>BIN</code> or <code>BOUT</code> .
CBUFSIZE	"Size" of the current buffer, i.e., byte offset that is one beyond the last byte.
CBUFMAXSIZE	For output, the maximum size the buffer can be written to. If <code>COFFSET</code> reaches <code>CBUFSIZE</code> , but <code>CBUFSIZE</code> is less than <code>CBUFMAXSIZE</code> , then the buffer can be extended.
CBUFPTR	Pointer to current buffer. Must be valid if <code>COFFSET</code> is less than <code>CBUFSIZE</code> and <code>BINABLE</code> or <code>BOUTABLE</code> is true. It is not necessary that this "buffer" be anything other than some chunk of memory, a portion of which contains interesting data. Thus, the bytes from offset <code>COFFSET</code> to <code>CBUFSIZE</code> must be valid, but <code>COFFSET</code> need not start at zero, nor need <code>CBUFSIZE</code> or <code>CBUFMAXSIZE</code> coincide with the end of the underlying structure.
CBUFDIRTY	Flag, true if current buffer has been written to.

In general, the device has sole responsibility for setting `CBUFSIZE`, `CBUFMAXSIZE`, and `CBUFPTR`; generic code does not touch those. The fields `COFFSET` and `CBUFDIRTY` can be changed by generic stream clients as well as by device-specific code. For example, code that simulates a `BIN` increments `COFFSET`; code that writes directly to the stream's buffer sets `CBUFDIRTY` true.

The following methods are defined for devices implementing buffered streams:

(<code>GETNEXTBUFFER</code> <i>STREAM</i> <i>WHATFOR</i> <i>NOERRORFLG</i>)	[Device method]
Called when <i>STREAM</i> needs to have its buffer fixed, i.e., the state of <i>STREAM</i> is such that <code>BIN</code> (<i>WHATFOR</i> = <code>READ</code>) or <code>BOUT</code> (<i>WHATFOR</i> = <code>WRITE</code>) cannot proceed. This method should do whatever is necessary to allow the operation to proceed. This typically includes disposing of the current buffer somehow (if	

GETNEXTBUFFER was invoked because the buffer was exhausted), and fetching a new buffer consistent with *STREAM*'s current position.

In the case of *WHATFOR* = READ, GETNEXTBUFFER returns T on success, i.e., if *STREAM* is not at end of file. When *STREAM* is at end of file, GETNEXTBUFFER should take standard end of stream action, returning whatever \EOF.ACTION returns (if anything). However, if *NOERRORFLG* is true, GETNEXTBUFFER should just return NIL immediately.

(RELEASEBUFFER *STREAM BUFFER*)

[Device method]

Performs any device-specific operation required when *BUFFER*, which is the current value of *STREAM*'s CBUFPTR field, is "released" (when the CBUFPTR field is replaced). This is used so that different pagemap-like devices can share certain code. For example, in the case of pagemapped streams, RELEASEBUFFER marks the buffer dirty in the case that the stream's CBUFDIRTY field has been set.

This method is not currently used.

The functions \BUFFERED.BIN, \BUFFERED.PEEKBIN, \BUFFERED.BOUT, \BUFFERED.BINS and \BUFFERED.BOUTS are supplied for use by buffered streams; they are standardly used to implement the BIN, PEEKBIN, BOUT, BLOCKIN and BLOCKOUT device methods. In addition, the function COPYBYTES, when presented with a source stream that is buffered, utilizes the GETNEXTBUFFER method to efficiently copy bytes to the destination a buffer-full at a time.

Pagemapped Streams

Pagemapped streams are a particular kind of random access Buffered stream that buffers its data in units of pages. The device provides methods that read or write data in units of pages, while system-supplied Pagemapped functions handle the responsibilities of a Buffered stream, as well as managing the file pointer for random access. In general, a stream can have several pages of a file buffered at a time, allowing the code to make some effort to make efficient use of multi-paged transfers where applicable.

To create a pagemapped device, create an FDEV, fill in the necessary private fields, then call the following function:

(\MAKE.PMAP.DEVICE *DEVICE*)

[Function]

Fills in fields in the device appropriate for pagemapped devices, and returns the updated device. The fields it fills are the flag fields FDBINABLE, FDBOUTABLE, RESETABLE, RANDOMACCESSP, PAGEMAPPED, BUFFERED (all true), and the methods BIN, BOUT, PEEKBIN, BLOCKIN, BLOCKOUT, READP, EOF, GETFILEPTR, BACKFILEPTR, SETFILEPTR, GETEOF, SETEOF, GETNEXTBUFFER and FORCEOUTPUT.

A Pagemapped device is required to supply the following methods (in addition to those required of all devices and not filled in by \MAKE.PMAP.DEVICE):

(READPAGES *STREAM FIRSTPAGE# BUFFERS*)

[Device method]

Causes pages of *STREAM* to be read into *BUFFERS*. The first page read is *FIRSTPAGE#* (zero for the first page of the file). *BUFFERS* is either a single page-sized buffer (a VMEMPAGEP), in which case exactly one page is read, or it is a list of such buffers. READPAGES returns the total number of bytes read. If the last page read is not a full page, READPAGES should zero out the rest of its buffer. READPAGES can assume that the buffers are page-aligned, although they need not be consecutive.

(WRITEPAGES *STREAM* *FIRSTPAGE#* *BUFFERS*)

[Device method]

Writes data from *BUFFERS* out to *STREAM*. The first page written is *FIRSTPAGE#*. *BUFFERS* is as with *READPAGES*.

Neither *READPAGES* nor *WRITEPAGES* affects *STREAM*'s file pointer or end of file; those are managed by higher-level pagemapped routines. *WRITEPAGES* might, however, want to look at *STREAM*'s *EPAGE* and *EOFFSET* fields if it needs to take any special action around the end of the file. It is possible, for no particularly good reason, for *READPAGES* to get called for a page beyond the end of file; in fact, this standardly happens when writing a new file. The *READPAGES* method in this case should just clear the buffer and return zero.

(TRUNCATEFILE *STREAM* *PAGE#* *OFFSET*)

[Device method]

Truncates *STREAM* so that its end of file is *PAGE#*, *OFFSET*, which should be defaulted to *STREAM*'s *EPAGE* and *EOFFSET*. Can be used to either shorten or lengthen a file; if lengthening, the file should be padded with nulls. Used by `\PAGED.SETEOFPTR` and `\PAGED.FORCEOUTPUT`. As of this writing there are still bugs in this code in certain funny cases, such as when you `SETFILEPTR` beyond eof and then `BOUT`.

The following fields of a stream are meaningful for a pagemapped device. The generic pagemapped codes maintain them as operations on the file are performed, but they should all be initialized appropriately by the device's *OPENFILE* method:

CPAGE	For pagemapped streams, the current page position in the stream. Together with <i>COFFSET</i> , this constitutes the stream's file pointer. The device's <i>OPENFILE</i> method should set <i>CPAGE</i> and <i>COFFSET</i> to zero, except for files opened with access <i>APPEND</i> , in which case they should be set to the end of file.
EPAGE, EOFFSET	For pagemapped files, the page and byte offset of the end of file. Note that this is the <i>logical</i> end of the file; it need have nothing to do with the physical end of file, except that when a file is closed, the device should see to it that its logical and physical EOFs are the same (normally seen to by the <i>TRUNCATEFILE</i> inside of <code>\CLEARMAP</code> , below). In fact, as a typical file is being written, <i>EPAGE</i> tends to stay several pages ahead of the physical end of file by virtue of the fact that pages are being buffered before being written out.
BUFFS	For pagemapped streams, a pointer to the stream's <i>BUFFER</i> chain. Initially <i>NIL</i> (no buffers allocated). The device usually has no direct interest in this field.
MAXBUFFERS	For pagemapped streams, the maximum number of buffers desired in the stream's <i>BUFFS</i> chain. If the code needs another buffer and there are already <i>MAXBUFFERS</i> buffers, it will try to recycle the least recently referenced buffer. Defaults to <code>\STREAM.DEFAULT.MAXBUFFERS</code> . The user can change this field for an open stream by calling <i>SETFILEINFO</i> with attribute <i>BUFFERS</i> .
MULTIBUFFERHINT	Flag. For pagemapped streams, is a hint to the pagemap code that the device prefers to transfer data more than one buffer at a time. If this flag is true, the pagemap code tries to write out (<i>WRITEPAGES</i>) more than one buffer at a time when the opportunity arises. A similar improvement is planned, but not implemented, for reading multiple buffers at a time.

The following functions are of use for pagemapped devices:

(`\PAGED.FORCEOUTPUT` *STREAM* *WAITFORFINISH*)

[Function]

This function implements the `FORCEOUTPUT` method for pagemapped streams: it causes any dirty pages to be written out (using `WRITEPAGES`), then calls the `TRUNCATEFILE` method to set the end of file.

This function is normally installed as the `FORCEOUTPUT` method by the function `\MAKE.PMAP.DEVICE`. However, the device can override this default (by supplying its own function in that field), in which case it might want to call the function `\PAGED.FORCEOUTPUT` explicitly as part of its more comprehensive `FORCEOUTPUT` method.

There is an unpleasantness in the implementation of pagemapped devices that stems from the fact that originally all devices (the few that existed in the distant past) were made to support the `PMAP` package, a means whereby a programmer could get direct access to the buffers of a file, much as one can with the `PMAP JSYS` in Tenex. As a result, the buffers used by pagemapped streams are set up in a special manner so that the garbage collector can tell when the user no longer has access to a `PMAP` buffer. The `PMAP` package is being phased out.

This is all exceedingly crufty, and is of little concern to the device implementer, except for the fact that it requires that the buffers be explicitly released when a stream is closed; the buffers are not automatically collected when the stream is dropped.

(`FORGETPAGES STREAM FROMPAGE TOPAGE`) [Function]

"Forgets" pages `FROMPAGE` thru `TOPAGE` of `STREAM`; i.e., removes those pages from the set of pages being currently buffered, and frees the buffers they were occupying. If `FROMPAGE = TOPAGE = NIL`, forgets all pages, and releases all of `STREAM`'s buffers.

(`\CLEARMAP STREAM`) [Function]

Performs a `FORCEOUTPUT` (if `STREAM` is open for output) followed by a `FORGETPAGES`. This is the standard action that should be taken by a pagemapped stream's `CLOSEFILE` method.

Directory Enumeration

This section describes how directory enumeration works—what you need to know in order to implement the `GENERATEFILES` device method, and what you need to know as a programmer trying to enumerate a directory via anything more elaborate than the function `DIRECTORY`.

The general idea is that the directory enumeration code is given a pattern, and it returns a generator that, each time it is poked, returns another file name matching the pattern. In addition, the generator provides a handle for getting file attributes of each enumerated file. This second handle is important for efficiency: although one could just take the file name given by the enumerator and pass it to `GETFILEINFO`, the device, in the course of enumeration, usually has its fingers on the file closely enough that it need not perform the second directory lookup that a `GETFILEINFO` out of the blue would require. The caller of the directory enumeration code specifies ahead of time which, if any, attributes will be required (a necessity for most file server implementations).

Information for device implementors. A *file generator* is an object represented as a list described by the record `FILEGENOBJ`, exported from `FILEIO`:

```
(RECORD FILEGENOBJ (NEXTFILEFN FILEINFOFN . GENFILESTATE))
```

`NEXTFILEFN` and `FILEINFOFN` are functions of the device's choosing that when called will return the next file, and attributes for that file. `GENFILESTATE` is arbitrary state maintained by the generator. With that as background, here are the pieces of directory enumeration:

(GENERATEFILES *DEVICE PATTERN DESIREDPROPS OPTIONS*)

[Device method]

Returns a generator that enumerates files matching *PATTERN*, which is a string that has host and directories suitably filled in from defaults, and may contain the pattern character "*" to match an arbitrary number of characters. *DESIREDPROPS* is a list of file attributes that may be requested during the enumeration; they must be valid *ATTRIBUTE* arguments to *GETFILEINFO*. *OPTIONS* is a list of options to the enumeration, chosen from among the following:

SORT	The files should be enumerated in sorted order. If this option is not specified, the device is free to enumerate files in any convenient order.
	There is some question as to whether files should be enumerated lowest version first (as IFS's do) or highest version first (as Twenex does). I prefer the latter, but given servers that do the former, we currently make no requirement about version order.
RESETLST	Informs the enumerator that the enumeration context is surrounded by a RESETLST, so that it may perform RESETSAVES to clean up after itself if the enumeration is aborted. Cleaning up can be a very messy business without this information about the scope of the enumeration, so all callers of \GENERATEFILES are strongly encouraged to provide it.

GENERATEFILES should return a file generator with a suitable NEXTFILEFN and FILEINFOFN.

Fine point about missing fields in the pattern: null fields in *PATTERN* match only files for which the corresponding field is null. A null version is interpreted as highest. Thus,

DIR *. = DIR *. * = DIR *. * ; *	enumerates everything.
DIR *. = DIR *. ; *	enumerates all versions of files with null extension.
DIR *. ;	enumerates highest version of files with null extension.
DIR *. * ;	enumerates highest version of everything.

It is difficult for some devices to enumerate only highest version of files; there are several devices in the system that treat a null version the same as version *. However, every device should try its best. With some work, any device that can enumerate all versions can enumerate just highest version if it enumerates in sorted order and uses perhaps a little lookahead to assure that any name it returns is the one of highest version.

(NEXTFILEFN *GENFILESTATE NAMEONLY*)

[File Generator Component]

Generates the next file, returning its name as a string, or NIL if the generator is exhausted. *GENFILESTATE* is the state component of the file generator returned from GENERATEFILES. *NAMEONLY* means that the caller is only interested in the file's Name.Ext fields, not the full file name (and no more than one version of the file need be enumerated); however, it is always permissible to return the full file name. The *NAMEONLY* option is used by SPELLFILE.

(FILEINFOFN *GENFILESTATE ATTRIBUTE*)

[File Generator Component]

Returns the value of the *ATTRIBUTE* property of the file most recently generated by the NEXTFILEFN, i.e., effectively (GETFILEINFO latest-name *ATTRIBUTE*), but hopefully much faster. *ATTRIBUTE* must have been a member of the *DESIREDPROPS* argument to GENERATEFILES.

Not all device implementors are enthused about implementing a pattern matcher for file names. The following functions are provided to help out:

(*DIRECTORY.MATCH.SETUP PATTERN*) [Function]

Accepts as *PATTERN* a file name string such as passed to *GENERATEFILES*. Returns an object suitable as a filter to *DIRECTORY.MATCH*.

(*DIRECTORY.MATCH FILTER TESTNAME*) [Function]

Matches *TESTNAME*, a file name, against *FILTER*, the object returned from *DIRECTORY.MATCH.SETUP*. Returns true if *TESTNAME* matches the pattern, false if not. The match is case-insensitive.

(*\NULLFILEGENERATOR*) [Function]

Returns a file generator that produces no files.

(*\GENERATENOFILES DEVICE PATTERN DESIREDPROPS OPTIONS*) [Function]

Returns a "stupid" file generator for devices that don't know how to enumerate in general. If *PATTERN* contains no wildcards, but names a file that is *INFILEP*, then the generator produces exactly that file. If *PATTERN* contains a wildcard in the version field, it uses *GETFILENAME* to laboriously generate all the versions of the file. In all other cases, *\GENERATENOFILES* returns a null file generator.

Information for clients of device enumeration. The following functions make up the "public" interface to directory enumeration:

(*\GENERATEFILES PATTERN DESIREDPROPS OPTIONS*) [Function]

Returns a file generator object for enumerating the files matching *PATTERN*. *PATTERN* is expanded by adding the default host and/or directory if appropriate. See description of the *GENERATEFILES* method for description of *DESIREDPROPS* and *OPTIONS*.

(*\GENERATENEXTFILE GENERATOR NAMEONLY*) [Function]

Returns the next file, as a string. *GENERATOR* is the object returned from *\GENERATEFILES*; *NAMEONLY* indicates caller does not require that the full name be returned, but that the name and extension are sufficient.

(*\GENERATEFILEINFO GENERATOR ATTRIBUTE*) [Function]

Returns the value of the *ATTRIBUTE* property of the file most recently generated by *\GENERATENEXTFILE*, i.e., effectively (*GETFILEINFO latest-name ATTRIBUTE*). *ATTRIBUTE* must have been a member of the *DESIREDPROPS* argument to *\GENERATEFILES*.

(*DIRECTORY.FILL.PATTERN PATTERN DEFAULTTEXT DEFAULTVERS*) [Function]

This function is used to fill in defaults in *PATTERN* before passing it to *\GENERATEFILES*. If *PATTERN* does not include an extension or version, but those fields are not explicitly omitted (e.g., "FOO", but not "FOO."; "FOO.BAR", but not "FOO.BAR;"), they are filled in with *DEFAULTTEXT* and *DEFAULTVERS*, which themselves default to "*". This function is used by the *DIR* command, and should probably be used by any code that takes a user-supplied pattern and enumerates files from it.

Streams and File Devices

Edited 30 Nov 84, van Melle

An Interlisp Stream is an object of datatype `STREAM` that is capable of performing, at the least, sequential input and/or output of bytes. Some streams can do much more. Streams are used for access to open files, for writing to the display, for chatting to remote hosts, and whatever other uses people come up with. This document describes how one goes about defining a new device, the meanings of the record fields of the `STREAM` and `FDEV` datatypes, and anything else that seemed relevant at the time.

The implementation of Streams is strongly object-oriented. Every `STREAM` has a pointer to a device (the datatype `FDEV`), which contains a vector of functions to be called when certain operations are required of the stream. There can be many streams with the same device. In the object-oriented terms of `LOOPS`, one can think of the device as a *class*, which provides a set of *methods* that implement class operations, and the streams as *instances*. Devices and streams also have local state, which might be thought of as class and instance variables. Declarations for `STREAM` and `FDEV` can be obtained by loading `EXPORTS.ALL`.

`OPENSTREAM`, `CLOSEF`, `FORCEOUTPUT`, `READP`, `EOFP`, `GETEOFPTR`, `GETFILEINFO`, `SETFILEINFO`, `DIRECTORY`, `COPYBYTES`, `DELFILE`, `RENAMEFILE`, `FULLNAME` are some of the Lisp functions called by the programmer that ultimately turn into operations at the device level. The descriptions that follow sometimes allude to these functions, and knowledge of how they operate may occasionally give the reader additional clues as to how the device operations work.

Typically, some part of the operation is handled by the "generic" file system code, which then calls on the device to handle that part of the operation that is device-specific. For example, the function `OPENFILE` takes the name of the file it is to open and fills in host and directory defaults, and decides which device handles such a file. It then calls on the particular device to actually open the file. After the file is opened, the generic file system code registers the file on (`OPENP`). As another example, operations involving open streams first coerce non-streams (e.g. filenames) to open streams before calling the device-specific operation.

Devices

A device is an object of type `FDEV` (so named for historical reasons: "File Device"). The standard way to define a new device is to create such an object, by performing (**create** `FDEV` --), and then pass the newly created `FDEV` to the function `\DEFINEDEVICE`. `\DEFINEDEVICE` is the way a device "announces" itself to the generic file system.

(`\DEFINEDEVICE` *NAME* *DEV*)

[Function]

Installs device *DEV*, giving it the name *NAME*. *NAME* must be an uppercase litatom. The generic file system code makes use of the name to locate the device that is willing to deal with files whose full name begins {*NAME*}. It is permissible to have more than one name map to the same device; this effectively provides device synonyms. Devices are encouraged, however, to always create file names using the canonical device name, independent of what name was passed in.

NAME can be `NIL`, in which case the generic file system code never consults the device directly. However, its `EVENTFN` is still run around Lisp exits, and it can be used as the device for a stream created by nonstandard methods.

If a device never wants to be invoked by name, and has no interesting `EVENTFN` or `HOSTNAMEP` methods, then there is no need to ever register it with `\DEFINEDEVICE`.

(\REMOVEDEVICE *DEV*) [Function]

Removes device *DEV* from the list of known devices, as well as any name that maps to that device.

(\GETDEVICEFROMNAME *NAME NOERROR DONTCREATE*) [Function]

Returns the device associated with *NAME*. *NAME* can be a litatom or string; it is coerced to uppercase, and if it begins with an open brace, is assumed to be a file name, from which the host name is extracted. If no such device is known, attempts to find one by polling the *HOSTNAMEP* methods of all known devices (see below); if a device is still not found, causes a *FILE NOT FOUND* error unless *NOERROR* is true. If *DONTCREATE* is true, it never attempts to create a device, just returns an existing device if there is one, *NIL* otherwise.

The fields of an *FDEV* are divided up into informational fields and "methods".

DEVICENAME	A pointer field, the name of the device, standardly a litatom. Use of this field is largely up to the device, but it is usually selected to be the name that appears inside braces in filenames opened on this device. For devices that do not support the notion of named files, <i>DEVICENAME</i> can be anything that the implementor cares to use for debugging assistance.
RESETABLE	A flag, true if (<i>SETFILEPTR</i> stream 0) can be performed. Currently unused.
RANDOMACCESSP	True if the stream is randomly accessable, i.e., if <i>SETFILEPTR</i> works on this kind of stream.
NODIRECTORIES	True if files opened on this device do not (usually) have a directory as part of their name. The principal use for this is by the <i>CONN</i> command, which will not try to connect to the user's home directory if given a host only, e.g., <i>CONN {DSK}</i> .
BUFFERED	True if streams of this sort are buffered in a manner compatible with the microcoded versions of <i>BIN</i> and <i>BOUT</i> . More specifically, <i>BUFFERED</i> implies that the device implements the <i>GETNEXTBUFFER</i> method. See description of buffered streams.
PAGEMAPPED	True if this stream is implemented by the pagemapped functions. All pagemapped streams are also buffered, so if this flag is true, so should be <i>BUFFERED</i> . See description of pagemapped streams.
FDBINABLE	True if streams on this device obey the rules for microcoded <i>BIN</i> whenever such stream is open for input access.
FDBOUTABLE	True if streams on this device obey the rules for microcoded <i>BOUT</i> whenever such stream is open for output access. Currently unused, as the spec needs revision.
FDEXTENDABLE	Special kind of <i>FDBOUTABLE</i> . Currently unused, as the spec needs revision.
DEVICEINFO	A pointer to arbitrary device-specific information. The standard use for this is to hold local state specific to one of several similar devices that share methods. For example, the Dolphin disk provides a separate <i>FDEV</i> for each partition of the machine; the <i>DEVICEINFO</i> field of each has pointers to the partition's directory and other information specific to files on that partition only.

The following fields are all pointer fields, and contain functions for implementing various device operations. Not all devices need have all fields filled in; the required ones are listed first and so indicated. Some "required" fields have defaults specified in the FDEV (or STREAM) record declaration, so the implementor need not explicitly fill those fields if the default is reasonable. Each field is presented with its arguments, in the style of a function definition; of course, it is the contents of the field, not the field name, that is the function. Using object-oriented terminology, the occupants of these fields are referred to as "methods". For example, "the BIN method" means "the function that occupies the BIN field".

One of the arguments to each method is usually either the device itself, or a stream open on the device, so that the device (and hence its DEVICEINFO) is usually accessible to all these functions. Arguments that are file names or patterns or pieces of file names can be either litatoms or strings, and already have their host and/or directory parts appropriately filled in from the connected directory defaults. The device may assume that the host field of the file name is indeed a name that the device has said it implements (see HOSTNAMEP). "Full" file names returned by these functions (or stored in the FULLFILENAME field of a stream) should be litatoms, and at least in the current implementation should be all uppercase.

Fields required of every device:

(HOSTNAMEP *HOSTNAME* *DEVICE*)

Called by the generic file system code when presented with a host name for which there is as yet no device defined. The function should return non-NIL if it "recognizes" *HOSTNAME*. There are two ways in which this method is invoked:

- (1) To obtain a device for *HOSTNAME*, for example, so that a file can be opened on it. In this case, *DEVICE* is an already defined device (the one whose HOSTNAMEP method is being called), and the function should return either a new device, or T, meaning it is willing to take responsibility for this host name as well as any previous name under which the device was registered. In either case, the caller will install the returned device, or *DEVICE* if value was T, as the device to which *HOSTNAME* maps.
- (2) As a pure predicate. In this case, *DEVICE* is NIL, and the function need only return T or NIL, indicating whether it believes that *HOSTNAME* is the name of a host.

In practice, the HOSTNAMEP method need only take care of the first case, since that also takes care of the second case. The second case is provided so that the device need not be created until there is an actual use for it, should the device wish to avoid unnecessary work. In practice it is rare that anyone tests a host name without subsequently needing to have the device created in full.

There are basically three kinds of devices in the system as distinguished by their HOSTNAMEP methods.

- (1) Predefined devices with exactly one name, or strictly internal devices with no notion of name. For example, the CORE device always exists, and has exactly one name; the SPP device (a network byte stream) has no name (it supports no files directly). Such devices have a HOSTNAMEP method of NIL—the only name they ever go by is the one they gave to \DEFINEDEVICE, if any. This is the default.
- (2) Devices that don't know ahead of time what their name will be, but for which there might be many incarnations. This is the model for remote file servers. The standard way of handling this case is to define a dummy device that has only a HOSTNAMEP method, and no name. When the HOSTNAMEP method gets called with a name that the device knows it can service, it creates a device by that name. If given a name that is a synonym of another name, it might just return the existing device of the canonical name (using \GETDEVICEFROMNAME to find the right device).

In either case, the `HOSTNAMEP` method of the new device is usually `NILL`—the original device is the only one that worries about creating new instances of this class of device.

(3) Like (2), but all the different names are handled by a single device, which takes care internally of the multiplexing among, say, different remote hosts. `HOSTNAMEP` returns `T` in this case. This is usually clumsier than (3), so discouraged.

(`EVENTFN` *DEVICE* *EVENT*)

Called around Lisp exits, to allow the device to do any necessary cleaning up, clearing of caches, disconnects with remote hosts, etc. *EVENT* is one of the following litatoms: `BEFORELOGOUT`, `BEFORESYSOUT`, `BEFOREMAKESYS`, `BEFORESAVEVM`, `AFTERLOGOUT`, `AFTERSYSOUT`, `AFTERMAKESYS`, `AFTERSAVEVM`, `AFTERDOSYSOUT`, `AFTERDOMAKESYS`, `AFTERDOSAVEVM`. The `AFTERxxx` events are all run when Lisp is booted from a memory image that resulted from a `LOGOUT`, `SYSOUT`, etc. The `AFTERDOxxx` events run when continuing Lisp in the same incarnation following the `SYSOUT`, etc. (there is no such event for `LOGOUT`, of course). The "after" events are called in the same order in which the devices were defined; the "before" events in the reverse order.

For example, the `BEFORELOGOUT` event for the Leaf remote file server devices performs a `FORCEOUTPUT` on all its open files and then breaks the connection with the file server. The `AFTERxxx` events for the Leaf devices calls `\REMOVEDEVICE` on itself to flush any connection between the name and the server (since names and addresses can change over exit). The `AFTERxxx` events for the Dorado disk device rebuilds its cache of the disk's directory.

There are a few devices in the system that exist only for their `EVENTFN`. In most cases, a simpler way to tell the system you want something performed around exit is to add your event function to the list `AROUNDEXITFNS` instead of going to the expense of defining a device for it. There is yet another list, `\SYSTEMCACHEVARS`, for handling a more specialized "around exit" operation: every time Lisp is booted, each of the variables in the list `\SYSTEMCACHEVARS` is set to `NIL`.

The following are required of all named devices, that is, devices that map from some hostname to the device, upon which files might be opened or otherwise manipulated:

(`DIRECTORYNAMEP` *HOST/DIR* *DEVICE*)

True if *HOST/DIR* is a valid directory name on *DEVICE*. Function should ideally perform recognition as well, and return the "true" name. For example, given "`{PHYLEX:<LISP>}`" as argument, it might return `{Phylex:PARC:Xerox}<Lisp>`. *HOST/DIR* might include a subdirectory name. The device should attempt to tell the truth about whether the subdirectory exists or not, though this may not be possible for devices with fake subdirectories. Defaults to `NILL`, i.e., device supports no directories. Used by the command `CONN` and the function `DIRECTORYNAMEP`.

(`OPENFILE` *NAME* *ACCESS* *RECOG* *PARAMETERS* *DEVICE*)

Used to implement the `OPENFILE` and `OPENSTREAM` functions. Opens the file named *NAME* on this device for access *ACCESS*, returning a *STREAM*. The stream is usually on *DEVICE* (its *DEVICE* field is *DEVICE*), but is not required to be. The arguments *ACCESS*, *RECOG*, *PARAMETERS* are as with the `OPENFILE` function in the manual. Thus, if *NAME* does not include a version number, recognition is according to *RECOG*, which should be appropriately defaulted per *ACCESS* (`INPUT` implies `OLD`, `OUTPUT` implies `NEW`, `BOTH` implies `OLD/NEW`).

The argument *NAME* can also be a *STREAM*, which must be a closed stream. `OPENFILE` should "reopen" the stream. The value returned in this case may be a new stream (with the same

name as the old), or the old stream (*NAME*) itself. It is likely that the specification will be changed at some point to require that the old stream itself be returned, suitably reopened.

The argument *PARAMETERS* is a list of pairs (*OPTION VALUE*). The most interesting *OPTIONS* are as follows:

TYPE	For new files, the type of the file (TEXT or BINARY). If this parameter is not specified, the value of the global variable DEFAULTFILETYPE (initially TEXT) should be used.
CREATIONDATE	For new files, the date of its creation. The device should use this if at all possible instead of letting the creation date default to the current date and time.
LENGTH	The intended length of the file, in bytes. This need not be accurate—it is only a hint that may allow smarter allocation. For example, if the device knows that it does not have room for a file of the specified length, it should immediately cause a FILE SYSTEM RESOURCES EXCEEDED error for the intended file.
DON'T.CHANGE.DATE	For old files being opened for access BOTH, don't change the creation date of the file. ACCESS = BOTH would normally imply that the content of the file is to change, and thus its creation date should be updated. Use of this parameter is a form of "cheating" to make it look as though the file had not changed. For example, the code that rewrites filemaps uses this parameter, since rewriting the filemap does not logically change the file's content.
SEQUENTIAL	If T, is a hint that the file will, or need, only be accessed sequentially, which may allow the device to open the file in a more efficient mode.

Any parameters that the device does not understand should be ignored, rather than be cause for an error. All devices are encouraged to support at least TYPE and CREATIONDATE.

The additional options ENDOFSTREAMOP and BUFFERS are handled by the generic file system code; specifying them is equivalent to calling SETFILEINFO (q.v.) immediately after the open.

Fine point about ACCESS = OUTPUT: this operation always produces a new, empty file, independent of whether its name is exactly the name of an existing file. That is, it replaces any old file by the same name. On opening, such a file has an end of file of zero. Of course, since RECOG defaults to NEW in this case, the name can only clash with an old file name if a version was explicitly specified, or RECOG is OLD or OLD/NEW. To open an old file for output but preserve its contents, i.e., only write over part of the file, one should open for ACCESS = BOTH (since to preserve the old contents one implicitly reads them).

Exception handling: If the desired file is not found, the OPENFILE method should return NIL rather than cause a FILE NOT FOUND error. This is so that the generic file system code can cause the error using the original file name, not the one packed with host and directory passed in to the OPENFILE method. The device should feel free to signal any other errors itself on failing to open the file, e.g., FILE WON'T OPEN for a busy file, PROTECTION ERROR, or FILE SYSTEM RESOURCES EXCEEDED. Ideally, this error should be signaled in a way that is resumable, i.e., so that a user could, in the break, take some action to remedy the condition and then type OK to continue. In most cases it suffices that all the internal functions below the OPENFILE be named with backslashes, so that the error code will choose to resume by reverting to the OPENFILE and trying again.

The device does not need to know about the set of open files (i.e., the value of `(OPENP)`), and in general should ignore it. That is, the device should perform the open as if there were no other files open and hence no conflict. The generic file system code looks at the stream returned from the `OPENFILE` method and then worries about whether there is actually another stream open by the same name. If there is, it closes the newly opened stream and then either returns the pre-existing stream, or causes a `FILE WON'T OPEN` error if the new and old access modes are in conflict. This design is crafty, but I believe it stems principally from the recognition problem—you don't know the full name of a file until you open it, so you can't tell until then whether you should have tried to open it in the first place. It will, of course, have to be completely changed when we go to multiple streams per file.

`(REOPENFILE NAME ACCESS RECOG PARAMETERS DEVICE OLDSTREAM)`

This is exactly like `OPENFILE`, except that it is called after `LOGOUT` (or other "after" events) on the name of any stream that was left open over exit. The idea is to maintain the illusion that the file really was open over `LOGOUT`, but check and make sure nothing changed. The generic file system code uses the `VALIDATION` field to test whether the file changed behind your back.

`OLDSTREAM` is the stream that was open before exit, and is supplied for the benefit of devices where there is no possibility that the file changed (e.g., `{CORE}`), so that they can just return `OLDSTREAM` directly. `OLDSTREAM` is also of use for those devices that have to cheat in order to maintain the illusion.

This will have to change when we go to multiple streams per file.

`(GETFILENAME NAME RECOG DEVICE)`

Performs "recognition" on `NAME`. That is, it returns the full name of the file that would be opened by `OPENFILE` in the indicated recognition mode, or `NIL` if the file is not found. It is not necessary that `OPENFILE` actually be capable of opening the file (there is no need to check protection, for example). Used by `INFILEP`, `OUTFILEP`, `FULLNAME`.

`(DELETEFILE NAME DEVICE)`

Deletes the file named `NAME`, returning its full name on success, `NIL` on failure. Recognition mode is implicitly `OLDEST`. Local devices, after recognizing the file, should make sure that it is not `OPENP` (open files can not be deleted). This and `RENAMEFILE` are usually the only device methods that need to know anything about what files are open.

`(GENERATEFILES DEVICE PATTERN DESIREDPROPS OPTIONS)`

Enumerates files matching `PATTERN`. Returns a "file generator object" of the form `(NEXTFILEFN INFOFN . ArbitraryState)`. This is described in more gory detail under **Directory Enumeration**.

`(RENAMEFILE OLDNAME NEWNAME DEVICE)`

Renames the file named `OLDNAME` to have name `NEWNAME`. Returns the full name of the new file if successful, `NIL` if not. Recognition mode is implicitly `OLD` for `OLDNAME`, `NEW` for `NEWNAME`. The generic file system code invokes this method to implement the function `RENAMEFILE` only when the host fields of both filenames map to the same device. Defaults to `\GENERIC.RENAMEFILE`, which is also the function that the system calls when the old and new names are on different devices. `\GENERIC.RENAMEFILE` is defined to copy `OLDNAME` to `NEWNAME` and then delete `OLDNAME`.

The following methods are invoked for open streams. They are all required:

`(BIN STREAM)`

Returns the next byte of input from *STREAM*, or takes the appropriate action if at end of file. Unless a device has a good reason not to, it should call (`\EOF.ACTION STREAM`) at end of file/stream.

The device BIN method is actually not used directly. Rather, every stream has a *STRMBINFN* field, which is the function actually applied to do the input. The *STRMBINFN* field could thus be used to fake a specialization of the device differing only in the BIN method. However, the typical use of *STRMBINFN* is to temporarily override the device default. In particular, setting a stream's access to INPUT or BOTH automatically sets the stream's *STRMBINFN* to be the device's BIN method; setting access to NIL or OUTPUT sets the *STRMBINFN* to be an error. This relieves the device's BIN method of any need to check the stream's access on every call to BIN. Some network streams temporarily set their *STRMBINFN* to be an input eater when they receive a "clear output" command.

Currently, all Interlisp-D streams have bytesize 8, so BIN always returns an 8-bit integer.

Calls to the function BIN are compiled into the BIN opcode, which runs in microcode on some machines if the requirements for it are met. More on this later.

(BOUT *STREAM* *BYTE*)

Outputs *BYTE* to *STREAM*. As with BIN, this method is not used directly. Rather, every stream has a *STRMBOUTFN* field, which is the function actually applied to do the output. Setting a stream's access to OUTPUT or BOTH automatically sets the stream's *STRMBOUTFN* to be the device's BOUT method.

There exists a BOUT opcode, but the design is incomplete.

(PEEKBIN *STREAM* *NOERRORFLG*)

Returns the next input byte from *STREAM*, but does not advance the stream pointer. Thus a subsequent PEEKBIN or BIN will return the same byte. At end of stream, the device should take eof action as with BIN, unless *NOERRORFLG* is true, in which case it should return NIL.

(READP *STREAM* *FLG*)

Returns true if input is available from *STREAM*, that is, if a BIN right now would succeed without waiting. Defaults to `\GENERIC.READP`, which uses EOFP and PEEKBIN.

Roughly speaking, READP is the complement of EOFP for streams that are not arriving in real time. It is interestingly different for network streams, or the keyboard.

FLG is a bit of cruft that not everyone pays attention to, and may be flushed at some point: if *FLG* is NIL, then READP should return NIL if the only input waiting is an end of line character.

(EOFP *STREAM*)

Returns true when *STREAM* is "at end of file", i.e., a BIN would cause an end of file action to occur. Note that for a network stream, it is possible for both EOFP and READP to be false simultaneously, viz. when there is no input waiting (buffered locally), but the remote end of the stream has not indicated that there is no more input.

There are some who call EOFP on streams open only for output. This is a crock; output streams are always at end of file. But to avoid complaints, a device could return T for EOFP on an output stream.

(BLOCKIN *STREAM* *BUFFER* *BYTEOFFSET* *NBYTES*)

Performs bulk input transfer: retrieves the next *BYTES* bytes from *STREAM* and stores them in successive byte positions in *BUFFER* starting at *BYTEOFFSET*. Defaults to `\GENERIC.BINS`, which repeatedly calls `BIN` and `\PUTBASEBYTE`.

It is almost always the case that a device with a non-trivial `BLOCKIN` method can be made to be a Buffered device, thereby benefiting from other Buffered operations as well.

`(BLOCKOUT STREAM BUFFER BYTEOFFSET BYTES)`

Performs bulk output transfer: outputs *BYTES* bytes to *STREAM*, taking the bytes from *BUFFER* starting at *BYTEOFFSET*. Defaults to `\GENERIC.BOUTS`, which repeatedly calls `\GETBASEBYTE` and `BOUT`.

`(FORCEOUTPUT STREAM WAITFORFINISH)`

Forces to its ultimate destination any output buffered on *STREAM* but not yet sent. *WAITFORFINISH* means that the function should not return until it is confident that the output has reached its destination and been committed. Defaults to `NILL`, which is reasonable for unbuffered streams.

For example, for a network stream, `FORCEOUTPUT` sends the current packet being buffered up. For a buffered stream to the disk, `FORCEOUTPUT` writes out to the disk any "dirty" pages, and makes sure the file is in such a state that if the machine were booted after `FORCEOUTPUT` returns, that the file could be successfully reopened with no information lost.

`(GETFILEINFO NAME/STREAM ATTRIBUTE DEVICE)`

Returns the value of the specified *ATTRIBUTE* of *NAME/STREAM*, which can be an open Stream or the name of a (closed) file. Returns `NIL` for attributes it doesn't know about. It is considered good citizenship, though not absolutely required, to know about the following attributes:

<code>LENGTH</code>	Length of the stream/file in bytes. If the device's method returns <code>NIL</code> , but the stream is random access, the generic <code>GETFILEINFO</code> code tries the device's <code>GETEOFPTR</code> method instead.
<code>SIZE</code>	Length in pages, i.e., <code>(FOLDHI length BYTESPERPAGE)</code> .
<code>CREATIONDATE</code>	Date when the file's contents were created, as a string. The creationdate does not change when a file is copied or renamed, only when it is changed.
<code>WRITEDATE</code>	Date when the file was written to its current place of storage.
<code>READDATE</code>	Date when the file was last read.
<code>ICREATIONDATE, IWRTEDATE, IREADDATE</code>	The creation, write and read dates as integers, such as from the function <code>IDATE</code> .
<code>TYPE</code>	Type of the contents: <code>TEXT</code> for files that contain only "text" (generally meaning 7-bit ascii), <code>BINARY</code> for all others. <code>NIL</code> means unknown.
<code>AUTHOR</code>	Name of the user who created the file (a string).

The following "generic" attributes are generally handled by the generic side of `GETFILEINFO` if the device's `GETFILEINFO` method returns `NIL`:

<code>EOL</code>	The end of line convention of the stream (<code>CR</code> , <code>CRLF</code> or <code>LF</code>).
<code>BUFFERS</code>	The number of pagemap buffers for use by the stream (see description of <code>MAXBUFFERS</code> field of pagemapped streams).

ENDOFSTREAMOP Action to take on any attempt to read beyond the end of file. This is a function of one argument, the stream. The function can cause an error, or return a value, which is interpreted as a value to return from BIN. The default ENDOFSTREAMOP causes an END OF FILE error.

ACCESS An atom describing the access mode of the stream (INPUT, OUTPUT, etc). This is so generic that it is handled before the device's method ever sees it.

BYTESIZE, OPENBYTESIZE The size of bytes transmitted on the stream. Always 8 these days.

(SETFILEINFO NAME/STREAM ATTRIBUTE VALUE DEVICE)

Sets the value of the specified *ATTRIBUTE* of *NAME/STREAM* to be *VALUE*. Returns T if successful, NIL if unsuccessful, or for attributes it doesn't know about.

It is not generally required that SETFILEINFO recognize any attributes at all—NIL is a perfectly good filler for this slot. Most devices recognize no more than TYPE and CREATIONDATE (ICREATIONDATE), and even those are not very important, as most applications set those attributes in the *PARAMETERS* argument to OPENFILE when creating a file.

ATTRIBUTE = LENGTH implies actually truncating (or lengthening) the file; however, the SETFILEINFO need not handle this itself—if it returns NIL, then the generic file system will attempt to use the SETEOFPTR method instead.

The following operations are only required of random access streams. They default to the function \IS.NOT.RANDACCESSP, which causes a "Stream is not randaccessp" error when called.

(GETFILEPTR STREAM)

Returns the current file pointer (byte position) in *STREAM*. The file pointer is zero when the stream is opened (except for *ACCESS* = APPEND), and is incremented by one for each byte read.

Although this operation is only absolutely required for random access streams, it is desirable to supply it for other streams where possible. For example, when reading a file sequentially through PupFtp, the stream can count the bytes as they go by and thus give an accurate value for GETFILEPTR. If a stream has no idea at all of position, it can make its GETFILEPTR be the function ZERO and thereby at least avoid breaks from code that calls GETFILEPTR carelessly.

(GETEOFPTR STREAM)

Returns the file pointer of the end of *STREAM*, i.e., the file pointer that GETFILEPTR would return after the last byte of *STREAM* is read. Same as the LENGTH attribute for a stream that represents a file. Of course, non-random access streams may have no idea where the end is, and causing a non-randaccessp error is perfectly acceptable.

(SETFILEPTR STREAM BYTENUMBER)

Sets the file pointer of *STREAM* to be *BYTENUMBER*. The special value *BYTENUMBER* = -1 means the end of the stream; other negative values are illegal.

SETFILEPTR beyond the end of the stream is permissible, but it has no immediate effect beyond changing the logical file pointer. Attempting to then BIN causes an EOF error. Attempting to BOUT (for a file open for write) should extend the file, so that its eof is immediately beyond the newly BOUTed byte.

As with `GETFILEPTR`, there is no requirement that this work on non-random access streams, and it may be completely impossible on some of them. However, for those non-random access streams that perform `GETFILEPTR`, it is possible to fake `SETFILEPTR` for values larger than the current file pointer by skipping some number of bytes in the file, e.g., by performing `(RPTQ (DIFFERENCE BYTENUMBER (GETFILEPTR STREAM)) (BIN STREAM))`. There are some applications for which forward `SETFILEPTR` is all the random access that is actually required, so it is nice to be able to accommodate such applications.

`(BACKFILEPTR STREAM)`

Backs up the file pointer in *STREAM* by one byte. Functionally the same as `(SETFILEPTR STREAM (SUB1 (GETFILEPTR STREAM)))`, but may be possible on non-random access streams by maintaining a one-character buffer, which is all the backing up this operation is formally required to perform. I believe the main use for this is in `READ`, which needs to back up the stream one character when, for example, it reads a break character terminating an atom.

`(SETEOFPTR STREAM LENGTH)`

Changes the length of *STREAM* to be *LENGTH*, i.e., "sets" its end of file pointer. This may require lengthening or truncating the file. Used by the function `\SETEOFPTR` and by `SETFILEINFO` for attribute `LENGTH` when the device's `SETFILEINFO` method doesn't handle it.

The following three fields are place holders for possible future extensions. These fields are not currently used at all:

`(LASTC STREAM)`

Returns the last character read from *STREAM*, i.e., the last byte that was `BIN`ed, as a character. `LASTC` is currently implemented via `BACKFILEPTR`.

`(FREEPAGECOUNT HOST/DIR DEVICE)`

Intended use is to return the number of free pages on *HOST/DIR*. May be folded into a general `GET/SET` device/directory info operation.

`(MAKEDIRECTORY HOST/DIR DEVICE)`

Intended use is to create a new directory *HOST/DIR*.

The remaining fields in the `FDEV` are for buffered and page-mapped streams, and are ignored for non-buffered devices. These fields are described in separate sections.

Streams

The following fields are used by all streams:

<code>DEVICE</code>	Pointer to this stream's <code>FDEV</code> .
<code>FULLFILENAME</code>	"Full" name by which this file is known to the user. Should be an uppercase litatom, fully qualified so that giving the same name back to the file system should produce the same file (to the extent that the device can support such uniqueness). Is <code>NIL</code> for unnamed streams.

FULLNAME	Access field. Is the same as FULLFILENAME, unless that is NIL, in which case it is the stream itself. This avoids the circularity that would result if the FULLFILENAME field contained the stream datum.
NAMEDP	Access field. Is T if the streams is named, i.e., its FULLFILENAME is non-NIL.
ACCESSBITS	Contains a numeric code describing what access mode the file is open for: there are read, write and append bits. This field is usually accessed indirectly via the ACCESS field. However, there are macros for referring to particular types of access using more efficient bit test operations: (OPENED STREAM) ACCESS is not NIL. (READABLE STREAM) Read bit is on: ACCESS is INPUT or BOTH. (READONLY STREAM) Only the read bit is on: ACCESS is INPUT. (APPENDABLE STREAM) Append bit is on: ACCESS is OUTPUT, BOTH or APPEND. (APPENDONLY STREAM) Only the append bit is on: ACCESS is APPEND. (DIRTYABLE STREAM) Append or write bit is on: ACCESS is OUTPUT, BOTH or APPEND. Yes, this is operationally the same as APPENDABLE, given the four possible values of ACCESS. (OVERWRITEABLE STREAM) Write bit is on: ACCESS is OUTPUT or BOTH. (WRITEABLE STREAM) Write bit is on, or append bit is on and file is at EOF. Avoid using this one, it's a little strange.
ACCESS	Access field for referring to the ACCESSBITS field symbolically. Its value is one of the legal values of the ACCESS argument to OPENFILE: INPUT, OUTPUT, BOTH, APPEND; or NIL when the stream is closed. Replacing this field has the side effect of setting the BINABLE, BOUTABLE, STRMBINFN and STRMBOUTFN fields appropriately (from the corresponding device fields, or to values consistent with no access).
USERCLOSEABLE	Flag, true if the stream can be closed by CLOSEF. Default is T, but is NIL for such things as dribble files and the terminal.
USERVISIBLE	Flag, true if the stream is to be listed in the result of (OPENP). Default is T, but is NIL for such things as dribble files and the terminal.
BINABLE	True if BIN microcode can be used. Normally set automatically from FDBINABLE when input access is set.
BOUTABLE	True if BOUT microcode can be used. Normally set automatically from FDBOUTABLE when output access is set.
EXTENDABLE	True if BOUT can extend the buffer when COFFSET reaches CBUFSIZE. Obsolete.
STRMBINFN	Function called by BIN. This is normally set indirectly as a side effect of setting the ACCESS field. Setting ACCESS to an input access (INPUT or BOTH) sets the STRMBINFN to be the stream's device's BIN method. Setting to any other access sets the STRMBINFN to be a "file not open" trap.

STRMBOUTFN	Function called by BOUT. As with STRMBINFN, this is normally set indirectly (from the device's BOUT method) as a side effect of setting the ACCESS field.
OUTCHARFN	Function called to output a single byte. This is like STRMBOUTFN, except for being one level higher: it is intended for text output. Hence, this function should convert (CHARCODE EOL) into the stream's actual end of line sequence, and should adjust CHARPOSITION appropriately before invoking the stream's STRMBOUTFN to actually put the character. Defaults to \FILEOUTCHARFN. The OUTCHARFN for the display additionally worries about such things as ECHOCONTROL. CHARPOSITION Current horizontal character position in the stream. Incremented (and reset to zero) by OUTCHARFN. Used by the function POSITION. LINELENGTH Maximum line length of the stream, in characters. Used by the function LINELENGTH. Defaults (at creation time) to the value of the global variable FILELINELENGTH. EOLCONVENTION The stream's end of line convention: the manner in which "end of line" is encoded on this stream. That is, output of an end of line (function TERPRI) produces the stream's end of line sequence, and on input, the stream's end of line sequence is converted to (CHARCODE EOL) by READC. This is not necessarily the same as the way that end of line is encoded in the actual file written by, say, a file server. For example, Lisp might open a stream to a Tenex file server with EOLCONVENTION of CR, while the server might choose to take each of the CRs in the stream and actually store a CR, LF sequence in the physical file. The convention is encoded as a two-bit field; the constants CR.EOLC, LF.EOLC, CRLF.EOLC can be used to refer to the currently known values symbolically. Default in Interlisp-D is CR.EOLC. ENDOFSTREAMOP Function of one argument (the stream) called when an attempt to read beyond the end of file occurs. If this function returns something, it should be interpreted as a value to return from BIN (the value T is currently prohibited). Defaults to \EOSERROR, which causes an END OF FILE error. VALIDATION Pointer field, some compact encoding of the state of the file such that if the file's content changes, the VALIDATION changes. The file's ICREATIEONDATE attribute usually works well enough. The only use for this field is to check whether the file changed over LOGOUT, etc.—if the VALIDATION of the stream returned from REOPENFILE is EQUAL to the VALIDATION of the stream open before LOGOUT, the stream is assumed to be unchnaged. This will probably be the sole concern of the device when we go to multiple streams per file.
BYTESIZE	Byte size of the file, i.e., what BIN and BOUT traffic in. Defaults to 8. This field is not used by many; there are probably a lot of things that won't work if the byte size is not 8.
OTHERPROPS	List in property list format used by the function STREAMPROP. Analogous to WINDOWPROP, etc.
IMAGEOPS	Image operations vector (object of type IMAGEOPS) for use of device-independent graphics operations, such as DSPXPOSITION, DSPFONT. Defaults to \NOIMAGEOPS, a vector suitably defined for non-display devices. See the implementors' manual chapter Device-Independent Graphics .
IMAGEDATA	Device-dependent data for use by IMAGEOPS.

REVALIDATEFLG Flag. The standard use of this flag is to solve a problem with correctly maintaining the creation date. The problem is that the definition of "creation date" is that the creation date changes whenever the contents of the file change. If followed literally, this would mean, for example, that every time you wrote out a page of a {DSK} file, you would also have to rewrite its leader page with a new creation date. However, it suffices in practice to only change the creation date when it would matter, i.e., when there would be any possibility of some agent other than the currently running Lisp to see the change. Usually, this means the only time to worry about is when the Lisp vmem is saved and a file that was open before the save is written to again afterwards.

Thus, the use of this flag (for those devices that care) is as follows: the device's BEFORExxx events set this flag true for any streams open on the device. Then, whenever the device is about to do something that would change the file's content, e.g., write out a new page, it first tests REVALIDATEFLG. If the flag is true, it updates the file's creation date and clears the flag.

NONDEFAULTDATEFLG Flag. Standard use is in conjunction with REVALIDATEFLG, to mark a file that was opened in a way that the user constrained the creation date of the file (e.g., the *PARAMETERS* argument to *OPENFILE* included an explicit creation date, or the option *DON'T.CHANGE.DATE*).

F1, F2, F3, F4, F5 Pointer fields for private use by the stream, to maintain stream-specific state of concern only to the device. Stream clients that wish to hang information on a stream without regard to what kind of stream it is should use the function *STREAMPROP*.

FW6, FW7, FW8, FW9 16-bit word fields for private use by the stream.

DIRTYBITS Obsolete.

EXTRASTREAMOP ?

Buffered Streams

Buffered streams are ones that constrain themselves to obey a set of conventions that make it easy for an agent (e.g., microcode) to perform input or output on the stream without knowing about the details of the stream's physical i/o. The stream maintains a "current buffer" and two indices into that buffer, the offset of the next byte, and the offset of the end of the buffer. As long as the former index is less than the latter, the stream guarantees that the bytes in the buffer between those indices are the true contents of the file/stream starting at the current file pointer. Advancing the first index effectively advances the file pointer. When it reaches the second index, a stream-specific operation is called to "refill" the buffer.

The following fields are used by buffered streams:

COFFSET Byte offset in the buffer *CBUFPTR* of the next *BIN* or *BOUT*.

CBUFSIZE "Size" of the current buffer, i.e., byte offset that is one beyond the last byte.

CBUFMAXSIZE For output, the maximum size the buffer can be written to. If *COFFSET* reaches *CBUFSIZE*, but *CBUFSIZE* is less than *CBUFMAXSIZE*, then the buffer can be extended.

CBUFPTR	Pointer to current buffer. Must be valid if COFFSET is less than CBUFSIZE and BINABLE or BOUTABLE is true. It is not necessary that this "buffer" be anything other than some chunk of memory, a portion of which contains interesting data. Thus, the bytes from offset COFFSET to CBUFSIZE must be valid, but COFFSET need not start at zero, nor need CBUFSIZE or CBUFMAXSIZE coincide with the end of the underlying structure.
CBUFDIRTY	Flag, true if current buffer has been written to.

In general, the device has sole responsibility for setting CBUFSIZE, CBUFMAXSIZE, and CBUFPTR; generic code does not touch those. The fields COFFSET and CBUFDIRTY can be changed by generic stream clients as well as by device-specific code. For example, code that simulates a BIN increments COFFSET; code that writes directly to the stream's buffer sets CBUFDIRTY true.

The following methods are defined for devices implementing buffered streams:

(GETNEXTBUFFER *STREAM* *WHATFOR* *NOERRORFLG*) [Device method]

Called when *STREAM* needs to have its buffer fixed, i.e., the state of *STREAM* is such that BIN (*WHATFOR* = READ) or BOUT (*WHATFOR* = WRITE) cannot proceed. This method should do whatever is necessary to allow the operation to proceed. This typically includes disposing of the current buffer somehow (if GETNEXTBUFFER was invoked because the buffer was exhausted), and fetching a new buffer consistent with *STREAM*'s current position.

In the case of *WHATFOR* = READ, GETNEXTBUFFER returns T on success, i.e., if *STREAM* is not at end of file. When *STREAM* is at end of file, GETNEXTBUFFER should take standard end of stream action, returning whatever \EOF.ACTION returns (if anything). However, if *NOERRORFLG* is true, GETNEXTBUFFER should just return NIL immediately.

(RELEASEBUFFER *STREAM* *BUFFER*) [Device method]

Performs any device-specific operation required when *BUFFER*, which is the current value of *STREAM*'s CBUFPTR field, is "released" (when the CBUFPTR field is replaced). This is used so that different pagemap-like devices can share certain code. For example, in the case of pagemapped streams, RELEASEBUFFER marks the buffer dirty in the case that the stream's CBUFDIRTY field has been set.

This method is not currently used.

The functions \BUFFERED.BIN, \BUFFERED.PEEKBIN, \BUFFERED.BOUT, \BUFFERED.BINS and \BUFFERED.BOUTS are supplied for use by buffered streams; they are standardly used to implement the BIN, PEEKBIN, BOUT, BLOCKIN and BLOCKOUT device methods. In addition, the function COPYBYTES, when presented with a source stream that is buffered, utilizes the GETNEXTBUFFER method to efficiently copy bytes to the destination a buffer-full at a time.

Pagemapped Streams

Pagemapped streams are a particular kind of random access Buffered stream that buffers its data in units of pages. The device provides methods that read or write data in units of pages, while system-supplied Pagemapped functions handle the responsibilities of a Buffered stream, as well as managing the file pointer for random access. In general, a stream can have several pages of a file buffered at a time, allowing the code to make some effort to make efficient use of multi-paged transfers where applicable.

To create a pagemapped device, create an FDEV, fill in the necessary private fields, then call the following function:

(\MAKE.PMAP.DEVICE *DEVICE*)

[Function]

Fills in fields in the device appropriate for pagemapped devices, and returns the updated device. The fields it fills are the flag fields FDBINABLE, FDBOUTABLE, RESETABLE, RANDOMACCESSP, PAGEMAPPED, BUFFERED (all true), and the methods BIN, BOUT, PEEKBIN, BLOCKIN, BLOCKOUT, READP, EOF, GETFILEPTR, BACKFILEPTR, SETFILEPTR, GETEOF, SETEOF, GETNEXTBUFFER and FORCEOUTPUT.

A Pagemapped device is required to supply the following methods (in addition to those required of all devices and not filled in by \MAKE.PMAP.DEVICE):

(READPAGES *STREAM FIRSTPAGE# BUFFERS*)

[Device method]

Causes pages of *STREAM* to be read into *BUFFERS*. The first page read is *FIRSTPAGE#* (zero for the first page of the file). *BUFFERS* is either a single page-sized buffer (a VMEMPAGEP), in which case exactly one page is read, or it is a list of such buffers. READPAGES returns the total number of bytes read. If the last page read is not a full page, READPAGES should zero out the rest of its buffer. READPAGES can assume that the buffers are page-aligned, although they need not be consecutive.

(WRITEPAGES *STREAM FIRSTPAGE# BUFFERS*)

[Device method]

Writes data from *BUFFERS* out to *STREAM*. The first page written is *FIRSTPAGE#*. *BUFFERS* is as with READPAGES.

Neither READPAGES nor WRITEPAGES affects *STREAM*'s file pointer or end of file; those are managed by higher-level pagemapped routines. WRITEPAGES might, however, want to look at *STREAM*'s EPAGE and EOFFSET fields if it needs to take any special action around the end of the file. It is possible, for no particularly good reason, for READPAGES to get called for a page beyond the end of file; in fact, this standardly happens when writing a new file. The READPAGES method in this case should just clear the buffer and return zero.

(TRUNCATEFILE *STREAM PAGE# OFFSET*)

[Device method]

Truncates *STREAM* so that its end of file is *PAGE#*, *OFFSET*, which should be defaulted to *STREAM*'s EPAGE and EOFFSET. Can be used to either shorten or lengthen a file; if lengthening, the file should be padded with nulls. Used by \PAGED.SETEOF and \PAGED.FORCEOUTPUT. As of this writing there are still bugs in this code in certain funny cases, such as when you SETFILEPTR beyond eof and then BOUT.

The following fields of a stream are meaningful for a pagemapped device. The generic pagemapped codes maintain them as operations on the file are performed, but they should all be initialized appropriately by the device's OPENFILE method:

CPAGE	For pagemapped streams, the current page position in the stream. Together with COFFSET, this constitutes the stream's file pointer. The device's OPENFILE method should set CPAGE and COFFSET to zero, except for files opened with access APPEND, in which case they should be set to the end of file.
EPAGE, EOFFSET	For pagemapped files, the page and byte offset of the end of file. Note that this is the <i>logical</i> end of the file; it need have nothing to do with the physical end of file, except that when a file is closed, the device should see to it that its logical and physical EOFs are the same (normally seen to by the TRUNCATEFILE inside of \CLEARMAP, below). In fact, as a

typical file is being written, EPAGE tends to stay several pages ahead of the physical end of file by virtue of the fact that pages are being buffered before being written out.

BUFFS	For pagemapped streams, a pointer to the stream's BUFFER chain. Initially NIL (no buffers allocated). The device usually has no direct interest in this field.
MAXBUFFERS	For pagemapped streams, the maximum number of buffers desired in the stream's BUFFS chain. If the code needs another buffer and there are already MAXBUFFERS buffers, it will try to recycle the least recently referenced buffer. Defaults to <code>\STREAM.DEFAULT.MAXBUFFERS</code> . The user can change this field for an open stream by calling <code>SETFILEINFO</code> with attribute <code>BUFFERS</code> .
MULTIBUFFERHINT	Flag. For pagemapped streams, is a hint to the pagemap code that the device prefers to transfer data more than one buffer at a time. If this flag is true, the pagemap code tries to write out (<code>WRITEPAGES</code>) more than one buffer at a time when the opportunity arises. A similar improvement is planned, but not implemented, for reading multiple buffers at a time.

The following functions are of use for pagemapped devices:

`(\PAGED.FORCEOUTPUT STREAM WAITFORFINISH)` [Function]

This function implements the `FORCEOUTPUT` method for pagemapped streams: it causes any dirty pages to be written out (using `WRITEPAGES`), then calls the `TRUNCATEFILE` method to set the end of file.

This function is normally installed as the `FORCEOUTPUT` method by the function `\MAKE.PMAP.DEVICE`. However, the device can override this default (by supplying its own function in that field), in which case it might want to call the function `\PAGED.FORCEOUTPUT` explicitly as part of its more comprehensive `FORCEOUTPUT` method.

There is an unpleasantness in the implementation of pagemapped devices that stems from the fact that originally all devices (the few that existed in the distant past) were made to support the `PMAP` package, a means whereby a programmer could get direct access to the buffers of a file, much as one can with the `PMAP JSYS` in Tenex. As a result, the buffers used by pagemapped streams are set up in a special manner so that the garbage collector can tell when the user no longer has access to a `PMAP` buffer. The `PMAP` package is being phased out.

This is all exceedingly crufty, and is of little concern to the device implementer, except for the fact that it requires that the buffers be explicitly released when a stream is closed; the buffers are not automatically collected when the stream is dropped.

`(FORGETPAGES STREAM FROMPAGE TOPAGE)` [Function]

"Forgets" pages `FROMPAGE` thru `TOPAGE` of `STREAM`; i.e., removes those pages from the set of pages being currently buffered, and frees the buffers they were occupying. If `FROMPAGE = TOPAGE = NIL`, forgets all pages, and releases all of `STREAM`'s buffers.

`(\CLEARMAP STREAM)` [Function]

Performs a `FORCEOUTPUT` (if `STREAM` is open for output) followed by a `FORGETPAGES`. This is the standard action that should be taken by a pagemapped stream's `CLOSEFILE` method.

Directory Enumeration

This section describes how directory enumeration works—what you need to know in order to implement the `GENERATEFILES` device method, and what you need to know as a programmer trying to enumerate a directory via anything more elaborate than the function `DIRECTORY`.

The general idea is that the directory enumeration code is given a pattern, and it returns a generator that, each time it is poked, returns another file name matching the pattern. In addition, the generator provides a handle for getting file attributes of each enumerated file. This second handle is important for efficiency: although one could just take the file name given by the enumerator and pass it to `GETFILEINFO`, the device, in the course of enumeration, usually has its fingers on the file closely enough that it need not perform the second directory lookup that a `GETFILEINFO` out of the blue would require. The caller of the directory enumeration code specifies ahead of time which, if any, attributes will be required (a necessity for most file server implementations).

Information for device implementors. A *file generator* is an object represented as a list described by the record `FILEGENOBJ`, exported from `FILEIO`:

```
(RECORD FILEGENOBJ (NEXTFILEFN FILEINFOFN . GENFILESTATE) )
```

`NEXTFILEFN` and `FILEINFOFN` are functions of the device's choosing that when called will return the next file, and attributes for that file. `GENFILESTATE` is arbitrary state maintained by the generator. With that as background, here are the pieces of directory enumeration:

```
(GENERATEFILES DEVICE PATTERN DESIREDPROPS OPTIONS) [Device method]
```

Returns a generator that enumerates files matching *PATTERN*, which is a string that has host and directories suitably filled in from defaults, and may contain the pattern character "*" to match an arbitrary number of characters. *DESIREDPROPS* is a list of file attributes that may be requested during the enumeration; they must be valid *ATTRIBUTE* arguments to `GETFILEINFO`. *OPTIONS* is a list of options to the enumeration, chosen from among the following:

SORT	The files should be enumerated in sorted order. If this option is not specified, the device is free to enumerate files in any convenient order.
	There is some question as to whether files should be enumerated lowest version first (as IFS's do) or highest version first (as Twenex does). I prefer the latter, but given servers that do the former, we currently make no requirement about version order.
RESETLST	Informs the enumerator that the enumeration context is surrounded by a <code>RESETLST</code> , so that it may perform <code>RESETSAVES</code> to clean up after itself if the enumeration is aborted. Cleaning up can be a very messy business without this information about the scope of the enumeration, so all callers of <code>\GENERATEFILES</code> are strongly encouraged to provide it.

`GENERATEFILES` should return a file generator with a suitable `NEXTFILEFN` and `FILEINFOFN`.

Fine point about missing fields in the pattern: null fields in *PATTERN* match only files for which the corresponding field is null. A null version is interpreted as highest. Thus,

`DIR * = DIR *. * = DIR *. * ; *` enumerates everything.

`DIR *. = DIR *. ;` enumerates all versions of files with null extension.

`DIR *. ;` enumerates highest version of files with null extension.

`DIR *.*;` enumerates highest version of everything.

It is difficult for some devices to enumerate only highest version of files; there are several devices in the system that treat a null version the same as version *. However, every device should try its best. With some work, any device that can enumerate all versions can enumerate just highest version if it enumerates in sorted order and uses perhaps a little lookahead to assure that any name it returns is the one of highest version.

`(NEXTFILEFN GENFILESTATE NAMEONLY)`

[File Generator Component]

Generates the next file, returning its name as a string, or `NIL` if the generator is exhausted. `GENFILESTATE` is the state component of the file generator returned from `GENERATEFILES`. `NAMEONLY` means that the caller is only interested in the file's `Name.Ext` fields, not the full file name (and no more than one version of the file need be enumerated); however, it is always permissible to return the full file name. The `NAMEONLY` option is used by `SPELLFILE`.

`(FILEINFOFN GENFILESTATE ATTRIBUTE)`

[File Generator Component]

Returns the value of the `ATTRIBUTE` property of the file most recently generated by the `NEXTFILEFN`, i.e., effectively `(GETFILEINFO latest-name ATTRIBUTE)`, but hopefully much faster. `ATTRIBUTE` must have been a member of the `DESIREDPROPS` argument to `GENERATEFILES`.

Not all device implementors are enthused about implementing a pattern matcher for file names. The following functions are provided to help out:

`(DIRECTORY.MATCH.SETUP PATTERN)`

[Function]

Accepts as `PATTERN` a file name string such as passed to `GENERATEFILES`. Returns an object suitable as a filter to `DIRECTORY.MATCH`.

`(DIRECTORY.MATCH FILTER TESTNAME)`

[Function]

Matches `TESTNAME`, a file name, against `FILTER`, the object returned from `DIRECTORY.MATCH.SETUP`. Returns true if `TESTNAME` matches the pattern, false if not. The match is case-insensitive.

`(\NULLFILEGENERATOR)`

[Function]

Returns a file generator that produces no files.

`(\GENERATENOFILES DEVICE PATTERN DESIREDPROPS OPTIONS)`

[Function]

Returns a "stupid" file generator for devices that don't know how to enumerate in general. If `PATTERN` contains no wildcards, but names a file that is `INFILEP`, then the generator produces exactly that file. If `PATTERN` contains a wildcard in the version field, it uses `GETFILENAME` to laboriously generate all the versions of the file. In all other cases, `\GENERATENOFILES` returns a null file generator.

Information for clients of device enumeration. The following functions make up the "public" interface to directory enumeration:

`(\GENERATEFILES PATTERN DESIREDPROPS OPTIONS)`

[Function]

Returns a file generator object for enumerating the files matching *PATTERN*. *PATTERN* is expanded by adding the default host and/or directory if appropriate. See description of the `GENERATEFILES` method for description of *DESIREDPROPS* and *OPTIONS*.

(`\GENERATENEXTFILE` *GENERATOR* *NAMEONLY*) [Function]

Returns the next file, as a string. *GENERATOR* is the object returned from `\GENERATEFILES`; *NAMEONLY* indicates caller does not require that the full name be returned, but that the name and extension are sufficient.

(`\GENERATEFILEINFO` *GENERATOR* *ATTRIBUTE*) [Function]

Returns the value of the *ATTRIBUTE* property of the file most recently generated by `\GENERATENEXTFILE`, i.e., effectively (`GETFILEINFO` latest-name *ATTRIBUTE*). *ATTRIBUTE* must have been a member of the *DESIREDPROPS* argument to `\GENERATEFILES`.

(`DIRECTORY.FILL.PATTERN` *PATTERN* *DEFAULTTEXT* *DEFAULTVERS*) [Function]

This function is used to fill in defaults in *PATTERN* before passing it to `\GENERATEFILES`. If *PATTERN* does not include an extension or version, but those fields are not explicitly omitted (e.g., "FOO", but not "FOO."; "FOO.BAR", but not "FOO.BAR;"), they are filled in with *DEFAULTTEXT* and *DEFAULTVERS*, which themselves default to "*". This function is used by the `DIR` command, and should probably be used by any code that takes a user-supplied pattern and enumerates files from it.

"TEDIT BEHIND EXEC WINDOW"
PRELIMINARY DESIGN NOTES
JIM BLUM
6/3/85

This paper is a list of issues and alternatives to designing and implementing TEDIT behind EXEC windows.

KNOWN ISSUES TO BE SOLVED

1. Certain functions which normally expect a display stream as the argument have to be changed to accept a textstream as a valid type of stream (or whatever kind of stream it ends up being, for now I will refer to it as a TEDITSTREAM). Some example functions are TTYDISPLAYSTREAM, (or whatever mechanism we use to replace WFROMDS functionality).
2. There has to be a place holder or mechanism by which it is know to TEDIT where the already processed text ends, and the current edible text starts and where/when it becomes processed, so that unprocessed may be edited and already processed input may not be.
3. The input may be different or at least have different "looks" (echoing) than the output. Examples, raising lower case to upper case; confirmation by carriage return may be replaced by "Yes" in the output stream, etc. How do we handle this?
4. Other functionality of TTYIN which at the time of this writing I am not yet familiar yet.
5. To what level do we support display stream graphic operations, such as MOVETO, DRAWLINE, clippingregion, XY coordinates, etc. Do we just paint graphics and don't capture them in any way, do we create imageobjects, or what?
6. Should we support two kinds of EXEC, one which uses the display stream to insure that all old programs will work as before, and a new type of EXEC which uses TEDITSTREAMS.
7. Do we implement this in one window in which it operates as one TEDITSTREAM, or do use a readonly TEDITSTREAM for backing, and use a separate window below the TEDIT window (with no border) to contain the current text which would use TTYIN to do the editing.

One advantage to implementing it as one TEDITSTREAM is that we could provide one standard method of backing display streams and handling text (ie, change the standard TEDIT interface to work this way) and possibly eliminate duplication of code such as TTYIN, thus making the system easier to maintain. On the other hand, it may turn out, that trying to handle this case as a general case in TEDIT is too envolved or too slow, and keeping them in two separate windows makes more sense.

Venue *MEDLEY LANGUAGE REFERENCE*

Address comments to:
Venue
User Documentation
1549 Industrial Road
San Carlos, CA 94070
415-508-9672

MEDLEY REFERENCE MANUAL

VOLUME I: LANGUAGE

April, 1993

Copyright © 1985, 1991, 1993 by Venue.

All rights reserved.

Medley is a trademark of Venue.

InterPress is a trademark of Xerox Corporation.

PostScript is a registered trademark of Adobe Systems Inc.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

The information in this document is subject to change without notice and should not be construed as a commitment by Venue. While every effort has been made to ensure the accuracy of this document, Venue assumes no responsibility for any errors that may appear.

Text was written and produced with Venue text formatting tools; PostScript printers were used to produce masters. The typeface is Palatino.

1. INTRODUCTION

Medley is a *programming system* that consists of a programming *language*, a large number of predefined programs (or *functions*) that you can use directly or as subroutines, and an *environment* that supports you with a variety of specialized programming tools. The language and predefined functions of Lisp are rich, but similar to those of other modern programming languages. The Medley programming environment, on the other hand, is very distinctive. Its main feature is an integrated set of programming tools that know enough about Interlisp and Common Lisp to act as semi-autonomous, intelligent "assistants" to you. This environment provides a completely self-contained world for creating, debugging and maintaining Lisp programs.

This manual describes all three parts of Medley. There are discussions of the language, about the pieces of the system that can be incorporated into your programs, and about the environment. The line between your code and the environment is thin and changing. Most users extend the environment with some special features of their own. Because Medley is so easily extended, the system has grown over time to incorporate many different ideas about effective and useful ways to program. This gradual accumulation over many years has resulted in a rich and diverse system. It is also the reason this manual is so large.

The rest of this manual describes the individual pieces of Medley; this chapter describes system as a whole—including the otherwise-unstated philosophies that tie it all together. It will give you a global view of Medley.

Lisp as a Programming Language

This manual is not an introduction to programming in Lisp. This section highlights a few key points about Lisp that will make the rest of the manual clear.

In Lisp, large programs (or functions) are built up by composing the results of smaller ones. Although Medley, like most modern Lisps, lets you program in almost any style you can imagine, the natural style of Lisp is functional and recursive—each function computes its result by calling lower-level “building-block” functions, then passing that result back to its caller (rather than by producing “side-effects” on external data structures, for example).

Lisp is also a list-manipulation language. Like other languages, Lisp can process characters and numbers. But you get more power if you program at a higher level. The primitive data objects of Lisp are “atoms” (symbols or identifiers) and “lists” (sequences of atoms or lists), which you use to represent information and relationships. Each Lisp dialect has a set of operations that act on atoms and lists, and these operations comprise the core of the language.

Invisible in the programs, but essential to the Lisp style of programming, is an automatic memory management system (an “allocator” and a “garbage collector”). New storage is allocated automatically whenever you create a new data object. And that storage is automatically reclaimed for reuse when no other object refers to it. Automated memory management is essential for rapid,

INTERLISP-D REFERENCE MANUAL

large-scale program development because it frees you from the task of maintaining the details of memory administration, which change constantly during rapid program evolution.

A key property of Lisp is that Lisp function definitions are just pieces of Lisp list data. Each subfunction "call" (or *function application*) is written as a list with the function first, followed by its arguments. Thus, (PLUS 1 2) represents the expression 1+2. A function's definition, then, is just a list of such function applications, to be evaluated in order. This representation of program as data lets you use the same operations on programs that you use on data—making it very easy to write Lisp programs that look at and change *other Lisp programs*. This, in turn, makes it easy to develop programming tools and translators, which was essential to the development of the Medley environment.

The most important benefit of this is that you can extend the Lisp programming language itself. Do you miss some favorite programming idiom? Just define a function that translates the desired expression into simpler Lisp. Now your idiom is *part of the language*. Medley has extensive facilities for making this type of language extension. Using this ability to extend itself, Interlisp has incorporated many of the constructs that have been developed in other modern programming languages (e.g. if-then-else, do loops, etc.).

Medley as an Interactive Environment

Medley programs should not be thought of as simple files of source code. All Medley programming takes place within the Medley environment, which is a completely self-sufficient environment for developing and using Medley programs. Beyond the obvious programming facilities (e.g., program editors, compilers, debuggers, etc.), the environment also contains a variety of tools that "keep track" of what happens. For example, the Medley File Manager notices when programs or data have been changed, so the system will know what needs to be saved at the end of a session. The "residential" style, where you stay inside the environment throughout the development, is essential for these tools to operate. Furthermore, this same environment is available to support the final production version, some parts providing run time support and other parts being ignored until the need arises for further debugging or development.

For terminal interaction, Medley provides a top level "Read-Eval-Print" executive, which reads whatever you type in, evaluates it, and prints the result. (This interaction is also recorded, so you can ask to do an action again, or even to undo the effects of a previous action.) Although Executives understand some specialized commands, most of the interaction will consist of simple Lisp expressions. So rather than special commands for operations like manipulating your files, you just type the same expressions that you would use to accomplish them in a Lisp program. This creates a very rich, simple, and uniform set of interactive commands, since any Lisp expression can be typed at an executive and evaluated immediately.

In normal use, you write a program (or rather, "define a function") by typing in an expression that invokes the "function defining" function (DEFINEQ), giving it the name of the function being defined and its new definition. The newly-defined function can be executed immediately, simply by using it in a Lisp expression.

INTRODUCTION

In addition to these basic programming tools, Medley also provides a wide variety of programming support mechanisms:

- List structure editor Since Lisp programs are represented as list structure, Medley provides an editor which allows one to change the list structure of a function's definition directly. See Chapter 16.
- Pretty-printer The pretty printer is a function that prints Lisp function definitions so that their syntactic structure is displayed by the indentation and fonts used. See page Chapter 26.
- Debugger When errors occur, the debugger is called, allowing you to examine and modify the context at the point of the error. Often, this lets you continue execution without starting from the beginning. Within a break, the full power of Interlisp is available to you. Thus, the broken function can be edited, data structures can be inspected and changed, other computations carried out, and so on. All of this occurs in the context of the suspended computation, which remains available to be resumed. See Chapter 14.
- DWIM The "Do What I Mean" package automatically fixes misspellings and errors in typing. See Chapter 20.
- Programmer's Assistant Medley keeps track of your actions during a session and allows each one to be replayed, undone, or altered. See Chapter 13.
- Masterscope Masterscope is a program analysis and management tool which can analyze users' functions and build (and automatically maintain) a data base of the results. This allows you to ask questions like "WHO CALLS ARCTAN" or "WHO USES COEF1 FREELY" or to request systematic changes like "EDIT WHERE ANY [function] FETCHES ANY FIELD OF [the data structure] FOO". See Chapter 19.
- Record/Datatype Package Medley allows you to define new data structures. This enables one to separate the issues of data access from the details of how the data is actually stored. See Chapter 8.
- File Manager Source code files in Medley are managed by the system, removing the problem of ensuring timely file updates from the user. The file manager can be modified and extended to accomodate new types of data. See Chapter 17.
- Performance Analysis These tools allow statistics on program operation to be collected and analyzed. See Chapter 22.
- Multiple Processes Multiple and independent processes simplify problems which require logically separate pieces of code to operate in parallel. See Chapter 23.

INTERLISP-D REFERENCE MANUAL

- Windows The ability to have multiple, independent windows on the display allows many different processes or activities to be active on the screen at once. See Chapter 28.
- Inspector The inspector is a display tool for examining complex data structures encountered during debugging. See Chapter 26.

These facilities are tightly integrated, so they know about and use each other, just as they can be used by user programs. For example, Masterscope uses the structural editor to make systematic changes. By combining the program analysis features of Masterscope with the features of the structural editor, large scale system changes can be made with a single command. For example, when the lowest-level interface of the Medley I/O system was changed to a new format, the entire edit was made by a single call to Masterscope of the form `EDIT WHERE ANY CALLS ' (BIN BOUT . . . )`. [Burton et al., 1980] This caused Masterscope to invoke the editor at each point in the system where any of the functions in the list `' (BIN BOUT . . . )` were called. This ensured that no functions used in input or output were overlooked during the modification.

Philosophy

Medley's extensive environmental support has developed over the years to support a particular style of programming called "exploratory programming" [Sheil, 1983]. For many complex programming problems, the task of program creation is *not* simply one of writing a program to fulfill specifications. Instead, it is a matter of exploring the problem (trying out various solutions expressed as partial programs) until one finds a good solution (or sometimes, any solution at all!). Such programs are by nature evolutionary; they are transformed over time from one realization to another in response to a growing understanding of the problem. This point of view has led to an emphasis on having the tools available to analyze, alter, and test programs easily. One important aspect of this is that the tools be designed to work together in an integrated fashion, so that knowledge about the user's programs, once gained, is available throughout the environment.

The development of programming tools to support exploratory programming is itself an exploration. No one knows all the tools that will eventually be found useful, and not all programmers want all of the tools to behave the same way. In response to this diversity, Interlisp has been shaped, by its implementors and by its users, to be easily extensible in several different ways. First, there are many places in the system where its behavior can be adjusted by the user. One way that this can be done is by changing the value of various "flags" or variables whose values are examined by system code to enable or suppress certain behavior. The other is where the user can provide functions or other behavioral specifications of what is to happen in certain contexts. For example, the format used for each type of list structure when it is printed by the pretty-printer is determined by specifications that are found on the list `PRETTYPRINTMACROS`. Thus, this format can be changed for a given type simply by putting a printing specification for it on that list.

Another way in which users can affect Medley's behavior is by redefining or changing system functions. The "Advise" capability, for instance, lets you modify the operation of virtually any function in the system by wrapping code "around" the selected function. (This same philosophy extends to breaking and tracing, so almost any function in the system can be broken or traced.) Since

INTRODUCTION

the entire system is implemented in Lisp, there are few places where the system's behavior depends on anything that you can't modify (such as a low level system implementation language).

While these techniques provide a fair amount of tailorability, there's a price: Medley is complex. There are many flags, parameters, and controls that affect its behavior. Because of this complexity, Interlisp tends to be more comfortable for experts, rather than casual users. Beginning users of Interlisp should depend on the default settings of parameters until they learn what dimensions of flexibility are available. At that point, they can begin to "tune" the system to their preferences.

Appropriately enough, even Medley's underlying philosophy was itself discovered during Medley's development, rather than laid out beforehand. The Medley environment and its interactive style were first analyzed in Sandewall's excellent paper [Sandewall, 1978]. The notion of "exploratory programming" and the genesis of the Interlisp programming tools in terms of the characteristic demands of this style of programming was developed in [Sheil, 1983]. The evolution and structure of the Interlisp programming environment are discussed in greater depth in [Teitelman & Masinter, 1981].

How to Use this Manual

This document is a reference manual, not a primer. We have tried to provide a manual that is complete, and that lets you find particular items as easily as possible. Sometimes, these goals have been achieved at the expense of simplicity. For example, many functions have a number of arguments that are rarely used. In the interest of providing a complete reference, these arguments are fully explained, even though you will normally let them default. There is a lot of information in this manual that is of interest only to experts.

Do not try to read straight through this manual, like a novel. In general, the chapters are organized with overview explanations and the most useful functions at the beginning of the chapter, and implementation details towards the end. If you are interested in becoming acquainted with Medley, we urge you to work through *An Introduction to Medley* before attempting this manual.

A few comments about the notational conventions used in this manual:

Lisp object notation: All Interlisp objects in this manual are printed in the same font: Functions (AND, PLUS, DEFINEQ, LOAD); Variables (MAX.INTEGER, FILELST, DFNFLG); and arbitrary Interlisp expressions: (PLUS 2 3), (PROG ((A 1)) ...), etc.

Case is significant: *In Interlisp, upper and lower case is significant.* The variable FOO is not the same as the variable foo or the variable Foo. By convention, most Interlisp system functions and variables are all uppercase, but users are free to use upper and lower case for their own functions and variables as they wish.

One exception to the case-significance rule is provided by the CLISP facility, which lets you type iterative statements and record operations in either all uppercase or all lowercase letters: (for X

INTERLISP-D REFERENCE MANUAL

from 1 to 5 ...) is the same as (FOR X FROM 1 TO 5 ...). The few situations where this is the case are explicitly mentioned in the manual. Generally, assume that case is significant.

This manual contains a large number of descriptions of functions, variables, commands, etc, which are printed in the following standard format:

(FOO BAR BAZ)	[Function]

This is a description for the function named FOO. FOO has two arguments, BAR and BAZ. Some system functions have extra optional arguments that are not documented and should not be used. These extra arguments are indicated by "—".

The descriptor [Function] indicates that this is a function, rather than a [Variable], [Macro], etc. For function definitions only, this can also indicate whether the function takes a fixed or variable number of arguments, and whether the arguments are evaluated or not. [Function] indicates a lambda spread function (fixed number of arguments, evaluated), the most common type.

References

[Burton, et al., 1980]	Burton, R. R., L. M. Masinter, A. Bell, D. G. Bobrow, W. S. Haugeland, R.M. Kaplan and B.A. Sheil, "Interlisp-D: Overview and Status" — in [Sheil & Masinter, 1983].
[Sandewall, 1978]	Sandewall, Erik, "Programming in the Interactive Environmnet: The LISP Experience" — <i>ACM Computing Surveys</i> , vol 10, no 1, pp 35-72, (March 1978).
[Sheil, 1983]	Sheil, B.A., "Environments for Exploratory Programming" — <i>Datamation</i> , (February, 1983) — also in [Sheil & Masinter, 1983].
[Sheil & Masinter, 1983]	Sheil, B.A. and L. M. Masinter, "Papers on Interlisp-D", Xerox PARC Technical Report CIS-5 (Revised), (January, 1983).
[Teitelman & Masinter, 1981]	Teitelman, W. and L. M. Masinter, "The Interlisp Programming Environment" — <i>Computer</i> , vol 14, no 4, pp 25-34, (April 1981) — also in [Sheil & Masinter, 1983].

2. SYMBOLS (LITATOMS)

A litatom (for “literal atom”) is an object that conceptually consists of a print name, a value, a function definition, and a property list. Litatoms are also known as “symbols” in Common Lisp. For clarity, we will use the term “symbol”.

A symbol is read as any string of non-delimiting characters that cannot be interpreted as a number. The syntactic characters that delimit symbols are called “separator” or “break” characters (see Chapter 25) and normally are space, end-of-line, line-feed, left parenthesis (, right parenthesis), double quote ", left square bracket [, and right square bracket]. However, any character may be included in a symbol by preceding it with the character %. Here are some examples of symbols:

```
A wxyz 23SKIDDOO %]  
Long% Litatom% With% Embedded% Spaces
```

(LITATOM X) [Function]

Returns T if X is a symbol, NIL otherwise. Note that a number is not a symbol.

```
(LITATOM NIL) = T
```

(ATOM X) [Function]

Returns T if X is an atom (i.e., a symbol or a number) or NIL (e.g. (ATOM NIL) = T); otherwise returns NIL.

Warning: (ATOM X) is NIL if X is an array, string, etc. In Common Lisp, the function CL:ATOM is defined equivalent to the Interlisp function NLISTP.

Each symbol has a print name, a string of characters that uniquely identifies that symbol: Those characters that are output when the symbol is printed using PRIN1, e.g., the print name of the symbol ABC%D consists of the five characters ABC%D.

Symbols are unique: If two symbols print the same, they will always be EQ. Note that this is not true for strings, large integers, floating-point numbers, etc.; they all can print the same without being EQ. Thus, if PACK or MKATOM is given a list of characters corresponding to a symbol that already exists, they return a pointer to that symbol, and do not make a new symbol. Similarly, if the read program is given as input a sequence of characters for which a symbol already exists, it returns a pointer to that symbol.

Symbol names are limited to 255 characters. Attempting to create a larger symbol will cause an error: Atom too long.

Sometimes we'll refer to a “PRIN2-name”. The PRIN2-name of a symbol is those characters output when it is printed using PRIN2. So the PRIN2-name of the symbol ABC%D is the six characters ABC%D. The PRIN2-name depends on what readtable is being used (see Chapter 25), since this determines where %s will be inserted. Many of the functions below allow either print names or PRIN2-names to be used, as specified by FLG and RDTBL arguments. If FLG is NIL, print names are used. Otherwise, PRIN2-names are used, computed with respect to the readtable RDTBL (or the current readtable, if RDTBL = NIL).

INTERLISP-D REFERENCE MANUAL

(MKATOM *X*) [Function]

Creates and returns a symbol whose print name is the name as that of the string *X* or, if *X* is not a string, the same as that of (MKSTRING *X*). Examples:

```
(MKATOM ' (A B C) ) => % (A% B% C%)
(MKATOM "1.5") => 1.5
```

Note that the last example returns a number, not a symbol. It is a deeply-ingrained feature of Interlisp that no symbol can have the print name of a number.

(SUBATOM *X N M*) [Function]

Returns a symbol made from the *N*th through *M*th characters of the print name of *X*. If *N* or *M* are negative, they specify positions counting backwards from the end of the print name. Equivalent to (MKATOM (SUBSTRING *X N M*)). Examples:

```
(SUBATOM "FOO1.5BAR" 4 6) => 1.5
(SUBATOM ' (A B C) 2 -2) => A% B% C
```

(PACK *X*) [Function]

If *X* is a list of symbols, PACK returns a single symbol whose print name is the concatenation of the print names of the symbols in *X*. If the concatenated print name is the same as that of a number, PACK returns that number. For example:

```
(PACK ' (A BC DEF G) ) => ABCDEFG
(PACK ' (1 3.4) ) => 13.4
(PACK ' (1 E -2) ) => .01
```

Although *X* is usually a list of symbols, it can be a list of arbitrary objects. The value of PACK is still a single symbol whose print name is the concatenation of the print names of all the elements of *X*, e.g.,

```
(PACK ' ( (A B) "CD" ) ) => % (A% B%)CD
```

If *X* is not a list or NIL, PACK generates the error `Illegal arg.`

(PACK* *X X ... X*) [NoSpread Function]

Version of PACK that takes an arbitrary number of arguments, instead of a list. Examples:

```
(PACK* 'A 'BC 'DEF 'G => ABCDEFG
(PACK* 1 3.4) => 13.4
```

(GENSYM *PREFIX* - - -) [Function]

Returns a symbol of the form *Xnnnn*, where *X* = *PREFIX* (or A if *PREFIX* is NIL) and *nnnn* is an integer. Thus, the first one generated is A0001, the second A0002, etc. The integer suffix is always at least four characters long, but it can grow beyond that. For example, the next symbol produced after A9999 would be A10000. GENSYM provides a way of generating symbols for various uses within the system.

Note: The Common Lisp function CL:GENSYM is not the same as Interlisp's GENSYM. Interlisp always creates interned symbols whereas CL:GENSYM creates uninterned symbols.

SYMBOLS (LITATOMS)

GENNUM

[Variable]

The value of GENNUM, initially 0, determines the next GENSYM, e.g., if GENNUM is set to 23, (GENSYM) = A0024.

The term “gensym” is used to indicate a symbol that was produced by the function GENSYM. Symbols generated by GENSYM are the same as any other symbols: they have property lists, and can be given function definitions. The symbols are not guaranteed to be new. For example, if the user has previously created A0012, either by typing it in, or via PACK or GENSYM itself, then if GENNUM is set to 11, the next symbol returned by GENSYM will be the A0012 already in existence.

(MAPATOMS FN)

[Function]

Applies *FN* (a function or lambda expression) to every symbol in the system. Returns NIL. For example:

```
(MAPATOMS (FUNCTION (LAMBDA(X) (if (GETD X) then (PRINTX))
```

will print every symbol with a function definition.

Warning: Be careful if *FN* is a lambda expression or an interpreted function: since NOBIND is a symbol, it will eventually be passed as an argument. The first reference to that argument within the function will signal an error.

A way around this problem is to use a Common Lisp function, so that the Common Lisp interpreter will be invoked. It will treat the argument as local, not special and no error will be signaled. An alternative solution is to include the argument to the Interlisp function in a LOCALVARS declaration and then compile the function before passing it to MAPATOMS. This will significantly speed up MAPATOMS.

(APROPOS STRING ALLFLG QUITFLG OUTPUT)

[Function]

APROPOS scans all symbols in the system for those which have *STRING* as a substring and prints them on the terminal along with a line for each relevant item defined for each selected symbol. Relevant items are:

- function definitions, for which only the arglist is printed
- dynamic variable values
- non-null property lists

PRINTLEVEL (see Chapter 25) is set to (3 . 5) when APROPOS is printing.

If *ALLFLG* is NIL, then symbols with no relevant items and “internal” symbols are omitted (“internal” currently means those symbols whose print name begins with a \ or those symbols produced by GENSYM). If *ALLFLG* is a function, it is used as a predicate on symbols selected by the substring match, with value NIL meaning to omit the symbol. If *ALLFLG* is any other non-NIL value, then no symbols are omitted.

Note: Unlike CL:APROPOS which lets you designate the package to search, APROPOS searches *all* packages.

Using Symbols as Variables

Symbols are commonly used as variable names. Each symbol has a “top level” value, which can be an arbitrary object. Symbols may also be given special variable bindings within PROGS or functions, which only exist for the duration of the function. When a symbol is evaluated, the “current” variable binding is returned. This is the most recent special variable binding, or the top-level binding if the symbol hasn’t been rebound. SETQ is used to change the current binding. For more information on variable bindings in Interlisp, see Chapter 11.

A symbol whose top-level value is the symbol NOBIND is considered to have no value. If a symbol has no local bindings, and its top-level value is NOBIND, trying to evaluate it will cause an unbound-atom error. In addition, if a symbol’s local binding is to NOBIND, trying to evaluate it will cause an error.

The symbols T and NIL always evaluate to themselves. Attempting to change the value of T or NIL with the functions below will generate the error; Attempt to set T or Attempt to set NIL.

The following functions (except BOUNDP) will also generate the error Arg not litatom, if not given a symbol.

(BOUNDP VAR) [Function]

Returns T if VAR has a special variable binding, or if VAR has a top-level value other than NOBIND; otherwise NIL. That is, if X is a symbol, (EVAL X) will cause an Unbound atom error if and only if (BOUNDP X) returns NIL.

Note: The Interlisp interpreter has been modified so that it will generate an Unbound Variable error when it encounters any symbol bound to NOBIND. This is a change from previous releases that only signaled an error when a symbol had a top-level binding of NOBIND in addition to no dynamic binding.

(SET VAR VALUE) [NoSpread Function]

Sets the “current” value of VAR to VALUE, and returns VALUE.

SET is a normal function, so both VAR and VALUE are evaluated before it is called. Thus, if the value of X is B, and value of Y is C, then (SET X Y) would result in B being set to C, and C being returned as the value of SET.

(SETQ VAR VALUE) [NoSpread Function]

Like SET, but VAR is not evaluated, VALUE is. Thus, if the value of X is B and the value of Y is C, (SETQ X Y) would result in X (not B) being set to C, and C being returned.

Actually, neither argument is evaluated during the calling process. However, SETQ itself calls EVAL on its second argument. As a result, typing (SETQ VAR FORM) and SETQ (VAR FORM) to the Interlisp Executive are equivalent: in both cases VAR is not evaluated, and FORM is.

(SETQQ VAR VALUE) [NoSpread Function]

Like SETQ, but neither argument is evaluated, e.g., (SETQQ X (A B C)) sets X to (A B C).

SYMBOLS (LITATOMS)

(**PSETQ** *VAR* *VALUE* ... *VAR* *VALUE*) [Macro]

Does a SETQ in parallel of *VAR* (unevaluated) to *VALUE* , *VAR* to *VALUE* , etc. All of the *VALUE* terms are evaluated before any of the assignments. Therefore, (PSETQ A B B A) can be used to swap the values of the variables A and B.

(**GETTOPVAL** *VAR*) [Function]

Returns the top level value of *VAR* (even if NOBIND), regardless of any intervening local bindings.

(**SETTOPVAL** *VAR* *VALUE*) [Function]

Sets the top level value of *VAR* to *VALUE*, regardless of any intervening bindings, and returns *VALUE*.

(**GETATOMVAL** *VAR*) [Function]

Same as (GETTOPVAL *VAR*) .

(**SETATOMVAL** *VAR* *VALUE*) [Function]

Same as SETTOPVAL.

Note: The compiler (see Chapter 18) treats variables somewhat differently from the interpreter, and you need to be aware of these differences when writing functions that will be compiled. For example, variable references in compiled code are not checked for NOBIND, so compiled code will not generate unbound-atom errors. In general, it is better to debug interpreted code, before compiling it for speed. The compiler offers some facilities to increase the efficiency of variable use in compiled functions: Global variables can be defined so that the entire stack is not searched at each variable reference. Local variables have bindings that are not visible outside the function, which reduces variable conflicts and makes variable lookup faster.

Function Definition Cells

Each symbol has a function-definition cell, which is accessed when that symbol is used as a function. This is described in detail in Chapter 10.

Property Lists

Each symbol has an associated property list, which allows a set of named objects to be associated with the symbol. A property list associates a name (known as a “property name” or “property”) with an arbitrary object (the “property value” or “value”). Sometimes the phrase “to store on the property X” is used, meaning to place the indicated information on a property list under the property name X.

Property names are usually symbols or numbers, although no checks are made. However, the standard property list functions all use EQ to search for property names, so they may not work with non-atomic property names. The same object can be used as both a property name and a property value.

Many symbols in the system already have property lists, with properties used by the compiler, the break package, DWIM, etc. Be careful not to clobber such system properties. The variable SYSPROPS is a list of property names used by the system.

INTERLISP-D REFERENCE MANUAL

The functions below are used to manipulate the property lists of symbols. Except when indicated, they generate the error *ATM is not a SYMBOL*, if given an object that is not a symbol.

(**GETPROP** *ATM PROP*) [Function]

Returns the property value for *PROP* from the property list of *ATM*. Returns *NIL* if *ATM* is not a symbol, or *PROP* is not found. **GETPROP** also returns *NIL* if there is an occurrence of *PROP* but the corresponding property value is *NIL*. This can be a source of program errors.

Note: **GETPROP** used to be called **GETP**.

(**PUTPROP** *ATM PROP VAL*) [Function]

Puts the property *PROP* with value *VAL* on the property list of *ATM*. *VAL* replaces any previous value for the property *PROP* on this property list. Returns *VAL*.

(**ADDPROP** *ATM PROP NEW FLG*) [Function]

Adds the value *NEW* to the list which is the value of property *PROP* on the property list of the *ATM*. If *FLG* is *T*, *NEW* is *CONSED* onto the front of the property value of *PROP*; otherwise, it is *NCONCed* on the end (using *NCONC1*). If *ATM* does not have a property *PROP*, or the value is not a list, then the effect is the same as (**PUTPROP** *ATM PROP (LIST NEW)*). **ADDPROP** returns the (new) property value. Example:

```
← (PUTPROP 'POCKET 'CONTENTS NIL)
(NIL)
← (ADDPROP 'POCKET 'CONTENTS 'COMB)
(COMB)
← (ADDPROP 'POCKET 'CONTENTS 'WALLET)
(COMB WALLET)
```

(**REMPROP** *ATM PROP*) [Function]

Removes all occurrences of the property *PROP* (and its value) from the property list of *ATM*. Returns *PROP* if any were found (*T* if *PROP* is *NIL*), otherwise *NIL*.

(**CHANGEPROP** *X PROP1 PROP2*) [Function]

Changes the property name of property *PROP1* to *PROP2* on the property list of *X* (but does not affect the value of the property). Returns *X*, unless *PROP1* is not found, in which case it returns *NIL*.

(**PROPNames** *ATM*) [Function]

Returns a list of the property names on the property list of *ATM*.

(**DEFLIST** *L PROP*) [Function]

Used to put values under the same property name on the property lists of several symbols. *L* is a list of two-element lists. The first element of each is a symbol, and the second element is the property value of the property *PROP*. Returns *NIL*. For example:

```
(DEFLIST ' ((FOO MA) (BAR CA) (BAZ RI)) 'STATE)
```

SYMBOLS (LITATOMS)

puts MA on FOO's STATE property, CA on BAR's STATE property, and RI on BAZ's STATE property.

Property lists are conventionally implemented as lists of the form

(NAME VALUE NAME VALUE ...)

although the user can store anything as the property list of a symbol. However, the functions which manipulate property lists observe this convention by searching down the property lists two CDRs at a time. Most of these functions also generate the error `Arg not litatom` if given an argument which is not a symbol, so they cannot be used directly on lists. (`LISTPUT`, `LISTPUT1`, `LISTGET`, and `LISTGET1` are functions similar to `PUTPROP` and `GETPROP` that work directly on lists (see Chapter 3). The property lists of symbols can be directly accessed with the following functions.

(**GETPROPLIST** *ATM*) [Function]

Returns the property list of *ATM*.

(**SETPROPLIST** *ATM LST*) [Function]

If *ATM* is a symbol, sets the property list of *ATM* to be *LST*, and returns *LST* as its value.

(**GETLIS** *X PROPS*) [Function]

Searches the property list of *X*, and returns the property list as of the first property on *PROPS* that it finds. For example:

```
← (GETPROPLIST 'X)
  (PROP1 A PROP3 B A C)
← (GETLIS 'X ' (PROP2 PROP3))
  (PROP3 B A C)
```

Returns NIL if no element on props is found. *X* can also be a list itself, in which case it is searched as described above. If *X* is not a symbol or a list, returns NIL.

(**REMPROPLIST** *ATM PROPS*) [Function]

Removes all occurrences of all properties on the list *PROPS* (and their corresponding property values) from the property list of *ATM*. Returns NIL.

Print Names

The term "print name" has an extended meaning: The characters that are output when *any object* is printed. In Medley, all objects have print names, although only symbols and strings have their print names explicitly stored. Symbol print names are limited to 255 characters.

This section describes a set of functions that can be used to access and manipulate the print names of any object, though they are primarily used with the print names of symbols. In Medley, print functions qualify symbol names with a package prefix if the symbol is not accessible in the current package. The exception is Interlisp's `PRIN1`, which does not include a package prefix.

The print name of an object is those characters that are output when the object is printed using `PRIN1`, e.g., the print name of the list (A B "C") consists of the seven characters (A B C) (two of the characters are spaces).

INTERLISP-D REFERENCE MANUAL

The PRIN2-name of an object is those characters output when the object is printed using PRIN2. Thus the PRIN2-name of the list (A B "C") is the 9 characters (A B "C") (including the two spaces). The PRIN2-name depends on what readtable is being used (see Chapter 25), since this determines where %s will be inserted. Many of the functions below allow either print names or PRIN2-names to be used, as specified by FLG and RDTBL arguments. If FLG is NIL, print names are used. Otherwise, PRIN2-names are used, computed with respect to the readtable RDTBL (or the current readtable, if RDTBL = NIL).

The print name of an integer depends on the setting of RADIX (see Chapter 25). The functions described in this section (UNPACK, NCHARS, etc.) define the print name of an integer as though the radix was 10, so that (PACK (UNPACK 'X9)) will always be X9 (and not X11, if RADIX is set to 8). However, integers will still be printed by PRIN1 using the current radix. The user can force these functions to use print names in the current radix by changing the setting of the variable PRXFLG (see Chapter 25).

(CL:SYMBOL-NAME *SYM*) [Common Lisp Function]

Returns a string displaced to the *SYM* print name. Strings returned from CL:SYMBOL-NAME may be destructively modified without affecting *SYM*'s print name.

(NCHARS *X FLG RDTBL*) [Function]

Returns the number of characters in the print name of *X*. If *FLG* = T, the PRIN2-name is used. Examples:

```
(NCHARS 'ABC) => 3
(NCHARS "ABC" T) => 5
```

NCHARS works most efficiently on symbols and strings, but can be given any object.

(NTHCHAR *X N FLG RDTBL*) [Function]

Returns *X*, if *X* is a tail of the list *Y*; otherwise NIL. *X* is a tail of *Y* if it is EQ to 0 or more CDRs of *Y*.

```
(NTHCHAR 'ABC 2) => B
(NTHCHAR 15.6 2) => 5
(NTHCHAR 'ABC% (D -3 T) => %%
(NTHCHAR "ABC" 2) => B
(NTHCHAR "ABC" 2 T) => A
```

NTHCAR and NCHARS work much faster on objects that actually have an internal representation of their print name, i.e., symbols and strings, than they do on numbers and lists, since they don't have to simulate printing.

(L-CASE *X FLG*) [Function]

Returns a lowercase version of *X*. If *FLG* is T, the first letter is capitalized. If *X* is a string, the value of L-CASE is also a string. If *X* is a list, L-CASE returns a new list in which L-CASE is computed for each corresponding element and non-NIL tail of the original list. Examples:

```
(L-CASE 'FOO) => foo
(L-CASE 'FOO T) => Foo
(L-CASE "FILE NOT FOUND" T) => "File not found"
```

SYMBOLS (LITATOMS)

```
(L-CASE ' (JANUARY FEBRUARY (MARCH "APRIL")) T) =>  
' (January February (March "April"))
```

(**U-CASE** *X*) [Function]

Like L-CASE, but returns the uppercase version of *X*.

(**U-CASEP** *X*) [Function]

Returns T if *X* contains no lowercase letters; NIL otherwise.

Characters and Character Codes

Characters are represented 3 different ways in Medley. In Interlisp they are single-character symbols or integer character codes. In Common Lisp they are instances of the CHARACTER datatype. In general Interlisp character functions don't accept Common Lisp characters and vice versa. The only exceptions are Interlisp string-manipulation functions that accept "string or symbol" types as arguments.

You can convert between Interlisp and Common Lisp characters by using the functions CL:CODE-CHAR, CL:CHAR-CODE, and CHARCODE (see below).

Medley uses the 16-bit NS character set, described in the document Character Code Standard (Xerox System Integration Standards, XSI 058404, April 1984). Legal character codes range from 0 to 65535. The NS (Network Systems) character encoding encompasses a much wider set of available characters than the 8-bit character standards (such as ASCII), including characters comprising many foreign alphabets and special symbols. For instance, Medley supports the display and printing of the following:

- Le système d'information Medley est remarquablement polyglotte
- Das Medley Kommunikationssystem bietet merkwürdige multilinguale Nutzungsmöglichkeiten
- $M \subseteq \square[w] \Leftrightarrow \forall v \text{ with } R_{wv}: M \subseteq [v]$

These characters can be used in strings, symbol print names, symbolic files, or anywhere else 8-bit characters could be used. All of the standard string and print name functions (RPLSTRING, GNC, NCHARS, STRPOS, etc.) accept symbols and strings containing NS characters. For example:

```
← (STRPOS "char" "this is an 8-bit character string")  
18  
← (STRPOS "char" "celui-ci comporte des caractères NS")  
23
```

In almost all cases, a program does not have to distinguish between NS characters or 8-bit characters. The exception to this rule is the handling of input/output operations (see Chapter 25).

The function CHARCODE (see below) provides a simple way to create individual NS character codes. The VirtualKeyboards library module provides a set of virtual keyboards that allows keyboard or mouse entry of NS characters.

(**PACKC** *X*) [Function]

Like PACK except *X* is a list of character codes. For example,

```
(PACKC ' (70 79 79)) => FOO
```

INTERLISP-D REFERENCE MANUAL

(**CHCON** *X FLG RDTBL*) [Function]

Like UNPACK, but returns the print name of *X* as a list of character codes. If *FLG* = T, the PRIN2-name is used. For example:

(CHCON 'FOO) => (70 79 79)

(**DCHCON** *X SCRATCHLIST FLG RDTBL*) [Function]

Like DUNPACK.

(**NTHCHARCODE** *X N FLG RDTBL*) [Function]

Like NTHCHAR, but returns the character code of the *N*th character of the print name of *X*. If *N* is negative, it is interpreted as a count backwards from the end of *X*. If the absolute value of *N* is greater than the number of characters in *X*, or 0, then the value of NTHCHARCODE is NIL.

If *FLG* is T, then the PRIN2-name of *X* is used, computed with respect to the readtable.

(**CHCON1** *X*) [Function]

Returns the character code of the first character of the print name of *X*; equal to (NTHCHARCODE *X* 1).

(**CHARACTER** *N*) [Function]

N is a character code. Returns the symbol having the corresponding single character as its print name.

(CHARACTER 70) => F

(**FCHARACTER** *N*) [Function]

Fast version of CHARACTER that compiles open.

The following function makes it possible to gain the efficiency that comes from dealing with character codes without losing the symbolic advantages of character symbols.

(**CHARCODE** *CHAR*) [Function]

Returns the character code specified by *CHAR* (unevaluated). If *CHAR* is a one-character symbol or string, the corresponding character code is simply returned. Thus, (CHARCODE A) is 65, (CHARCODE 0) is 48. If *CHAR* is a multi-character symbol or string, it specifies a character code as described below. If *CHAR* is NIL, CHARCODE simply returns NIL. Finally, if *CHAR* is a list structure, the value is a copy of *CHAR* with all the leaves replaced by the corresponding character codes. For instance, (CHARCODE (A (B C))) => (65 (66 67)).

If a character is specified by a multi-character symbol or string, CHARCODE interprets it as follows:

CR, SPACE, etc.

SYMBOLS (LITATOMS)

The variable CHARACTERNAMES contains an association list mapping special symbols to character codes. Among the characters defined this way are CR (13), LF (10), SPACE or SP (32), ESCAPE or ESC (27), BELL (7), BS (8), TAB (9), NULL (0), and DEL (127). The symbol EOL maps into the appropriate end-of-line character code in the different Interlisp implementations (31 in Interlisp-10, 13 in Interlisp-D, 10 in Interlisp-VAX). Examples:

```
(CHARCODE SPACE) => 32
(CHARCODE CR)   => 13
```

CHARSET, CHARNUM, CHARSET-CHARNUM

If the character specification is a symbol or string of the form CHARSET, CHARNUM, or CHARSET-CHARNUM, the character code for the character number CHARNUM in the character set CHARSET is returned.

The 16-bit NS character encoding is divided into a large number of “character sets”. Each 16-bit character can be decoded into a character set (an integer from 0 to 254 inclusive) and a character number (also an integer from 0 to 254 inclusive). CHARSET is either an octal number, or a symbol in the association list CHARACTERSETNAMES (which defines the character sets for GREEK, CYRILLIC, etc.).

CHARNUM is either an octal number, a single-character symbol, or a symbol from the association list CHARACTERNAMES. If CHARNUM is a single-digit number, it is interpreted as the character “2”, rather than as the octal number 2. Examples:

```
(CHARCODE 12,6)  => 2566
(CHARCODE 12,SPACE) => 2592
(CHARCODE GREEK,A) => 9793
```

↑CHARSPEC (control chars)

If the character specification is a symbol or string of one of the forms above, preceded by the character ↑, this indicates a “control character,” derived from the normal character code by clearing the seventh bit of the character code (normally set). Examples:

```
(CHARCODE ↑A)    => 1
(CHARCODE ↑GREEK,A) => 9729
```

#CHARSPEC (meta chars)

If the character specification is a symbol or string of one of the forms above, preceded by the character #, this indicates a meta character, derived from the normal character code by setting the eighth bit of the character code (normally cleared). ↑ and # can both be set at once. Examples:

```
(CHARCODE #A)    => 193
(CHARCODE #↑GREEK,A) => 9857
```

A CHARCODE form can be used wherever a structure of character codes would be appropriate. For example:

INTERLISP-D REFERENCE MANUAL

```
(FMEMB (NTHCHARCODE X 1) (CHARCODE (CR LF SPACE ↑A)))  
(EQ (READCCODE FOO) (CHARCODE GREEK, A))
```

There is a macro for CHARCODE which causes the character-code structure to be constructed at compile-time. Thus, the compiled code for these examples is exactly as efficient as the less readable:

```
(FMEMB (NTHCHARCODE X 1) (QUOTE (13 10 32 1)))  
(EQ (READCCODE FOO) 9793)
```

(CL:CHAR-CODE CHAR) [Common Lisp Function]

Returns the Interlisp character code of *CHAR*. Use to convert a Common Lisp character to an Interlisp character code.

(CL:CODE-CHAR N) [Common Lisp Function]

Returns a character with the given non-negative integer *N* code. Returns NIL if no character is possible with *N*. Use to convert an Interlisp character code to a Common Lisp character.

(SELCHARQ E CLAUSE ... CLAUSE DEFAULT) [Function]

Lets you branch one of several ways, based on the character code *E*. The first item in each *CLAUSE* is a character code or list of character codes, given in the form CHARCODE would accept. If the value of *E* is a character code or NIL, and it is EQ or MEMB to the result of applying CHARCODE to the first element of a clause, the remaining forms of that clause are evaluated. Otherwise, the default is evaluated.

Thus

```
(SELCHARQ (BIN FOO))  
  ((SPACE TAB) (FUM))  
  ((↑D NIL) (BAR))  
  (a (BAZ))  
  (ZIP))
```

is exactly equivalent to

```
(SELECTQ (BIN FOO))  
  ((32 9) (FUM))  
  ((4 NIL) (BAR))  
  (97 (BAZ))  
  (ZIP))
```

If (BIN FOO) returned 32 (the SPACE character), the function FUM would be called.

SYMBOLS (LITATOMS)

[This page intentionally left blank]

TABLE of CONTENTS

Volume 1 - Language Reference

1. Introduction	1
2. Litatoms (Symbols)	2-1
Using Symbols as Variables	2-1
Function Definition Cells.....	2-3
Property Lists	2-4
Print Names.....	2-5
Characters and Character Codes.....	2-9
3. Lists	3-1
Creating Lists.....	3-3
Building Lists from Left to Right.....	3-4
Copying Lists	3-6
Extracting Tails of Lists.....	3-6
Counting List Cells	3-8
Logical Operations	3-9
Searching Lists	3-10
Substitution Functions	3-10
Association Lists and Property Lists.....	3-11
Sorting Lists	3-13
Other List Functions.....	3-15
4. Strings	4-1
5. Arrays	5-1
6. Hash Arrays	6-1
Hash Overflow	6-3
User-Specified Hashing Functions.....	6-3
7. Numbers and Arithmetic Functions	7-1
Generic Arithmetic	7-2
Integer Arithmetic	7-3
Logical Arithmetic Functions.....	7-6
Floating-Point Arithmetic.....	7-8
Other Arithmetic Functions	7-10

8. Record Package	8-1
FETCH and REPLACE	8-1
CREATE	8-2
TYPE?	8-3
WITH	8-4
Record Declarations	8-4
Record Types	8-5
Optional Record Specifications	8-10
Defining New Record Types	8-12
Record Manipulation Functions	8-12
Changetran	8-13
Built-in and User Data Types	8-15
9. Conditionals and Iterative Statements	9-1
Data Type Predicates	9-1
Equality Predicates	9-2
Logical Predicates	9-3
COND Conditional Function	9-3
The IF Statement	9-4
Selection Functions	9-5
PROG and Associated Control Functions	9-6
The Iterative Statement	9-7
I.s. Types	9-8
Iterative Variable I.s.oprs	9-9
Condition I.s.oprs	9-12
Other I.s.oprs	9-13
Miscellaneous Hints on I.s.oprs	9-13
Errors in Iterative Statements	9-15
Defining New Iterative Statement Operators	9-15
10. Function Definition, Manipulation, and Evaluation	10-1
Function Types	10-2
Lambda-Spread Functions	10-2
Nlambda-Spread Functions	10-3
Lambda-Nospread Functions	10-4
Nlambda-Nospread Functions	10-4
Compiled Functions	10-5
Function Type Functions	10-5
Defining Functions	10-7
Function Evaluation	10-1
Iterating and Mapping Functions	10-1
Function Arguments	10-1
Macros	10-1
DEFMACRO	10-15
Interpreting Macros	10-15

11. Variable Binds and the Interlisp Stack	11-1
Spaghetti Stack	11-2
Stack Functions	11-3
Searching the Stack	11-4
Variable Binds in Stack Frames	11-5
Evaluating Expressions in Stack Frames	11-6
Altering Flow of Control	11-6
Releasing and Reusing Stack Pointers	11-7
Backtrace Functions	11-8
Other Stack Functions	11-10
The Stack and the Interpreter	11-10
Generators	11-12
Coroutines	11-14
Possibilities Lists	11-15
12. Miscellaneous	12-1
Greeting and Initialization Files	12-1
Idle Mode	12-3
Saving Virtual Memory State	12-5
System Version Information	12-9
Date and Time Functions	12-11
Timers and Duration Functions	12-13
Resources	12-15
A Simple Example	12-16
Trade-offs in More Complicated Cases	12-18
Macros for Accessing Resources	12-18
Saving Resources in a File	12-19
Pattern Matching	12-19
Pattern Elements	12-20
Element Patterns	12-20
Segment Patterns	12-21
Assignments	12-23
Place-Markers	12-23
Replacements	12-24
Reconstruction	12-24
Examples	12-25

Volume 2 - Environment Reference

13. Interlisp Executive	13-1
Input Formats	13-3
Programmer's Assistant Commands	13-4
Event Specification	13-4

Commands	13-6
P.A. Commands Applied to P.A. Commands.....	13-15
Changing the Programmer's Assistant	13-16
Undoing	13-19
Undoing Out of Order	13-20
SAVESET	13-21
UNDONLSETQ and RESETUNDO.....	13-22
Format and Use of the History List	13-23
Programmer's Assistant Functions.....	13-26
The Editor and the Programmer's Assistant	13-32
14. Errors and Breaks	14-1
Breaks.....	14-1
Break Windows.....	14-2
Break Commands	14-3
Controlling When to Break	14-10
Break Window Variables.....	14-11
Creating Breaks with BREAK1	14-12
Signalling Errors.....	14-14
Catching Errors.....	14-16
Changing and Restoring System State	14-18
Error List.....	14-20
15. Breaking, Tracing, and Advising	15-1
Breaking Functions and Debugging.....	15-1
Advising	15-7
Implementation of Advising	15-7
Advise Functions.....	15-8
16. List Structure Editor	16-1
SEdit	16-1
Local Attention-Changing Commands.....	16-10
Commands That Search	16-14
Search Algorithm.....	16-15
Search Commands.....	16-16
Location Specification.....	16-18
Commands That Save and Restore the Edit Chain	16-21
Commands That Modify Structure.....	16-22
Implementation	16-23
The A, B, and : Commands	16-24
Form Oriented Editing and the Role of UP	16-26
Extract and Embed	16-26
The MOVE Command	16-28
Commands That Move Parentheses.....	16-30
TO and THRU	16-31

The R Command	16-34
Commands That Print.....	16-35
Commands for Leaving the Editor.....	16-37
Nested Calls to Editor	16-39
Manipulating the Characters of an Atom or String.....	16-39
Manipulating Predicates and Conditional Expressions.....	16-40
History Commands in the Editor	16-41
Miscellaneous Commands	16-41
Commands That Evaluate	16-43
Commands That Test	16-45
Edit Macros.....	16-46
Undo	16-48
EDITDEFAULT	16-50
Editor Functions.....	16-51
Time Stamps	16-57

17. File Package 17-1

Loading Files	17-3
Storing Files	17-8
Remaking a Symbolic File	17-12
Loading Files in a Distributed Environment	17-13
Marking Changes.....	17-13
Noticing Files.....	17-15
Distributing Change Information.....	17-16
File Package Types	17-16
Functions for Manipulating Typed Definitions	17-19
Defining New File Package Types	17-23
File Package Commands.....	17-25
Functions and Macros	17-26
Variables.....	17-27
Litatom Properties	17-29
Miscellaneous File Package Commands	17-30
DECLARE:	17-31
Exporting Definitions.....	17-33
FileVars.....	17-34
Defining New File Package Commands	17-35
Functions for Manipulating File Command Lists.....	17-37
Symbolic File Format.....	17-38
Copyright Notices.....	17-40
Functions Used Within Source Files	17-42
File Maps.....	17-42

18. Compiler 18-1

Compiler Printout.....	18-2
Global Variables.....	18-3

Local Variables and Special Variables.....	18-4
Constants	18-5
Compiling Function Calls	18-6
FUNCTION and Functional Arguments	18-7
Open Functions.....	18-8
COMPILETYPELST	18-8
Compiling CLISP.....	18-9
Compiler Functions.....	18-9
Block Compiling	18-12
Block Declarations.....	18-13
Block Compiling Functions.....	18-15
Compiler Error Messages.....	18-16
19. DWIM	20-1
Spelling Correction Protocol.....	20-3
Parentheses Errors Protocol.....	20-4
Undefined Function T Errors.....	20-4
DWIM Operation.....	20-5
DWIM Correction: Unbound Atoms.....	20-6
Undefined CAR of Form	20-7
Undefined Function in APPLY.....	20-8
DWIMUSERFORMS	20-8
DWIM Functions and Variables.....	20-10
Spelling Correction.....	20-11
Synonyms	20-12
Spelling Lists	20-12
Generators for Spelling Correction	20-14
Spelling Corrector Algorithm.....	20-14
Spelling Corrector Functions and Variables.....	20-15
20. CLISP	21-1
CLISP Interaction with User	21-4
CLISP Character Operators.....	21-5
Declarations.....	21-9
CLISP Operation.....	21-10
CLISP Translations.....	21-12
DWIMIFY	21-13
CLISPIFY	21-16
Miscellaneous Functions and Variables.....	21-18
CLISP Internal Conventions	21-20
21. Performance Issues	22-1
Storage Allocation and Garbage Collection	22-1
Variable Bindings	22-4
Performance Measuring	22-5

BREAKDOWN	22-7
GAINSPACE	22-9
Using Data Types Instead of Records.....	22-9
Using Incomplete File Names.....	22-10
Using "Fast" and "Destructive" Functions.....	22-10

22. Processes 23-1

Creating and Destroying Processes	23-1
Process Control Constructs	23-4
Events	23-5
Monitors	23-7
Global Resources.....	23-8
Typein and the TTY Process.....	23-9
Switing the TTY Process	23-9
Handling of Interrupts.....	23-11
Keeping the Mouse Alive	23-12
Process Status Window.....	23-12
Non-Process Compatibility	23-14

Volume 3 - I/O Reference

23. Streams and Files 24-1

Opening and Closing File Streams.....	24-1
File Names	24-4
Incomplete File Names	24-7
Version Recognition	24-9
Using File Names Instead of Streams	24-10
File Name Efficiency Considerations.....	24-11
Obsolete File Opening Functions	24-11
Converting Old Programs	24-11
Using Files with Processes	24-12
File Attributes.....	24-12
Closing and Reopening Files	24-15
Local Hard Disk Device	24-16
Floppy Disk Device	24-18
I/O Operations To and From Strings	24-22
Temporary Files and the CORE Device.....	24-23
NULL Device.....	24-24
Deleting, Copying, and Renaming Files.....	24-24
Searching File Directories	24-24
Listing File Directories	24-25
File Servers.....	24-28
PUP File Server Protocols	24-28

Xerox NS File Server Protocols.....	24-28
Operating System Designations	24-29
Logging In	24-30
Abnormal Conditions	24-31
24. Input/Output Functions	25-1
Specifying Streams for Input/Output Functions	25-1
Input Functions.....	25-2
Output Functions	25-6
PRINTLEVEL	25-8
Printing Numbers.....	25-10
User Defined Printing.....	25-12
Printing Unusual Data Structures.....	25-13
Random Access File Operations	25-14
Input/Output Operations with Characters and Bytes	25-17
PRINTOUT	25-17
Horizontal Spacing Commands	25-19
Vertical Spacing Commands	25-20
Special Formatting Controls	25-20
Printing Specifications	25-20
Paragraph Format	25-21
Right-Flushing	25-21
Centering	25-22
Numbering	25-22
Escaping to Lisp.....	25-23
User-Defined Commands	25-23
Special Printing Functions	25-24
READFILE and WRITEFILE	25-25
Read Tables	25-25
Read Table Functions.....	25-26
Syntax Classes.....	25-26
Read Macros.....	25-29
25. User Input/Output Packages	26-1
Inspector	26-1
Calling the Inspector	26-1
Multiple Ways of Inspecting.....	26-2
Inspect Windows.....	26-3
Inspect Window Commands	26-3
Interaction with Break Windows	26-4
Controlling the Amount Displayed During Inspection.....	26-4
Inspect Macros	26-4
INSPECTWs	26-5
PROMPTFORWORD	26-7
ASKUSER	26-9

Format of KEYLST	26-10
Options	26-12
Operation	26-13
Completing a Key	26-14
Special Keys	26-15
Startup Protocol and Typeahead	26-16
TTYIN Typein Editor	26-17
Entering Input with TTYIN	26-17
Mouse Commands (Interlisp-D Only).....	26-19
Display Editing Commands.....	26-19
Using TTYIN for Lisp Input.....	26-22
Useful Macros	26-23
Programming with TTYIN	26-23
Using TTYIN as a General Editor.....	26-25
? = Handler	26-26
Read Macros	26-27
Assorted Flags	26-28
Special Responses	26-29
Display Types.....	26-30
Prettyprint.....	26-31
Comment Feature	26-33
Comment Pointers.....	26-34
Converting Comments to Lowercase	26-35
Special Prettyprint Controls.....	26-36

26. Graphics Output Operations 27-1

Primitive Graphics Concepts	27-1
Positions	27-1
Regions	27-1
Bitmaps.....	27-2
Textures.....	27-5
Opening Image Streams.....	27-6
Accessing Image Stream Fields	27-8
Current Position of an Image Stream	27-10
Moving Bits Between Bitmaps with BITBLT	27-11
Drawing Lines.....	27-13
Drawing Curves.....	27-14
Miscellaneous Drawing and Printing Operations	27-15
Drawing and Shading Grids	27-17
Display Streams	27-18
Fonts	27-19
Font Files and Font Directories.....	27-24
Font Profiles.....	27-24
Image Objects	27-27
IMAGEFNS Methods	27-28

Registering Image Objects.....	27-30
Reading and Writing Image Objects on Files.....	27-31
Copying Image Objects Between Windows	27-31
Implementation of Image Streams.....	27-32
27. Windows and Menus	28-1
Using the Window System.....	28-1
Changing the Window System.....	28-6
Interactive Display Functions.....	28-7
Windows.....	28-9
Window Properties	28-10
Creating Windows	28-10
Opening and Closing Windows.....	28-11
Redisplaying Windows	28-12
Reshaping Windows.....	28-13
Moving Windows.....	28-14
Exposing and Burying Windows	28-16
Shrinking Windows into Icons.....	28-16
Coordinate Systems, Extents, and Scrolling	28-18
Mouse Activity in Windows.....	28-21
Terminal I/O and Page Holding.....	28-22
TTY Process and the Caret.....	28-23
Miscellaneous Window Functions.....	28-24
Miscellaneous Window Properties	28-25
Example: A Scrollable Window	28-26
Menus.....	28-28
Menu Fields.....	28-29
Miscellaneous Menu Functions.....	28-32
Examples of Menu Use.....	28-32
Attached Windows	28-34
Attaching Menus to Windows.....	28-37
Attached Prompt Windows	28-38
Window Operations and Attached Windows.....	28-39
Window Properties of Attached Windows	28-41
28. Hardcopy Facilities	29-1
Hardcopy Functions	29-1
Low-Level Hardcopy Variables	29-4
29. Terminal Input/Output	30-1
Interrupt Characters.....	30-1
Terminal Tables	30-4
Terminal Syntax Classes.....	30-4
Terminal Control Functions.....	30-5
Line-Buffering	30-7

Dribble Files.....	30-10
Cursor and Mouse	30-10
Changing the Cursor Image.....	30-11
Flashing Bars on the Cursor.....	30-13
Cursor Position	30-13
Mouse Button Testing	30-14
Low-Level Mouse Functions.....	30-15
Keyboard Interpretation	30-15
Display Screen.....	30-18
Miscellaneous Terminal I/O	30-19

30. Ethernet 31-1

Ethernet Protocols.....	31-1
Protocol Layering	31-1
Level Zero Protocols.....	31-2
Level One Protocols.....	31-2
Higher Level Protocols	31-3
Connecting Networks: Routers and Gateways	31-3
Addressing Conflicts with Level Zero Mediums.....	31-3
References	31-4
Higher-Level PUP Protocol Functions	31-4
Higher-Level NS Protocol Functions	31-5
Name and Address Conventions	31-5
Clearinghouse Functions	31-7
NS Printing	31-9
SPP Stream Interface	31-9
Courier Remote Procedure Call Protocol.....	31-11
Defining Courier Programs	31-11
Courier Type Definitions	31-12
Pre-defined Types	31-13
Constructed Types	31-13
User Extensions to the Type Language.....	31-15
Performing Courier Transactions	31-16
Expedited Procedure Call	31-17
Expanding Ring Broadcast.....	31-18
Using Bulk Data Transfer.....	31-18
Courier Subfunctions for Data Transfer.....	31-19
Level One Ether Packet Format	31-20
PUP Level One Functions.....	31-21
Creating and Managing Pups	31-21
Sockets	31-22
Sending and Receiving Pups.....	31-23
Pup Routing Information	31-23
Miscellaneous PUP Utilities	31-24
PUP Debugging Aids	31-24

NS Level One Functions.....	31-28
Creating and Managing XIPs.....	31-28
NS Sockets	31-28
Sending and Receiving XIPs	31-29
NS Debugging Aids	31-29
Support for Other Level One Protocols.....	31-29
The SYSQUEUE Mechanism	31-31
Glossary	GLOSSARY-1
Index	INDEX-1

[This page intentionally left blank]

4. STRINGS

A string represents a sequence of characters. Interlisp strings are a subtype of Common Lisp strings. Medley provides functions for creating strings, concatenating strings, and creating sub-strings of a string; all accepting or producing Common Lisp-acceptable strings.

A string is typed as a double quote ("), followed by a sequence of any characters except double quote and %, terminated by a double quote. To include % or " in a string, type % in front of them:

```
"A string"
"A string with %" in it, and a %%"
""           ; an empty string
```

Strings are printed by PRINT and PRIN2 with initial and final double quotes, and %s inserted where necessary for it to read back in properly. Strings are printed by PRIN1 without the double quotes and extra %s. The null string is printed by PRINT and PRIN2 as "". (PRIN1 "") doesn't print anything.

Internally, a string is stored in two parts: a "string header" and the sequence of characters. Several string headers may refer to the the same character sequence, so a substring can be made by creating a new string header, without copying any characters. Functions that refer to "strings" actually manipulate string headers. Some functions take an "old string" argument, and re-use the string pointer.

(STRINGP X) [Function]

Returns X if X is a string, NIL otherwise.

(STREQUAL X Y) [Function]

Returns T if X and Y are both strings and they contain the same sequence of characters, otherwise NIL. EQUAL uses STREQUAL. Note that strings may be STREQUAL without being EQ. For instance,

```
(STREQUAL "ABC" "ABC") => T
(EQ "ABC" "ABC") => NIL
```

STREQUAL returns T if X and Y are the same string pointer, or two different string pointers which point to the same character sequence, or two string pointers which point to different character sequences which contain the same characters. Only in the first case would X and Y be EQ.

(STRING-EQUAL X Y) [Function]

Returns T if X and Y are either strings or symbols, and they contain the same sequence of characters, ignoring case. For instance,

```
(STRING-EQUAL "FOO" "Foo") => T
(STRING-EQUAL "FOO" 'Foo) => T
```

This is useful for comparing things that might want to be considered "equal" even though they're not both symbols in a consistent case, such as file names and user names.

INTERLISP-D REFERENCE MANUAL

(**STRING.EQUAL** *X Y*) [Function]

Returns T if the print names of *X* and *Y* contain the same sequence of characters, ignoring case. For instance,

```
(STRING-EQUAL "320" 320) => T
(STRING-EQUAL "FOO" 'Foo) => T
```

This is like `STRING-EQUAL`, but handles numbers, etc., where `STRING-EQUAL` doesn't.

(**ALLOCSTRING** *N INITCHAR OLD FATFLG*) [Function]

Creates a string of length *N* characters of *INITCHAR* (which can be either a character code or something coercible to a character). If *INITCHAR* is `NIL`, it defaults to character code 0. If *OLD* is supplied, it must be a string pointer, which is modified and returned.

If *FATFLG* is non-`NIL`, the string is allocated using full 16-bit NS characters (see Chapter 2) instead of 8-bit characters. This can speed up some string operations if NS characters are later inserted into the string. This has no other effect on the operation of the string functions.

(**MKSTRING** *X FLG RDTBL*) [Function]

If *X* is a string, returns *X*. Otherwise, creates and returns a string containing the print name of *X*. Examples:

```
(MKSTRING "ABC") => "ABC"
(MKSTRING '(A B C)) => "(A B C)"
(MKSTRING NIL) => "NIL"
```

Note that the last example returns the string "NIL", not the symbol `NIL`.

If *FLG* is T, then the `PRIN2`-name of *X* is used, computed with respect to the readable *RDTBL*. For example,

```
(MKSTRING "ABC" T) => "%\"ABC%\""
```

(**NCHARS** *X FLG RDTBL*) [Function]

Returns the number of characters in the print name of *X*. If *FLG*=T, the `PRIN2`-name is used. For example,

```
(NCHARS 'ABC) => 3
(NCHARS "ABC" T) => 5
```

Note: `NCHARS` works most efficiently on symbols and strings, but can be given any object.

(**SUBSTRING** *X N M OLDPTR*) [Function]

Returns the substring of *X* consisting of the *N*th through *M*th characters of *X*. If *M* is `NIL`, the substring contains the *N*th character thru the end of *X*. *N* and *M* can be negative numbers, which are interpreted as counts back from the end of the string, as with `NTHCHAR` (Chapter 2). `SUBSTRING` returns `NIL` if the substring is not well defined, (e.g., *N* or *M* specify character positions outside of *X*, or *N* corresponds to a character in *X* to the right of the character indicated by *M*). Examples:

STRINGS

```
(SUBSTRING "ABCDEFGH" 4 6) => "DEF"
(SUBSTRING "ABCDEFGH" 3 3) => "C"
(SUBSTRING "ABCDEFGH" 3 NIL) => "CDEFGH"
(SUBSTRING "ABCDEFGH" 4 -2) => "DEF"
(SUBSTRING "ABCDEFGH" 6 4) => NIL
(SUBSTRING "ABCDEFGH" 4 9) => NIL
```

If *X* is not a string, it is converted to one. For example,

```
(SUBSTRING ' (A B C) 4 6) => "B C"
```

SUBSTRING does not actually copy any characters, but simply creates a new string pointer to the characters in *X*. If *OLDPTR* is a string pointer, it is modified and returned.

(**GNC** *X*)

[Function]

“Get Next Character.” Returns the next character of the string *X* (as a symbol); also removes the character from the string, by changing the string pointer. Returns NIL if *X* is the null string. If *X* isn’t a string, a string is made. Used for sequential access to characters of a string. Example:

```
← (SETQ FOO "ABCDEFGH")
"ABCDEFGH"
← (GNC FOO)
A
← (GNC FOO)
B
← FOO
"CDEFGH"
```

Note that if *A* is a substring of *B*, (GNC *A*) does not remove the character from *B*.

(**GLC** *X*)

[Function]

“Get Last Character.” Returns the last character of the string *X* (as a symbol); also removes the character from the string. Similar to GNC. Example:

```
← (SETQ FOO "ABCDEFGH")
"ABCDEFGH"
← (GLC FOO)
G
← (GLC FOO)
F
← FOO
"ABCDE"
```

(**CONCAT** *X X ... X*)

[NoSpread Function]

Returns a new string which is the concatenation of (copies of) its arguments. Any arguments which are not strings are transformed to strings. Examples:

```
(CONCAT "ABC" 'DEF "GHI") => "ABCDEFGHI"
(CONCAT ' (A B C) "ABC") => " (A B C)ABC"
(CONCAT) returns the null string, ""
```

INTERLISP-D REFERENCE MANUAL

(CONCATLIST *L*) [Function]

L is a list of strings and/or other objects. The objects are transformed to strings if they aren't strings. Returns a new string which is the concatenation of the strings. Example:

```
(CONCATLIST ' (A B (C D) "EF")) => "AB(C D)EF"
```

(RPLSTRING *X N Y*) [Function]

Replaces the characters of string *X* beginning at character position *N* with string *Y*. *X* and *Y* are converted to strings if they aren't already. *N* may be positive or negative, as with SUBSTRING. Characters are smashed into (converted) *X*. Returns the string *X*. Examples:

```
(RPLSTRING "ABCDEF" -3 "END") => "ABCEND"  
(RPLSTRING "ABCDEFGHIJK" 4 ' (A B C) ) => "ABC(A B C)K"
```

Generates an error if there is not enough room in *X* for *Y*, i.e., the new string would be longer than the original. If *Y* was not a string, *X* will already have been modified since RPLSTRING does not know whether *Y* will "fit" without actually attempting the transfer.

Warning: In some implementations of Interlisp, if *X* is a substring of *Z*, *Z* will also be modified by the action of RPLSTRING or RPLCHARCODE. However, this is not guaranteed to be true in all cases, so programmers should not rely on RPLSTRING or RPLCHARCODE altering the characters of any string other than the one directly passed as argument to those functions.

(RPLCHARCODE *X N CHAR*) [Function]

Replaces the *N*th character of the string *X* with the character code *CHAR*. *N* may be positive or negative. Returns the new *X*. Similar to RPLSTRING. Example:

```
(RPLCHARCODE "ABCDE" 3 (CHARCODE F)) => "ABFDE"
```

(STRPOS *PAT STRING START SKIP ANCHOR TAIL CASEARRAY BACKWARDSFLG*) [Function]

STRPOS is a function for searching one string looking for another. *PAT* and *STRING* are both strings (or else they are converted automatically). STRPOS searches *STRING* beginning at character number *START*, (or 1 if *START* is NIL) and looks for a sequence of characters equal to *PAT*. If a match is found, the character position of the first matching character in *STRING* is returned, otherwise NIL. Examples:

```
(STRPOS "ABC" "XYZABCDEF") => 4  
(STRPOS "ABC" "XYZABCDEF" 5) => NIL  
(STRPOS "ABC" "XYZABCDEFABC" 5) => 10
```

SKIP can be used to specify a character in *PAT* that matches any character in *STRING*. Examples:

```
(STRPOS "A&C&" "XYZABCDEF" NIL '&) => 4  
(STRPOS "DEF&" "XYZABCDEF" NIL '&) => NIL
```

If *ANCHOR* is T, STRPOS compares *PAT* with the characters beginning at position *START* (or 1 if *START* is NIL). If that comparison fails, STRPOS returns NIL without searching any further down *STRING*. Thus it can be used to compare one string with some *portion* of another string. Examples:

```
(STRPOS "ABC" "XYZABCDEF" NIL NIL T) => NIL
```

STRINGS

```
(STRPOS "ABC" "XYZABCDEF" 4 NIL T) => 4
```

If *TAIL* is T, the value returned by STRPOS if successful is not the starting position of the sequence of characters corresponding to *PAT*, but the position of the first character after that, i.e., the starting position plus (NCHARS *PAT*). Examples:

```
(STRPOS "ABC" "XYZABCDEFABC" NIL NIL NIL T) => 7
(STRPOS "A" "A" NIL NIL NIL T) => 2
```

If *TAIL* = NIL, STRPOS returns NIL, or a character position within *STRING* which can be passed to SUBSTRING. In particular, (STRPOS "" "") => NIL. However, if *TAIL* = T, STRPOS may return a character position outside of *STRING*. For instance, note that the second example above returns 2, even though "A" has only one character.

If *CASEARRAY* is non-NIL, this should be a casearray like that given to FILEPOS (Chapter 25). The casearray is used to map the string characters before comparing them to the search string.

If *BACKWARDSFLG* is non-NIL, the search is done backwards from the end of the string.

(STRPOSL *A STRING START NEG BACKWARDSFLG*) [Function]

STRING is a string (or is converted automatically to a string), *A* is a list of characters or character codes. STRPOSL searches *STRING* beginning at character number *START* (or 1 if *START* = NIL) for one of the characters in *A*. If one is found, STRPOSL returns as its value the corresponding character position, otherwise NIL. Example:

```
(STRPOSL '(A B C) "XYZBCD") => 4
```

If *NEG* = T, STRPOSL searches for a character *not* on *A*. Example:

```
(STRPOSL '(A B C) "ABCDEF" NIL T) => 4
```

If any element of *A* is a number, it is assumed to be a character code. Otherwise, it is converted to a character code via CHCON1. Therefore, it is more efficient to call STRPOSL with *A* a list of character *codes*.

If *A* is a bit table, it is used to specify the characters (see MAKEBITTABLE below)

If *BACKWARDSFLG* is non-NIL, the search is done backwards from the end of the string.

STRPOSL uses a "bit table" data structure to search efficiently. If *A* is not a bit table, it is converted to a bit table using MAKEBITTABLE. If STRPOSL is to be called frequently with the same list of characters, a considerable savings can be achieved by converting the list to a bit table *once*, and then passing the bit table to STRPOSL as its first argument.

(MAKEBITTABLE *L NEG A*) [Function]

Returns a bit table suitable for use by STRPOSL. *L* is a list of characters or character codes, *NEG* is the same as described for STRPOSL. If *A* is a bit table, MAKEBITTABLE modifies and returns it. Otherwise, it will create a new bit table.

INTERLISP-D REFERENCE MANUAL

Note: If *NEG* = T, STRPOS_L must call MAKEBITTABLE whether *A* is a list or a bit table. To obtain bit table efficiency with *NEG*=T, MAKEBITTABLE should be called with *NEG*=T, and the resulting “inverted” bit table should be given to STRPOS_L with *NEG*=NIL.

STRINGS

[This page intentionally left blank]

5. ARRAYS

An Interlisp array is a one-dimensional vector of objects. Arrays are generally created by the function `ARRAY`. By contrast, Common Lisp arrays can be multi-dimensional.

Note: Interlisp arrays and Common Lisp arrays are *not* the same types. Interlisp functions only accept Interlisp arrays and vice versa. There are no functions to convert between the two types.

(`ARRAY` *SIZE TYPE INIT ORIG* *—*) [Function]

Creates and returns a new array that holds *SIZE* objects of type *TYPE*. If *TYPE* is `NIL`, the array can contain any arbitrary Lisp datum. In general, *TYPE* may be any of the various field specifications that are legal in `DATATYPE` declarations (see Chapter 8): `POINTER`, `FIXP`, `FLOATP`, `(BITS N)`, etc. Medley will, if necessary, choose an “enclosing” type if the given one is not supported; for example, an array of `(BITS 3)` may be represented by an array of `(BITS 8)`.

INIT is the initial value for each element of the new array. If not specified, the array elements will be initialized with 0 (for number arrays) or `NIL` (all other types).

Arrays can have either 0-origin or 1-origin indexing, as specified by the *ORIG* argument; if *ORIG* is not specified, the default is 1.

Arrays of type `FLOATP` are stored unboxed. This increases the space and time efficiency of `FLOATP` arrays. If you want to use boxed floating point numbers, use an array of type `POINTER` instead of `FLOATP`.

(`ARRAYP` *X*) [Function]

Returns *X* if *X* is an array, `NIL` otherwise.

(`ELT` *ARRAY N*) [Function]

Returns the *N*th element of the array *ARRAY*.

Causes the error, `Arg not array`, if *ARRAY* is not an array. Causes the error, `Illegal Arg`, if *N* is out of bounds.

(`SETA` *ARRAY N VAL*) [Function]

Sets the *N*th element of *ARRAY* to *VAL*, and returns *VAL*.

Causes the error, `Arg not array`, if *ARRAY* is not an array. the error, `Illegal Arg`, if *N* is out of bounds. Can cause the error, `Non-numeric arg`, if *ARRAY* is an array whose `ARRAYTYP` is `FIXP` or `FLOATP` and *VAL* is non-numeric.

(`ARRAYTYP` *ARRAY*) [Function]

Returns the type of the elements in *ARRAY*, a value corresponding to the second argument to *ARRAY*.

INTERLISP-D REFERENCE MANUAL

If `ARRAY` coerced the array type as described above, `ARRAYTYP` returns the *new* type. For example, `(ARRAYTYP (ARRAY 10 ' (BITS 3)))` returns `BYTE`.

(ARRAYSIZE *ARRAY*) [Function]

Returns the size of *ARRAY*. Generates the error, `Arg not array`, if *ARRAY* is not an array.

(ARRAYORIG *ARRAY*) [Function]

Returns the origin of *ARRAY*, which may be 0 or 1. Generates an error, `Arg not array`, if *ARRAY* is not an array.

(COPYARRAY *ARRAY*) [Function]

Returns a new array of the same size and type as *ARRAY*, and with the same contents as *ARRAY*. Generates an error, `Arg not array`, if *ARRAY* is not an array.

ARRAYS

[This page intentionally left blank]

6. HASHARRAYS

Hash arrays let you associate arbitrary Lisp objects (“hash keys”) with other objects (“hash values”), so you can get from key to value quickly. There are functions for creating hash arrays, putting a hash key/value pair in a hash array, and quickly retrieving the hash value associated with a given hash key.

By default, the hash array functions use EQ for comparing hash keys. This means that if non-symbols are used as hash keys, the exact same object (not a copy) must be used to retrieve the hash value. However, you can specify the function used to compare hash keys and to “hash” a hash key to a number. You can, for example, create hash arrays where EQUAL but non-EQ strings will hash to the same value. Specifying alternative hashing algorithms is described below.

In the description of the functions below, the argument *HARRAY* should be a hasharray created by *HASHARRAY*. For convenience in interactive program development, it may also be NIL, in which case a hash array (*SYSHASHARRAY*) provided by the system is used; you must watch out for confusions if this form is used to associate more than one kind of value with the same key.

Note: For backwards compatibility, the hash array functions will accept a list whose CAR is a hash array, and whose CDR is the “overflow method” for the hash array (see below). However, hash array functions are guaranteed to perform with maximum efficiency only if a direct value of *HASHARRAY* is given.

Note: Interlisp hash arrays and Common Lisp hash tables are the same data type, so functions from both may be intermixed. The only difference between the functions may be argument order, as in *MAPHASH* and *CL:MAPHASH* (see below).

(**HASHARRAY** *MINKEYS* *OVERFLOW* *HASHBITSFN* *EQUIVFN* *RECLAIMABLE* *REHASH-*
THRESHOLD) [Function]

Creates a hash array with space for at least *MINKEYS* hash keys, with overflow method *OVERFLOW*. See discussion of overflow behavior below.

If *HASHBITSFN* and *EQUIVFN* are non-NIL, they specify the hashing function and comparison function used to interpret hash keys. This is described in the section on user-specified hashing functions below. If *HASHBITSFN* and *EQUIVFN* are NIL, the default is to hash EQ hash keys to the same value.

If *RECLAIMABLE* is *T* the entries in the hash table will be removed if the key has a reference count of one and the table is about to be rehashed. This allows the system, in some cases, to reuse keys instead of expanding the table.

Note: *CL:MAKE-HASH-TABLE* does not allow you to specify your own hashing functions but does provide three built-in types specified by *Common Lisp, the Language*.

(**HARRAY** *MINKEYS*) [Function]

Provided for backward compatibility, this is equivalent to (*HASHARRAY* *MINKEYS* 'ERROR) , i.e. if the resulting hasarray gets full, an error occurs.

INTERLISP-D REFERENCE MANUAL

(**HARRAYP** *X*) [Function]

Returns *X* if it is a hash array; otherwise **NIL**.

HARRAYP returns **NIL** if *X* is a list whose **CAR** is an **HARRAYP**, even though this is accepted by the hash array functions (see below).

(**PUTHASH** *KEY VAL HARRAY*) [Function]

Associates the hash value *VAL* with the hash key *KEY* in *HARRAY*. Replaces the previous hash value, if any. If *VAL* is **NIL**, any old association is removed (hence a hash value of **NIL** is not allowed). If *HARRAY* is full when **PUTHASH** is called with a key not already in the hash array, the function **HASHOVERFLOW** is called, and the **PUTHASH** is applied to the value returned (see below). Returns *VAL*.

(**GETHASH** *KEY HARRAY*) [Function]

Returns the hash value associated with the hash key *KEY* in *HARRAY*. Returns **NIL**, if *KEY* is not found.

(**CLRHASH** *HARRAY*) [Function]

Clears all hash keys / values from *HARRAY*. Returns *HARRAY*.

(**HARRAYPROP** *HARRAY PROP NEWVALUE*) [NoSpread Function]

Returns the property *PROP* of *HARRAY*; *PROP* can have the system-defined values **SIZE** (the maximum occupancy of *HARRAY*), **NUMKEYS** (number of occupied slots), **OVERFLOW** (overflow method), **HASHBITSFN** (hashing function) and **EQUIVFN** (comparison function). Except for **SIZE** and **NUMKEYS**, a new value may be specified as *NEWVALUE*.

By using other values for *PROP*, the user may also set and get arbitrary property values, to associate additional information with a hash array.

The **HASHBITSFN** or **EQUIVFN** properties can only be changed if the hash array is empty.

(**HARRAYSIZE** *HARRAY*) [Function]

Returns the number of slots in *HARRAY*. It's equivalent to (**HARRAYPROP** *HARRAY* ' **SIZE**).

(**REHASH** *OLDHARRAY NEWHARRAY*) [Function]

Hashes all hash keys and values in *OLDHARRAY* into *NEWHARRAY*. The two hash arrays do not have to be (and usually aren't) the same size. Returns *NEWHARRAY*.

(**MAPHASH** *HARRAY MAPHFN*) [Function]

MAPHFN is a function of two arguments. For each hash key in *HARRAY*, *MAPHFN* will be applied to the hash value, and the hash key. For example:

```
[MAPHASH A
 (FUNCTION (LAMBDA (VAL KEY)
  (if (LISTP KEY) then (PRINT VAL))]
```

will print the hash value for all hash keys that are lists. **MAPHASH** returns *HARRAY*.

HASHARRAYS

Note: the argument order for `CL:MAPHASH` is `MAPHEN HARRAY`.

(**DMPHASH** *HARRAY HARRAY ... HARRAY*) [NLambda NoSpread Function]

Prints on the primary output file LOADable forms which will restore the hash-arrays contained as the values of the atoms *HARRAY*, *HARRAY*, ... *HARRAY*. Example: (`DMPHASH SYSHASHARRAY`) will dump the system hash-array.

All EQ identities except symbols and small integers are lost by dumping and loading because `READ` will create new structure for each item. Thus if two lists contain an EQ substructure, when they are dumped and loaded back in, the corresponding substructures while `EQUAL` are no longer EQ. The `HORRIBLEVARS` file package command (Chapter 17) provides a way of dumping hash tables such that these identities are preserved.

Hash Overflow

When a hash array becomes full, trying to add another hash key will cause the function `HASHOVERFLOW` to be called. This either enlarges the hash array, or causes the error `Hash table full`. How hash overflow is handled is determined by the value of the `OVERFLOW` property of the hash array (which can be accessed by `HARRAYPROP`). The possibilities for the overflow method are:

the symbol <code>ERROR</code>	The error <code>Hash array full</code> is generated when the hash array overflows. This is the default overflow behavior for hash arrays returned by <code>HARRAY</code> .
<code>NIL</code>	The array is automatically enlarged by at least a factor 1.5 every time it overflows. This is the default overflow behavior for hash arrays returned by <code>HASHARRAY</code> .
a positive integer <i>N</i>	The array is enlarged to include at least <i>N</i> more slots than it currently has.
a floating point number <i>F</i>	The array is changed to include <i>F</i> times the number of current slots.
a function or lambda expression <i>FN</i>	Upon hash overflow, <i>FN</i> is called with the hash array as its argument. If <i>FN</i> returns a number, that will become the size of the array. Otherwise, the new size defaults to 1.5 times its previous size. <i>FN</i> could be used to print a message, or perform some monitor function.

Note: For backwards compatibility, the hash array functions accept a list whose `CAR` is the hash array, and whose `CDR` is the overflow method. In this case, the overflow method specified in the list overrides the overflow method set in the hash array. Hash array functions perform with maximum efficiency only if a direct value of `HASHARRAY` is given.

Specifying Your Own Hashing Functions

In general terms, when a key is looked up in a hash array, it is converted to an integer, which is used to index into a linear array. If the key is not the same as the one found at that index, other indices are

INTERLISP-D REFERENCE MANUAL

tried until the desired key is found. The value stored with that key is then returned (from `GETHASH`) or replaced (from `PUTHASH`).

To customize hash arrays, you'll need to supply the "hashing function" used to convert a key to an integer and the comparison function used to compare the key found in the array with the key being looked up. For hash arrays to work correctly, any two objects which are equal according to the comparison function must "hash" to equal integers.

By default, Medley uses a hashing function that computes an integer from the internal address of a key, and use `EQ` for comparing keys. This means that if non-atoms are used as hash keys, *the exact same object* (not a copy) must be used to retrieve the hash value.

There are some applications for which the `EQ` constraint is too restrictive. For example, it may be useful to use strings as hash keys, without the restriction that `EQUAL` but not `EQ` strings are considered to be different hash keys.

The user can override this default behavior for any hash array by specifying the functions used to compare keys and to "hash" a key to a number. This can be done by giving the `HASHBITSFN` and `EQUIVFN` arguments to `HASHARRAY` (see above).

The `EQUIVFN` argument is a function of two arguments that returns non-NIL when its arguments are considered equal. The `HASHBITSFN` argument is a function of one argument that produces a positive small integer (in the range $[0..2^{16} - 1]$) with the property that objects that are considered equal by the `EQUIVFN` produce the same hash bits.

For an existing hash array, the function `HARRAYPROP` (see above) can be used to examine the hashing and equivalence functions as the `HASHBITSFN` and `EQUIVFN` hash array properties. These properties are read-only for non-empty hash arrays, as it makes no sense to change the equivalence relationship once some keys have been hashed.

The following function is useful for creating hash arrays that take strings as hash keys:

`(STRINGHASHBITS STRING)` [Function]

Hashes the string `STRING` into an integer that can be used as a `HASHBITSFN` for a hash array. Strings which are `STREQUAL` hash to the same integer.

Example:

```
(HASHARRAY MINKEYS OVERFLOW 'STRINGHASHBITS 'STREQUAL)
```

creates a hash array where you can use strings as hash keys.

HASHARRAYS

[This page intentionally left blank]

7. NUMBERS AND ARITHMETIC FUNCTIONS

There are four different types of numbers in Interlisp: small integers, large integers, bignums (arbitrary-size integers), and floating-point numbers. Small integers are in the range -65536 to 65535. Large integers and floating-point numbers are 32-bit quantities that are stored by “boxing” the number (see below). Bignums are “boxed” as a series of words.

Large integers and floating-point numbers can be any full word quantity. To distinguish among the various kinds of numbers, and other Interlisp pointers, these numbers are “boxed”. When a large integer or floating-point number is created (by an arithmetic operation or by READ), Interlisp gets a new word from “number storage” and puts the number into that word. Interlisp then passes around the pointer to that word, i.e., the “boxed number”, rather than the actual quantity itself. When a numeric function needs the actual numeric quantity, it performs the extra level of addressing to obtain the “value” of the number. This latter process is called “unboxing”. Unboxing does not use any storage, but each boxing operation uses one new word of number storage. If a computation creates many large integers or floating-point numbers, i.e., does lots of boxes, it may cause a garbage collection of large integer space, or of floating-point number space.

The following functions can be used to distinguish the different types of numbers:

(**SMALLP** *X*) [Function]

Returns *X*, if *X* is a small integer; NIL otherwise. Does not generate an error if *X* is not a number.

(**FIXP** *X*) [Function]

Returns *X*, if *X* is an integer; NIL otherwise. Note that FIXP is true for small integers, large integers, and bignums. Does not generate an error if *X* is not a number.

(**FLOATP** *X*) [Function]

Returns *X* if *X* is a floating-point number; NIL otherwise. Does not give an error if *X* is not a number.

(**NUMBERP** *X*) [Function]

Returns *X*, if *X* is a number of any type; NIL otherwise. Does not generate an error if *X* is not a number.

Note: In previous releases, NUMBERP was true only if (FLOATP *X*) or (FIXP *X*) were true. With the addition of Common Lisp ratios and complex numbers, NUMBERP now returns T for *all* number types. Code relying on the “old” behavior should be modified.

Each small integer has a unique representation, so EQ may be used to check equality. EQ should not be used for large integers, bignums, or floating-point numbers, EQP, IEQP, or EQUAL must be used instead.

INTERLISP-D REFERENCE MANUAL

(**EQP** *X Y*) [Function]

Returns T, if *X* and *Y* are equal numbers; NIL otherwise. EQ may be used if *X* and *Y* are known to be small integers. EQP does not convert *X* and *Y* to integers, e.g., (EQP 2000 2000.3) => NIL, but it can be used to compare an integer and a floating-point number, e.g., (EQP 2000 2000.0) => T. EQP does not generate an error if *X* or *Y* are not numbers.

EQP can also be used to compare stack pointers (see Chapter 11) and compiled code objects (see Chapter 10).

The action taken on division by zero and floating-point overflow is determined with the following function:

(**OVERFLOW** *FLG*) [Function]

Sets a flag that determines the system response to arithmetic overflow (for floating-point arithmetic) and division by zero; returns the previous setting.

For integer arithmetic: If *FLG* = T, an error occurs on division by zero. If *FLG* = NIL or 0, integer division by zero returns zero. Integer overflow cannot occur, because small integers are converted to bignums (see the beginning of this chapter).

For floating-point arithmetic: If *FLG* = T, an error occurs on floating overflow or floating division by zero. If *FLG* = NIL or 0, the largest (or smallest) floating-point number is returned as the result of the overflowed computation or floating division by zero.

The default value for OVERFLOW is T, meaning an error is generated on division by zero or floating overflow.

Generic Arithmetic

The functions in this section are “generic” arithmetic functions. If any of the arguments are floating-point numbers (see the Floating-Point Arithmetic section below), they act exactly like floating-point functions, floating all arguments and returning a floating-point number as their value. Otherwise, they act like the integer functions (see the Integer Arithmetic section below). If given a non-numeric argument, they generate an error, Non-numeric arg. The results of division by zero and floating-point overflow is determined by the function OVERFLOW (see the section above).

(**PLUS** *X X ... X*) [NoSpread Function]

$X + X + \dots + X$.

(**MINUS** *X*) [Function]

$- X$

(**DIFFERENCE** *X Y*) [Function]

$X - Y$

(**TIMES** *X X ... X*) [NoSpread Function]

$X * X * \dots * X$

NUMBERS AND ARITHMETIC FUNCTIONS

(**QUOTIENT** *X Y*) [Function]

If *X* and *Y* are both integers, returns the integer division of *X* and *Y*. Otherwise, converts both *X* and *Y* to floating-point numbers, and does a floating-point division.

(**REMAINDER** *X Y*) [Function]

If *X* and *Y* are both integers, returns (**IREMAINDER** *X Y*), otherwise (**FREMAINDER** *X Y*).

(**GREATERP** *X Y*) [Function]

T, if *X* > *Y*, NIL otherwise.

(**LESSP** *X Y*) [Function]

T if *X* < *Y*, NIL otherwise.

(**GEQ** *X Y*) [Function]

T, if *X* >= *Y*, NIL otherwise.

(**LEQ** *X Y*) [Function]

T, if *X* <= *Y*, NIL otherwise.

(**ZEROP** *X*) [Function]

The same as (**EQP** *X 0*).

(**MINUSP** *X*) [Function]

T, if *X* is negative; NIL otherwise. Works for both integers and floating-point numbers.

(**MIN** *X X . . . X*) [NoSpread Function]

Returns the minimum of *X*, *X*, . . . , *X*. (**MIN**) returns the value of **MAX**. **INTEGER** (see the Integer Arithmetic section below).

(**MAX** *X X . . . X*) [NoSpread Function]

Returns the maximum of *X*, *X*, . . . , *X*. (**MAX**) returns the value of **MIN**. **INTEGER** (see the Integer Arithmetic section below).

(**ABS** *X*) [Function]

X if *X* > 0, otherwise -*X*. **ABS** uses **GREATERP** and **MINUS** (not **IGREATERP** and **IMINUS**).

Integer Arithmetic

The input syntax for an integer is an optional sign (+ or -) followed by a sequence of decimal digits, and terminated by a delimiting character. Integers entered with this syntax are interpreted as decimal integers. Integers in other radices can be entered as follows:

123Q

#o123 If an integer is followed by the letter Q, or preceded by a pound sign and the letter “o”, the digits are interpreted as an octal (base 8) integer.

INTERLISP-D REFERENCE MANUAL

#b10101 If an integer is preceeded by a pound sign and the letter “b”, the digits are interpreted as a binary (base 2) integer.

#x1A90 If an integer is preceeded by a pound sign and the letter “x”, the digits are interpreted as a hexadecimal (base 16) integer.

#5r1243 If an integer is preceeded by a pound sign, a positive decimal integer **BASE**, and the letter “r”, the digits are interpreted as an integer in the base **BASE**. For example, **#8r123** = 123Q, and **#16r12A3** = **#x12A3**. When typing a number in a radix above ten, the uppercase letters A through Z can be used as the digits after 9 (but there is no digit above Z, so it is not possible to type all base-99 digits).

Medley keeps no record of how you typed a number, so 77Q and 63 both correspond to the same integer, and are indistinguishable internally. The function **RADIX** (see Chapter 25), sets the radix used to print integers.

PACK and **MKATOM** create numbers when given a sequence of characters observing the above syntax, e.g. (**PACK** ' (1 2 Q)) => 10. Integers are also created as a result of arithmetic operations.

The range of integers of various types is implementation-dependent. This information is accessible to you through the following variables:

MIN.SMALLP	[Variable]
MAX.SMALLP	[Variable]

The smallest/largest possible small integer.

MIN.FIXP	[Variable]
MAX.FIXP	[Variable]

The smallest/largest possible large integer.

MIN.INTEGER	[Variable]
MAX.INTEGER	[Variable]

The value of **MAX.INTEGER** and **MIN.INTEGER** are two special system datatypes. For some algorithms, it is useful to have an integer that is larger than any other integer. Therefore, the values of **MAX.INTEGER** and **MIN.INTEGER** are two special data types; the value of **MAX.INTEGER** is **GREATERP** than any other integer, and the value of **MIN.INTEGER** is **LESSP** than any other integer. Trying to do arithmetic using these special bignums, other than comparison, will cause an error.

All of the functions described below work on integers. Unless specified otherwise, if given a floating-point number, they first convert the number to an integer by truncating the fractional bits, e.g., (**IPLUS** 2.3 3.8) = 5; if given a non-numeric argument, they generate an error, Non-numeric arg.

(IPLUS X X ... X)	[NoSpread Function]

Returns the sum $X + X + \dots + X$ (**IPLUS**) = 0.

(IMINUS X)	[Function]

-X

NUMBERS AND ARITHMETIC FUNCTIONS

(IDIFFERENCE X Y) [Function]

$X - Y$

(ADD1 X) [Function]

$X + 1$

(SUB1 X) [Function]

$X - 1$

(ITIMES X X ... X) [NoSpread Function]

Returns the product $X * X * \dots * X$. (ITIMES) = 1.

(IQUOTIENT X Y) [Function]

X / Y truncated. Examples:

(IQUOTIENT 3 2) => 1

(IQUOTIENT -3 2) => -1

If Y is zero, the result is determined by the function OVERFLOW.

(IREMAINDER X Y) [Function]

Returns the remainder when X is divided by Y. Example:

(IREMAINDER 5 2) => 1

(IMOD X N) [Function]

Computes the integer modulus of $X \bmod N$; this differs from IREMAINDER in that the result is always a non-negative integer in the range $[0, N)$.

(IGREATERP X Y) [Function]

T, if $X > Y$; NIL otherwise.

(ILESSP X Y) [Function]

T, if $X < Y$; NIL otherwise.

(IGEQL X Y) [Function]

T, if $X \geq Y$; NIL otherwise.

(ILEQL X Y) [Function]

T, if $X \leq Y$; NIL otherwise.

(IMIN X X ... X) [NoSpread Function]

Returns the minimum of $X, X, \dots, X$. (IMIN) returns the largest possible large integer, the value of MAX.INTEGER.

INTERLISP-D REFERENCE MANUAL

(**IMAX** *X X ... X*) [NoSpread Function]

Returns the maximum of *X*, *X*, ..., *X*. (IMAX) returns the smallest possible large integer, the value of MIN.INTEGER.

(**IEQP** *X Y*) [Function]

Returns T if *X* and *Y* are equal integers; NIL otherwise. Note that EQ may be used if *X* and *Y* are known to be small integers. IEQP converts *X* and *Y* to integers, e.g., (IEQP 2000 2000.3) => T.

(**FIX** *N*) [Function]

If *N* is an integer, returns *N*. Otherwise, converts *N* to an integer by truncating fractional bits. For example, (FIX 2.3) => 2, (FIX -1.7) => -1.

Since FIX is also a programmer's assistant command (see Chapter 13), typing FIX directly to a Medley executive will not cause the function FIX to be called.

(**FIXR** *N*) [Function]

If *N* is an integer, returns *N*. Otherwise, converts *N* to an integer by rounding. FIXR will round towards the even number if *N* is exactly half way between two integers. For example, (FIXR 2.3) => 2, (FIXR -1.7) => -2, (FIXR 3.5) => 4).

(**GCD** *N N*) [Function]

Returns the greatest common divisor of *N* and *N*, (GCD 72 64)=8.

Logical Arithmetic Functions

(**LOGAND** *X X ... X*) [NoSpread Function]

Returns the logical AND of all its arguments, as an integer. Example:

(LOGAND 7 5 6) => 4

(**LOGOR** *X X ... X*) [NoSpread Function]

Returns the logical OR of all its arguments, as an integer. Example:

(LOGOR 1 3 9) => 11

(**LOGXOR** *X X ... X*) [NoSpread Function]

Returns the logical exclusive OR of its arguments, as an integer. Example:

(LOGXOR 11 5) => 14
(LOGXOR 11 5 9) = (LOGXOR 14 9) => 7

(**LSH** *X N*) [Function]

(Arithmetic) "Left Shift." Returns *X* shifted left *N* places, with the sign bit unaffected. *X* can be positive or negative. If *N* is negative, *X* is shifted right *-N* places.

NUMBERS AND ARITHMETIC FUNCTIONS

(**RSH** *X N*) [Function]

(Arithmetic) “Right Shift.” Returns *X* shifted right *N* places, with the sign bit unaffected, and copies of the sign bit shifted into the leftmost bit. *X* can be positive or negative. If *N* is negative, *X* is shifted left *-N* places.

Warning: Be careful if using RSH to simulate division; RSHing a negative number isn’t the same as dividing by a power of two.

(**LLSH** *X N*) [Function]

(**LRSH** *X N*) [Function]

“Logical Left Shift” and “Logical Right Shift”. The difference between a logical and arithmetic right shift lies in the treatment of the sign bit. Logical shifting treats it just like any other bit; arithmetic shifting will not change it, and will “propagate” rightward when actually shifting rightwards. Note that shifting (arithmetic) a negative number “all the way” to the right yields *-1*, not *0*.

Note: LLSH and LRSH always operate mod-2³² arithmetic. Passing a bignum to either of these will cause an error. LRSH of negative numbers will shift 0s into the high bits.

(**INTEGERLENGTH** *X*) [Function]

Returns the number of bits needed to represent *X*. This is equivalent to: $1 + \text{floor}[\log_2[\text{abs}[X]]]$. (**INTEGERLENGTH** 0) = 0.

(**POWEROFTWO** *X*) [Function]

Returns non-NIL if *X* (coerced to an integer) is a power of two.

(**EVENP** *X Y*) [NoSpread Function]

If *Y* is not given, equivalent to (**ZEROP** (**IMOD** *X* 2)); otherwise equivalent to (**ZEROP** (**IMOD** *X Y*)).

(**ODDP** *N MODULUS*) [NoSpread Function]

Equivalent to (**NOT** (**EVENP** *N MODULUS*)). *MODULUS* defaults to 2.

(**LOGNOT** *N*) [Macro]

Logical negation of the bits in *N*. Equivalent to (**LOGXOR** *N* -1).

(**BITTEST** *N MASK*) [Macro]

Returns T if any of the bits in *MASK* are on in the number *N*. Equivalent to (**NOT** (**ZEROP** (**LOGAND** *N MASK*))).

(**BITCLEAR** *N MASK*) [Macro]

Turns off bits from *MASK* in *N*. Equivalent to (**LOGAND** *N* (**LOGNOT** *MASK*)).

(**BITSET** *N MASK*) [Macro]

Turns on the bits from *MASK* in *N*. Equivalent to (**LOGOR** *N MASK*).

INTERLISP-D REFERENCE MANUAL

(MASK.1'S POSITION SIZE) [Macro]

Returns a bit-mask with *SIZE* one-bits starting with the bit at *POSITION*. Equivalent to
(LLSH (SUB1 (EXPT 2 *SIZE*)) *POSITION*).

(MASK.0'S POSITION SIZE) [Macro]

Returns a bit-mask with all one bits, except for *SIZE* bits starting at *POSITION*.
Equivalent to (LOGNOT (MASK.1'S *POSITION SIZE*)).

(LOADBYTE N POS SIZE) [Function]

Extracts *SIZE* bits from *N*, starting at position *POS*. Equivalent to (LOGAND (RSH *N POS*)
(MASK.1'S 0 *SIZE*)).

(DEPOSITBYTE N POS SIZE VAL) [Function]

Insert *SIZE* bits of *VAL* at position *POS* into *N*, returning the result. Equivalent to

(LOGOR (BITCLEAR *N* (MASK.1'S *POS SIZE*))
(LSH (LOGAND *VAL* (MASK.1'S 0 *SIZE*))
POS))

(ROT X N FIELD SIZE) [Function]

“Rotate bits in field”. It performs a bitwise left-rotation of the integer *X*, by *N* places,
within a field of *FIELD SIZE* bits wide. Bits being shifted out of the position selected by
(EXPT 2 (SUB1 *FIELD SIZE*)) will flow into the “units” position.

The notions of position and size can be combined to make up a “byte specifier”, which is constructed
by the macro **BYTE** [note reversal of arguments as compared with the above functions]:

(BYTE SIZE POSITION) [Macro]

Constructs and returns a “byte specifier” containing *SIZE* and *POSITION*.

(BYTESIZE BYTESPEC) [Macro]

Returns the *SIZE* component of the “byte specifier” *BYTESPEC*.

(BYTEPOSITION BYTESPEC) [Macro]

Returns the *POSITION* component of the “byte specifier” *BYTESPEC*.

(LDB BYTESPEC VAL) [Macro]

Equivalent to

(LOADBYTE *VAL* (BYTEPOSITION *BYTESPEC*) (BYTESIZE *BYTESPEC*))

(DPB N BYTESPEC VAL) [Macro]

Equivalent to

(DEPOSITBYTE *VAL* (BYTEPOSITION *BYTESPEC*) (BYTESIZE *BYTESPEC*) *N*)

NUMBERS AND ARITHMETIC FUNCTIONS

Floating-Point Arithmetic

A floating-point number is input as a signed integer, followed by a decimal point, and another sequence of digits called the fraction, followed by an exponent (represented by E followed by a signed integer) and terminated by a delimiter.

Both signs are optional, and either the fraction following the decimal point, or the integer preceding the decimal point may be omitted. One or the other of the decimal point or exponent may also be omitted, but at least one of them must be present to distinguish a floating-point number from an integer. For example, the following will be recognized as floating-point numbers:

5.	5.00	5.01	.3
5E2	5.1E2	5E-3	-5.2E+6

Floating-point numbers are printed using the format control specified by the function FLTFMT (see Chapter 25). FLTFMT is initialized to T, or free format. For example, the above floating-point numbers would be printed free format as:

5.0	5.0	5.01	.3
500.0	510.0	.005	-5.2E6

Floating-point numbers are created by the reader when a "." or an E appears in a number, e.g., 1000 is an integer, 1000. a floating-point number, as are 1E3 and 1.E3. Note that 1000D, 1000F, and 1E3D are perfectly legal literal atoms. Floating-point numbers are also created by PACK and MKATOM, and as a result of arithmetic operations.

PRINTNUM (see Chapter 25) permits greater control over the printed appearance of floating-point numbers, allowing such things as left-justification, suppression of trailing decimals, etc.

The floating-point number range is stored in the following variables:

MIN.FLOAT [Variable]

The smallest possible floating-point number.

MAX.FLOAT [Variable]

The largest possible floating-point number.

All of the functions described below work on floating-point numbers. Unless specified otherwise, if given an integer, they first convert the number to a floating-point number, e.g., (FPLUS 1 2.3) <=> (FPLUS 1.0 2.3) => 3.3; if given a non-numeric argument, they generate an error, Non-numeric arg.

(**FPLUS** X X ... X) [NoSpread Function]

X + X + ... + X

(**FMINUS** X) [Function]

- X

(**FDIFFERENCE** X Y) [Function]

X - Y

INTERLISP-D REFERENCE MANUAL

(**FTIMES** *X X ... X*) [NoSpread Function]

*X * X * ... * X*

(**FQUOTIENT** *X Y*) [Function]

X / Y.

The results of division by zero and floating-point overflow is determined by the function OVERFLOW.

(**FREMAINDER** *X Y*) [Function]

Returns the remainder when *X* is divided by *Y*. Equivalent to:

(FDIFFERENCE *X* (FTIMES *Y* (FIX (FQUOTIENT *X Y*))))

Example:

(FREMAINDER 7.5 2.3) => 0.6

(**FGREATERP** *X Y*) [Function]

T, if *X* > *Y*, NIL otherwise.

(**FLESSP** *X Y*) [Function]

T, if *X* < *Y*, NIL otherwise.

(**FEQP** *X Y*) [Function]

Returns T if *X* and *Y* are equal floating-point numbers; NIL otherwise. FEQP converts *X* and *Y* to floating-point numbers.

(**FMIN** *X X ... X*) [NoSpread Function]

Returns the minimum of *X*, *X*, ..., *X*. (FMIN) returns the largest possible floating-point number, the value of MAX.FLOAT.

(**FMAX** *X X ... X*) [NoSpread Function]

Returns the maximum of *X*, *X*, ..., *X*. (FMAX) returns the smallest possible floating-point number, the value of MIN.FLOAT.

(**FLOAT** *X*) [Function]

Converts *X* to a floating-point number. Example:

(FLOAT 0) => 0.0

Transcendental Arithmetic Functions

(**EXPT** *A N*) [Function]

Returns A^N . If *A* is an integer and *N* is a positive integer, returns an integer, e.g. (EXPT 3 4) => 81, otherwise returns a floating-point number. If *A* is negative and *N* fractional, generates the error, Illegal exponentiation. If *N* is floating and either too large or too small, generates the error, Value out of range expt.

NUMBERS AND ARITHMETIC FUNCTIONS

(**SQRT** *N*) [Function]

Returns the square root of *N* as a floating-point number. *N* may be fixed or floating-point. Generates an error if *N* is negative.

(**LOG** *X*) [Function]

Returns the natural logarithm of *X* as a floating-point number. *X* can be integer or floating-point.

(**ANTILOG** *X*) [Function]

Returns the floating-point number whose logarithm is *X*. *X* can be integer or floating-point. Example:

(ANTILOG 1) = $e \Rightarrow 2.71828\dots$

(**SIN** *X* *RADIANSFLG*) [Function]

Returns the sine of *X* as a floating-point number. *X* is in degrees unless *RADIANSFLG* = T.

(**COS** *X* *RADIANSFLG*) [Function]

Similar to SIN.

(**TAN** *X* *RADIANSFLG*) [Function]

Similar to SIN.

(**ARCSIN** *X* *RADIANSFLG*) [Function]

The value of ARCSIN is a floating-point number, and is in degrees unless *RADIANSFLG* = T. In other words, if (ARCSIN *X* *RADIANSFLG*) = *Z* then (SIN *Z* *RADIANSFLG*) = *X*. The range of the value of ARCSIN is -90 to +90 for degrees, $-\pi/2$ to $\pi/2$ for radians. *X* must be a number between -1 and 1.

(**ARCCOS** *X* *RADIANSFLG*) [Function]

Similar to ARCSIN. Range is 0 to 180, 0 to π .

(**ARCTAN** *X* *RADIANSFLG*) [Function]

Similar to ARCSIN. Range is 0 to 180, 0 to π .

(**ARCTAN2** *Y* *X* *RADIANSFLG*) [Function]

Computes (ARCTAN (FQUOTIENT *Y* *X*) *RADIANSFLG*), and returns a corresponding value in the range -180 to 180 (or $-\pi$ to π), i.e. the result is in the proper quadrant as determined by the signs of *X* and *Y*.

Generating Random Numbers

(RAND LOWER UPPER)

[Function]

Returns a pseudo-random number between *LOWER* and *UPPER* inclusive, i.e., *RAND* can be used to generate a sequence of random numbers. If both limits are integers, the value of *RAND* is an integer, otherwise it is a floating-point number. The algorithm is completely deterministic, i.e., given the same initial state, *RAND* produces the same sequence of values. The internal state of *RAND* is initialized using the function *RANDSET*.

(RANDSET X)

[Function]

Returns the internal state of *RAND*. If *X* = *NIL*, just returns the current state. If *X* = *T*, *RAND* is initialized using the clocks, and *RANDSET* returns the new state. Otherwise, *X* is interpreted as a previous internal state, i.e., a value of *RANDSET*, and is used to reset *RAND*. For example,

```

←(SETQ OLDSTATE (RANDSET))
...
←(for X from 1 to 10 do (PRIN1 (RAND 1 10)))
2847592748NIL
←(RANDSET OLDSTATE)
...
←(for X from 1 to 10 do (PRIN1 (RAND 1 10)))
2847592748NIL

```

NUMBERS AND ARITHMETIC FUNCTIONS

[This page intentionally left blank]

RECORDS AND DATA STRUCTURES

Hiding the details of your code makes it more readable, and lets you program more efficiently. Data structures are a good example: You're better off if you can say "Fetch me the `SPEED` field from this `AIRPLANE`" rather than having to say `(CAR (CDDDR (CADR AIRPLANE)))`. You can declare data structures used by your programs, then work with field names rather than access details. Using the declarations, Medley performs the access/storage operations you request. If you change a data structure's declaration, your programs automatically adjust.

You describe the format of a data structure (record) by making a "record declaration" (see the Record Declarations section below). The record declaration is a description of the record, associating names with its various parts, or "fields". For example, the record declaration

```
(RECORD MSG (FROM TO TEXT))
```

describes a data structure called `MSG`, that has three fields: `FROM`, `TO`, and `TEXT`. You can refer to these fields by name, to get their values or to store new values into them, by using `FETCH` and `REPLACE`:

```
(fetch (MSG FROM) of MYMSG)
(replace (MSG TO) of MYMSG with "John Doe")
```

You create new `MSGs` with `CREATE`:

```
(SETQ MYMSG (create MSG))
```

and `TYPE?` tells you whether some object is a `MSG`:

```
(IF (TYPE? MSG THIS-THING) then (SEND-MSG THIS-THING))
```

So far we've said nothing about *how* your `MSG` is represented—when you're writing `FETCHES` and `REPLACES`, it doesn't matter. But you *can* control the representation: The symbol `RECORD` in the declaration above causes each `MSG` to be represented as a list. There are a number of options, up to creating a completely new Lisp data type; each has its own specifier symbol, and they're described in detail below.

The record package is implemented using `DWIM` and `CLISP`, so it will do spelling correction on field names, record types, etc. Record operations are translated using all `CLISP` declarations in effect (standard/fast/undoable).

The file manager's `RECORDS` command lets you give record declarations (see Chapter 17), and `FILES?` and `CLEANUP` will tell you about record declarations that need to be dumped.

FETCH and REPLACE

The fields of a record are accessed and changed with `FETCH` and `REPLACE`. If `x` is a `MSG` data structure, `(fetch FROM of x)` will return the value of the `FROM` field of `x`, and `(replace FROM of x with y)` will replace this field with the value of `y`. In general, the value of a `REPLACE` operation is the same as the value stored into the field.

Note that `(fetch FROM of x)` assumes that `x` is an instance of the record `MSG`—the interpretation of `(fetch FROM of x)` never depends on the *value* of `x`. If `x` is not a `MSG`, this may produce incorrect results.

If there is another record declaration, `(RECORD REPLY (TEXT RESPONSE))`, then `(fetch TEXT of x)` is ambiguous, because `x` could be either a `MSG` or a `REPLY` record. In this case, an error will occur, `Ambiguous record field`. To clarify this, give `FETCH` and `REPLACE` a list for their "field" argument: `(fetch (MSG TEXT) of x)` will fetch

INTERLISP-D REFERENCE MANUAL

the `TEXT` field of a `MSG` record. If a field has an *identical* interpretation in two declarations, e.g., if the field `TEXT` occurred in the same location within the declarations of `MSG` and `REPLY`, then `(fetch TEXT of X)` would *not* be ambiguous.

If there's a conflict, “user” record declarations take precedence over “system” record declarations. System records are declared by including `(SYSTEM)` in the declaration (see the Record Declarations section below). All of the records defined in the standard Medley system are system records.

Another complication can occur if the fields of a record are themselves records. The fields of a record can be further broken down into sub-fields by a “subdeclaration” within the record declaration. For example,

```
(RECORD NODE (POSITION . LABEL) (RECORD POSITION (XLOC . YLOC)))
```

lets you access the `POSITION` field with `(fetch POSITION of X)`, or its subfield `XLOC` with `(fetch XLOC of X)`.

You may also declare that field name in a *separate* record declaration. For instance, the `TEXT` field in the `MSG` and `REPLY` records above may be subdivided with the separate record declaration `(RECORD TEXT (HEADER TXT))`. You get to fields of subfields (to any level of nesting) by specifying the “data path” as a list of record / field names, where there is some path from each record to the next in the list. For instance,

```
(fetch (MSG TEXT HEADER) of X)
```

treats `x` as a `MSG` record, fetches its `TEXT` field, and fetches *its* `HEADER` field. You only need to give enough of the data path to disambiguate it. In this case, `(fetch (MSG HEADER) of X)` is sufficient: Medley searches among all current record declarations for a path from each name to the next, considering first local declarations (see Chapter 21) and then global ones. Of course, if you had two records with `HEADER` fields, you get an *Ambiguous data path error*.

`FETCH` and `REPLACE` are translated using the CLISP declarations in effect (see Chapter 21). `FFETCH` and `FREPLACE` are fast versions that don't do any type checking. `/REPLACE` insures undoable declarations.

Record Declarations

You define records by evaluating declarations of the form:

```
(RECORD-TYPE RECORD-NAME RECORD-FIELDS . RECORD-TAIL)
```

RECORD-TYPE specifies the “type” of data you're declaring, and controls how instances will be stored internally. The different record types are described below.

RECORD-NAME is a symbol used to identify the record declaration for `CREATE`, `TYPE?`, `FETCH` and `REPLACE`, and dumping to files (see Chapter 17). `DATATYPE` and `TYPERECORD` declarations also use *RECORD-NAME* to identify the data structure (as described below).

RECORD-FIELDS describes the structure of the record. Its exact interpretation varies with *RECORD-TYPE*. Generally, it names the fields within the record that can be accessed with `FETCH` and `REPLACE`.

RECORD-TAIL is an optional list where you can specify default values for record fields, special `CREATE` and `TYPE?` forms, and subdeclarations (described below).

Record declarations are Lisp programs, and could be included in functions, changing a record declaration at run-time. *Don't do it*. You risk creating a structure with one declaration, and trying to fetch from it with another—complete chaos results. If you need to change record declarations dynamically, consider using association lists or property lists.

RECORDS AND DATA STRUCTURES

Record Types

The *RECORD-TYPE* field of the record declaration specifies how the data object is created, and how the various record fields are accessed. Depending on the record type, the record fields may be stored in a list, or in an array, or on a symbol's property list. The following record types are defined:

RECORD

[Record Type]

The fields of a *RECORD* are kept in a list. *RECORD-FIELDS* is a list; each non-*NIL* symbol is a field-name to be associated with the corresponding element or tail of a list structure. For example, with the declaration `(RECORD MSG (FROM TO . TEXT))`, `(fetch FROM of X)` translates as `(CAR X)`.

NIL can be used as a place marker for an unnamed field, e.g., `(A NIL B)` describes a three element list, with *B* corresponding to the third element. A number may be used to indicate a sequence of *NIL*s, e.g. `(A 4 B)` is interpreted as `(A NIL NIL NIL NIL B)`.

DATATYPE

[Record Type]

Defines a new user data type with type name *RECORD-NAME*. Unlike other record types, the instances of a *DATATYPE* are represented with a completely new Lisp type, and not in terms of other existing types.

RECORD-FIELDS is a list of field specifications, where each specification is either a list `(FIELDNAME FIELDTYPE)`, or an symbol *FIELDNAME*. If *FIELDTYPE* is omitted, it defaults to *POINTER*. Possible values for *FIELDTYPE* are:

POINTER Field contains a pointer to any arbitrary Interlisp object.

INTEGER

FIXP Field contains a signed integer. Caution: An *INTEGER* field is not capable of holding everything that satisfies *FIXP*, such as bignums.

FLOATING

FLOATP Field contains a floating point number.

SIGNEDWORD Field contains a 16-bit signed integer.

FLAG Field is a one bit field that "contains" *T* or *NIL*.

BITS N Field contains an *N*-bit unsigned integer.

BYTE Equivalent to *BITS 8*.

WORD Equivalent to *BITS 16*.

XPOINTER Field contains a pointer like *POINTER*, but the field is *not* reference counted by the garbage collector. *XPOINTER* fields are useful for implementing back-pointers in structures that would be circular and not otherwise collected by the reference-counting garbage collector.

Warning: Use *XPOINTER* fields with great care. You can damage the integrity of the storage allocation system by using pointers to objects that have been garbage collected. Code that uses *XPOINTER* fields should be sure

INTERLISP-D REFERENCE MANUAL

that the objects pointed to have not been garbage collected. This can be done in two ways: The first is to maintain the object in a global structure, so that it is never garbage collected until explicitly deleted from the structure, at which point the program must invalidate all the `XPOINTER` fields of other objects pointing at it. The second is to declare the object as a `DATATYPE` beginning with a `POINTER` field that the program maintains as a pointer to an object of another type (e.g., the object containing the `XPOINTER` pointing back at it), and test that field for reasonableness whenever using the contents of the `XPOINTER` field.

For example, the declaration

```
(DATATYPE FOO
  ((FLG BITS 12) TEXT HEAD (DATE BITS 18)
   (PRIO FLOATP) (READ? FLAG)))
```

would define a data type `FOO` with two pointer fields, a floating point number, and fields for a 12 and 18 bit unsigned integers, and a flag (one bit). Fields are allocated in such a way as to optimize the storage used and not necessarily in the order specified. Generally, a `DATATYPE` record is much more storage compact than the corresponding `RECORD` structure would be; in addition, access is faster.

Since the user data type must be set up at *run-time*, the `RECORDS` file package command will dump a `DECLAREDATATYPE` expression as well as the `DATATYPE` declaration itself. If the record declaration is otherwise not needed at runtime, it can be kept out of the compiled file by using a `(DECLARE: DONTCOPY --)` expression (see Chapter 17), but it is still necessary to ensure that the datatype is properly initialized. For this, one can use the `INITRECORDS` file package command (see Chapter 17), which will dump only the `DECLAREDATATYPE` expression.

Note: When defining a new data type, it is sometimes useful to call the function `DEFPRT` (see Chapter 25) to specify how instances of the new data type should be printed. This can be specified in the record declaration by including an `INIT` record specification (see the Optional Record Specifications section below), e.g. `(DATATYPE QV.TYPE ... (INIT (DEFPRT 'QV.TYPE (FUNCTION PRINT.QV.TYPE))))`.

`DATATYPE` declarations cannot be used within local record declarations (see Chapter 21).

TYPERECORD

[Record Type]

Similar to `RECORD`, but the record name is added to the front of the list structure to signify what “type” of record it is. This type field is used in the translation of `TYPE?` expressions. `CREATE` will insert an extra field containing `RECORD-NAME` at the beginning of the structure, and the translation of the access and storage functions will take this extra field into account. For example, for `(TYPERECORD MSG (FROM TO . TEXT))`, `(fetch FROM of X)` translates as `(CADR X)`, `not (CAR X)`.

ASSOCRECORD

[Record Type]

Describes lists where the fields are stored in association list format:

```
((FIELDNAME . VALUE) (FIELDNAME . VALUE) ...)
```

`RECORD-FIELDS` is a list of symbols, the permissible field names in the association list. Access is done with `ASSOC` (or `FASSOC`, if the current CLISP declarations are `FAST`, see Chapter 21), storing with `PUTASSOC`.

RECORDS AND DATA STRUCTURES

PROPRECORD

[Record Type]

Describes lists where the fields are stored in property list format:

(FIELDNAME VALUE FIELDNAME VALUE ...)

RECORD-FIELDS is a list of symbols, the permissible field names in the property list. Access is done with `LISTGET`, storing with `LISTPUT`.

Both `ASSOCRECORD` and `PROPRECORD` are useful for defining data structures where many of the fields are `NIL`. Creating one these record types only stores those fields that are non-`NIL`. Note, however, that with the record declaration `(PROPRECORD FIE (H I J))` the expression `(create FIE)` would still construct `(H NIL)`, since a later operation of `(replace J of X with Y)` could not possibly change the instance of the record if it were `NIL`.

ARRAYRECORD

[Record Type]

`ARRAYRECORDs` are stored as arrays. *RECORD-FIELDS* is a list of field names that are associated with the corresponding elements of an array. `NIL` can be used as a place marker for an unnamed field (element). Positive integers can be used as abbreviation for the corresponding number of `NILs`. For example, `(ARRAYRECORD (ORG DEST NIL ID 3 TEXT))` describes an eight-element array, with `ORG` corresponding to the first element, `ID` to the fourth, and `TEXT` to the eighth.

`ARRAYRECORD` only creates arrays of pointers. Other kinds of arrays must be implemented with `ACCESSFNS` (see below).

HASHLINK

[Record Type]

The `HASHLINK` record type can be used with any type of data object: it specifies that the value of a single field can be accessed by hashing the data object in a given hash array. Since the `HASHLINK` record type describes an access method, rather than a data structure, `CREATE` is meaningless for `HASHLINK` records.

RECORD-FIELDS is either a symbol *FIELD-NAME*, or a list *(FIELD-NAME HARRAYNAME HARRAYSIZE)*. *HARRAYNAME* is a variable whose value is the hash array to be used; if not given, `SYSHASHARRAY` is used. If the value of the variable *HARRAYNAME* is not a hash array (at the time of the record declaration), it will be set to a new hash array with a size of *HARRAYSIZE*. *HARRAYSIZE* defaults to 100.

The `HASHLINK` record type is useful as a subdeclaration to other records to add additional fields to already existing data structures (see the Optional Record Specifications section below). For example, suppose that `FOO` is a record declared with `(RECORD FOO (A B C))`. To add a new field `BAR`, without modifying the existing data structures, redeclare `FOO` with:

`(RECORD FOO (A B C) (HASHLINK FOO (BAR BARHARRAY)))`

Now, `(fetch BAR of X)` will translate into `(GETHASH X BARHARRAY)`, hashing off the existing list `X`.

ATOMRECORD

[Record Type]

`ATOMRECORDs` are stored on the property lists of symbols. *RECORD-FIELDS* is a list of property names. Accessing is performed with `GETPROP`, storing with `PUTPROP`. The `CREATE` expression is not initially defined for `ATOMRECORD` records.

INTERLISP-D REFERENCE MANUAL

BLOCKRECORD

[Record Type]

`BLOCKRECORD` is used in low-level system programming to “overlay” an organized structure over an arbitrary piece of raw storage. *RECORD-FIELDS* is interpreted exactly as with a `DATATYPE` declaration, except that fields are *not* automatically rearranged to maximize storage efficiency. Like an `ACCESSFNS` record, a `BLOCKRECORD` does not have concrete instances; it merely provides a way of interpreting some existing block of storage. So you can’t create an instance of a `BLOCKRECORD` (unless the declaration includes an explicit `CREATE` expression), nor is there a default `type?` expression for a `BLOCKRECORD`.

Warning: Exercise caution in using `BLOCKRECORD` declarations, as they let you fetch and store arbitrary data in arbitrary locations, thereby evading Medley’s normal type system. Except in very specialized situations, a `BLOCKRECORD` should never contain `POINTER` or `XPOINTER` fields, nor be used to overlay an area of storage that contains pointers. Such use could compromise the garbage collector and storage allocation system. You are responsible for ensuring that all `FETCH` and `REPLACE` expressions are performed only on suitable objects, as no type testing is performed.

A typical use for a `BLOCKRECORD` in user code is to overlay a non-pointer portion of an existing `DATATYPE`. For this use, the `LOCF` macro is useful. (`LOCF (fetch FIELD of DATUM)`) can be used to refer to the storage that begins at the first word that contains *FIELD* of *DATUM*. For example, to define a new kind of Ethernet packet, you could overlay the “body” portion of the `ETHERPACKET` datatype declaration as follows:

```
(ACCESSFNS MYPACKET
  ((MYBASE (LOCF (fetch (ETHERPACKET EPBODY) of DATUM))))
  (BLOCKRECORD MYBASE
    ((MYTYPE WORD) (MYLENGTH WORD) (MYSTATUS BYTE)
      (MYERRORCODE BYTE) (MYDATA INTEGER)))
    (TYPE? (type? ETHERPACKET DATUM))))
```

With this declaration in effect, the expression (`fetch MYLENGTH of PACKET`) would retrieve the second 16-bit field beyond the place inside `PACKET` where the `EPBODY` field starts.

ACCESSFNS

[Record Type]

`ACCESSFNS` lets you specify arbitrary functions to fetch and store data. For each field name, you specify how it is to be accessed and set. This lets you use arbitrary data structures, with complex access methods. Most often, `ACCESSFNS` are useful when you can compute one field’s value from other fields. If you’re representing a time period by its start and duration, you could add an `ACCESSFNS` definition for the ending time that did the obvious addition.

RECORD-FIELDS is a list of elements of the form (*FIELD-NAME ACCESSDEF SETDEF*). *ACCESSDEF* should be a function of one argument, the datum, and will be used for accessing the value of the field. *SETDEF* should be a function of two arguments, the datum and the new value, and will be used for storing a new value in a field. *SETDEF* may be omitted, in which case, no storing operations are allowed.

ACCESSDEF and/or *SETDEF* may also be a form written in terms of variables `DATUM` and (`in SETDEF`) `NEWVALUE`. For example, given the declaration

```
[ACCESSFNS FOO
  ((FIRSTCHAR (NTHCHAR DATUM 1) (RPLSTRING DATUM 1 NEWVALUE)) (RESTCHARS (SUBSTRING DATUM 2)
```

RECORDS AND DATA STRUCTURES

(replace (FOO FIRSTCHAR) of X with Y) would translate to (RPLSTRING X 1 Y). Since no *SETDEF* is given for the *RESTCHARS* field, attempting to perform (replace (FOO RESTCHARS) of X with Y) would generate an error, Replace undefined for field. Note that *ACCESSFNS* do not have a *CREATE* definition. However, you may supply one in the defaults or subdeclarations of the declaration, as described below. Attempting to *CREATE* an *ACCESSFNS* record without specifying a create definition will cause an error Create not defined for this record.

ACCESSDEF and *SETDEF* can also be a property list which specify *FAST*, *STANDARD* and *UNDOABLE* versions of the *ACCESSFNS* forms, e.g.

```
[ACCESSFNS LITATOM
  ((DEF (STANDARD GETD FAST FGETD)
        (STANDARD PUTD UNDOABLE /PUTD])
```

means if *FAST* declaration is in effect, use *FGETD* for fetching, if *UNDOABLE*, use */PUTD* for saving (see *CLISP* declarations, see Chapter 21).

SETDEF forms should be written so that they return the new value, to be consistent with *REPLACE* operations for other record types. The *REPLACE* does not enforce this, though.

ACCESSFNS let you use data structures not specified by one of the built-in record types. For example, one possible representation of a data structure is to store the fields in *parallel* arrays, especially if the number of instances required is known, and they needn't be garbage collected. To implement *LINK* with two fields *FROM* and *TO*, you'd have two arrays *FROMARRAY* and *TOARRAY*. The representation of an "instance" of *LINK* would be an integer, used to index into the arrays. This can be accomplished with the declaration:

```
[ACCESSFNS LINK
  ((FROM (ELT FROMARRAY DATUM)
        (SETA FROMARRAY DATUM NEWVALUE))
   (TO (ELT TOARRAY DATUM)
        (SETA TOARRAY DATUM NEWVALUE)))
  (CREATE (PROG1 (SETQ LINKCNT (ADD1 LINKCNT))
               (SETA FROMARRAY LINKCNT FROM)
               (SETA TOARRAY LINKCNT TO)))
  (INIT (PROGN
         (SETQ FROMARRAY (ARRAY 100))
         (SETQ TOARRAY (ARRAY 100))
         (SETQ LINKCNT 0))])
```

To create a new *LINK*, a counter is incremented and the new elements stored. (Note: The *CREATE* form given the declaration probably should include a test for overflow.)

Optional Record Specifications

After the *RECORD-FIELDS* item in a record declaration expression there can be an arbitrary number of additional expressions in *RECORD-TAIL*. These expressions can be used to specify default values for record fields, special *CREATE* and *TYPE?* forms, and subdeclarations. The following expressions are permitted:

FIELD-NAME $\leftarrow$ *FORM* Allows you to specify within the record declaration the default value to be stored in *FIELD-NAME* by a *CREATE* (if no value is given within the *CREATE* expression itself). Note that *FORM* is evaluated at *CREATE* time, not when the declaration is made.

(*CREATE FORM*) Defines the manner in which *CREATE* of this record should be performed. This provides a way of specifying how *ACCESSFNS* should be created or overriding the usual definition of *CREATE*. If *FORM* contains the field-names of the declaration as variables, the forms given in the

INTERLISP-D REFERENCE MANUAL

CREATE operation will be substituted in. If the word DATUM appears in the create form, the *original* CREATE definition is inserted. This effectively allows you to “advise” the create.

(INIT FORM) Specifies that FORM should be evaluated when the record is declared. FORM will also be dumped by the INITRECORDS file package command (see Chapter 17).

For example, see the example of an ACCESSFNS record declaration above. In this example, FROMARRAY and TOARRAY are initialized with an INIT form.

(TYPE? FORM) Defines the manner in which TYPE? expressions are to be translated. FORM may either be an expression in terms of DATUM or a function of one argument.

(SUBRECORD NAME .
DEFAULTS) NAME must be a field that appears in the current declaration and the name of another record. This says that, for the purposes of translating CREATE expressions, substitute the top-level declaration of NAME for the SUBRECORD form, adding on any defaults specified.

For example: Given (RECORD B (E F G)), (RECORD A (B C D) (SUBRECORD B)) would be treated like (RECORD A (B C D) (RECORD B (E F G))) for the purposes of translating CREATE expressions.

a subdeclaration If a record declaration expression occurs among the record specifications of another record declaration, it is known as a “subdeclaration.” Subdeclarations are used to declare that fields of a record are to be interpreted as another type of record, or that the record data object is to be interpreted in more than one way.

The RECORD-NAME of a subdeclaration must be either the RECORD-NAME of its immediately superior declaration or one of the superior’s field-names. Instead of identifying the declaration as with top level declarations, the record-name of a subdeclaration identifies the parent field or record that is being described by the subdeclaration. Subdeclarations can be nested to an arbitrary depth.

Giving a subdeclaration (RECORD NAME NAME) is a simple way of defining a *synonym* for the field NAME.

It is possible for a given field to have more than one subdeclaration. For example, in

```
(RECORD FOO (A B) (RECORD A (C D)) (RECORD A (Q R)))
```

(Q R) and (C D) are “overlaid,” i.e. (fetch Q of X) and (fetch C of X) would be equivalent. In such cases, the *first* subdeclaration is the one used by CREATE.

(SYNONYM FIELD

RECORDS AND DATA STRUCTURES

(*SYN* ... *SYN*) *FIELD* must be a field that appears in the current declaration. This defines *SYN* ... *SYN* all as synonyms of *FIELD*. If there is only one synonym, this can be written as (*SYNONYM FIELD SYN*).

(*SYSTEM*) If (*SYSTEM*) is included in a record declaration, this indicates that the record is a “system” record rather than a “user” record. The only distinction between the two types of records is that “user” record declarations take precedence over “system” record declarations, in cases where an unqualified field name would be considered ambiguous. All of the records defined in the standard Medley system are defined as system records.

CREATE

You can create RECORDS by hand if you like, using *CONS*, *LIST*, etc. But that defeats the whole point of hiding implementation details. So much easier to use:

(create *RECORD-NAME* . *ASSIGNMENTS*)

CREATE translates into an appropriate Interlisp form that uses *CONS*, *LIST*, *PUTHASH*, *ARRAY*, etc., to create the new datum with the its fields initialized to the values you specify. *ASSIGNMENTS* is optional and may contain expressions of the following form:

FIELD-NAME ← *FORM* Specifies initial value for *FIELD-NAME*.

USING *FORM* *FORM* is an existing instance of *RECORD-NAME*. If you don’t specify a value for some field, the value of the corresponding field in *FORM* is to be used.

COPYING *FORM* Like USING, but the corresponding values are copied (with *COPYALL*).

REUSING *FORM* Like USING, but wherever possible, the corresponding *structure* in *FORM* is used.

SMASHING *FORM* A new instance of the record is not created at all; rather, new field values are smashed into *FORM*, which CREATE then returns.

When it makes a difference, Medley goes to great pains to make its translation do things in the same order as the original CREATE expression. For example, given the declaration (RECORD CONS (CAR . CDR)), the expression (create CONS CDR←X CAR←Y) will translate to (CONS Y X), but (create CONS CDR←(FOO) CAR←(FIE)) will translate to ((LAMBDA (\$\$1) (CONS (PROGN (SETQ \$\$1 (FOO)) (FIE)) \$\$1))) because FOO might set some variables used by FIE.

How are USING and REUSING different? (create RECORD reusing *FORM* ...) doesn’t do any destructive operations on the value of *FORM*, but will incorporate as much as possible of the old data structure into the new one. On the other hand, (create RECORD using *FORM* ...) will create a completely new data structure, with only the *contents* of the fields re-used. For example, REUSING a PROPRECORD just CONSES the new property names and values onto the list, while USING copies the top level of the list. Another example of this distinction occurs when a field is elaborated by a subdeclaration: USING will create a new instance of the sub-record, while REUSING will use the old contents of the field (unless some field of the subdeclaration is assigned in the CREATE expression.)

INTERLISP-D REFERENCE MANUAL

If the value of a field is neither explicitly specified, nor implicitly specified via `USING`, `COPYING` or `REUSING`, the default value in the declaration is used, if any, otherwise `NIL`. (For `BETWEEN` fields in `DATATYPE` records, `N` is used; for other non-pointer fields zero is used.) For example, following `(RECORD A (B C D) D ← 3)`

```
(create A B ← T) ==> (LIST T NIL 3)
(create A B ← T using X) ==> (LIST T (CADR X) (CADDR X))
(create A B ← T copying X) ==> [LIST T (COPYALL (CADR X)) (COPYALL (CADDR X))
(create A B ← T reusing X) ==> (CONS T (CDR X))
```

TYPE?

The record package allows you to test if a given datum “looks like” an instance of a record. This can be done via an expression of the form `(type? RECORD-NAME FORM)`.

`TYPE?` is mainly intended for records with a record type of `DATATYPE` or `TYPERECORD`. For `DATATYPES`, the `TYPE?` check is exact; i.e. the `TYPE?` expression will return non-`NIL` only if the value of `FORM` is an instance of the record named by `RECORD-NAME`. For `TYPERECORDS`, the `TYPE?` expression will check that the value of `FORM` is a list beginning with `RECORD-NAME`. For `ARRAYRECORDS`, it checks that the value is an array of the correct size. For `PROPRECORDS` and `ASSOCRECORDS`, a `TYPE?` expression will make sure that the value of `FORM` is a property / association list with property names among the field-names of the declaration.

There is no built-in type test for records of type `ACCESSFNS`, `HASHLINK` or `RECORD`. Type tests can be defined for these kinds of records, or redefined for the other kinds, by including an expression of the form `(TYPE? COM)` in the record declaration (see the Record Declarations section below). Attempting to execute a `TYPE?` expression for a record that has no type test causes an error, `Type? not implemented for this record`.

WITH

Often one wants to write a complex expression that manipulates several fields of a single record. The `WITH` construct can make it easier to write such expressions by allowing one to refer to the fields of a record as if they were variables within a lexical scope:

```
(with RECORD-NAME RECORD-INSTANCE FORM ... FORM )
```

`RECORD-NAME` is the name of a record, and `RECORD-INSTANCE` is an expression which evaluates to an instance of that record. The expressions `FORM ... FORM` are evaluated so that references to variables which are field-names of `RECORD-NAME` are implemented via `FETCH` and `SETQs` of those variables are implemented via `REPLACE`.

For example, given

```
(RECORD REC1 (FLD1 FLD2))
(SETQ INST (create REC1 FLD1 ← 10 FLD2 ← 20))
```

Then the construct

```
(with REC1 INST (SETQ FLD2 (PLUS FLD1 FLD2)
```

is equivalent to

```
(replace FLD2 of INST with (PLUS (fetch FLD1 of INST) (fetch FLD2 of INST)
```

Warning: `WITH` is implemented by doing simple substitutions in the body of the forms, without regard for how the record fields are used. This means, for example, if the record `FOO` is defined by `(RECORD FOO (POINTER1 POINTER2))`, then the form

```
(with FOO X (SELECTQ Y (POINTER1 POINTER1) NIL)
```

RECORDS AND DATA STRUCTURES

will be translated as

```
(SELECTQ Y ((CAR X) (CAR X)) NIL)
```

Be careful that record field names are not used except as variables in the `WITH` forms.

Defining New Record Types

In addition to the built-in record types, you can declare your own record types by performing the following steps:

1. Add the new record-type to the value of `CLISPRECORDTYPES`.
2. Perform `(MOVD 'RECORD RECORD-TYPE)`.
3. Put the name of a function which will return the translation on the property list of `RECORD-TYPE`, as the value of the property `USERRECORDTYPE`. Whenever a record declaration of type `RECORD-TYPE` is encountered, this function will be passed the record declaration as its argument, and should return a *new* record declaration which the record package will then use in its place.

Manipulating Record Declarations

`(EDITREC NAME COM ... COM)`

[NLambda NoSpread Function]

`EDITREC` calls the editor on a copy of all declarations in which *NAME* is the record name or a field name. On exit, it redeclares those that have changed and undeclares any that have been deleted. If *NAME* is `NIL`, *all* declarations are edited.

COM ... COM are (optional) edit commands.

When you redeclare a global record, the translations of all expressions involving that record or any of its fields are automatically deleted from `CLISPARRAY`, and thus will be recomputed using the new information. If you change a *local* record declaration (see Chapter 21), or change some other CLISP declaration (see Chapter 21), e.g., `STANDARD` to `FAST`, and wish the new information to affect record expressions already translated, you must make sure the corresponding translations are removed, usually either by `CLISPIFYING` or using the `DW` edit macro.

`(RECLook RECNAME - - - -)`

[Function]

Returns the entire declaration for the record named *RECNAME*; `NIL` if there is no record declaration with name *RECNAME*. Note that the record package maintains internal state about current record declarations, so performing destructive operations (e.g. `NCONC`) on the value of `RECLook` may leave the record package in an inconsistent state. To change a record declaration, use `EDITREC`.

`(FIELDLOOK FIELDNAME)`

[Function]

Returns the list of declarations in which *FIELDNAME* is the name of a field.

`(RECORDFIELDNAMES RECORDNAME -)`

[Function]

Returns the list of fields declared in record *RECORDNAME*. *RECORDNAME* may either be a name or an entire declaration.

INTERLISP-D REFERENCE MANUAL

(RECORDACCESS FIELD DATUM DEC TYPE NEWVALUE) [Function]

TYPE is one of `FETCH`, `REPLACE`, `FFETCH`, `FREPLACE`, `/REPLACE` or their lowercase equivalents. *TYPE*=`NIL` means `FETCH`. If *TYPE* corresponds to a fetch operation, i.e. is `FETCH`, or `FFETCH`, `RECORDACCESS` performs (*TYPE* *FIELD* of *DATUM*). If *TYPE* corresponds to a replace, `RECORDACCESS` performs (*TYPE* *FIELD* of *DATUM* with *NEWVALUE*). *DEC* is an optional declaration; if given, *FIELD* is interpreted as a field name of that declaration.

Note that `RECORDACCESS` is relatively inefficient, although it is better than constructing the equivalent form and performing an `EVAL`.

(RECORDACCESSFORM FIELD DATUM TYPE NEWVALUE) [Function]

Returns the form that would be compiled as a result of a record access. *TYPE* is one of `FETCH`, `REPLACE`, `FFETCH`, `FREPLACE`, `/REPLACE` or their lowercase equivalents. *TYPE*=`NIL` means `FETCH`.

Changetran

Often, you'll want to assign a new value to some datum that is a function of its current value:

Incrementing a counter: `(SETQ X (IPLUS X 1))`

Pushing an item on the front of a list: `(SETQ X (CONS Y X))`

Popping an item off a list: `(PROG1 (CAR X) (SETQ X (CDR X)))`

Those are simple when you're working with a variable; it gets complicated when you're working with structured data. For example, if you want to modify `(CAR X)`, the above examples would be:

```
(CAR (RPLACA X (IPLUS (CAR X) 1)))
(CAR (RPLACA X (CONS Y (CAR X))))
(PROG1 (CAAR X) (RPLACA X (CDR X)))
```

and if you're changing an element in an array, `(ELT A N)`, the examples would be:

```
(SETA A N (IPLUS (ELT A N) 1))
(SETA A N (CONS Y (ELT A N)))
(PROG1 (CAR (ELT A N)) (SETA A N (CDR (ELT A N))))
```

Changetran is designed to provide a simpler way to express these common (but user-extensible) structure modifications. Changetran defines a set of CLISP words that encode the kind of modification to take place—pushing on a list, adding to a number, etc. More important, you only indicate the item to be modified once. Thus, the “change word” `ADD` is used to increase the value of a datum by the sum of a set of numbers. Its arguments are the datum, and a set of numbers to be added to it. The datum must be a variable or an accessing expression (involving `FETCH`, `CAR`, `LAST`, `ELT`, etc) that can be translated to the appropriate setting expression.

For example, `(ADD X 1)` is equivalent to:

```
(SETQ X (PLUS X 1))
```

and `(ADD (CADDR X) (FOO))` is equivalent to:

```
(CAR (RPLACA (CDDR X) (PLUS (FOO) (CADDR X))))
```

If the datum is a complicated form involving function calls, such as `(ELT (FOO X) (PIE Y))`, Changetran goes to some lengths to make sure that those subsidiary functions are evaluated only once, even though they are used in both the setting and accessing parts of the translation. You can rely on the fact that the forms will be evaluated only as often as they appear in your expression.

RECORDS AND DATA STRUCTURES

For `ADD` and all other changewords, the lowercase version (`add`, etc.) may also be specified. Like other CLISP words, change words are translated using all CLISP declarations in effect (see Chapter 21).

The following is a list of those change words recognized by `Changetran`. Except for `POP`, the value of all built-in changeword forms is defined to be the new value of the datum.

(`ADD` *DATUM* *ITEM* *ITEM* ...) [Change Word]

Adds the specified items to the current value of the datum, stores the result back in the datum location. The translation will use `IPLUS`, `PLUS`, or `FPLUS` according to the CLISP declarations in effect (see Chapter 21).

(`PUSH` *DATUM* *ITEM* *ITEM* ...) [Change Word]

CONSES the items onto the front of the current value of the datum, and stores the result back in the datum location. For example, `(PUSH X A B)` would translate as `(SETQ X (CONS A (CONS B X)))`.

(`PUSHNEW` *DATUM* *ITEM*) [Change Word]

Like `PUSH` (with only one item) except that the item is not added if it is already `FMEMB` of the datum's value.

Note that, whereas `(CAR (PUSH X 'FOO))` will always be `FOO`, `(CAR (PUSHNEW X 'FOO))` might be something else if `FOO` already existed in the middle of the list.

(`PUSHLIST` *DATUM* *ITEM* *ITEM* ...) [Change Word]

Similar to `PUSH`, except that the items are APPENDED in front of the current value of the datum. For example, `(PUSHLIST X A B)` translates as `(SETQ X (APPEND A B X))`.

(`POP` *DATUM*) [Change Word]

Returns `CAR` of the current value of the datum after storing its `CDR` into the datum. The current value is computed only once even though it is referenced twice. Note that this is the only built-in changeword for which the value of the form is not the new value of the datum.

(`SWAP` *DATUM* *DATUM*) [Change Word]

Sets *DATUM* to *DATUM* and vice versa.

(`CHANGE` *DATUM* *FORM*) [Change Word]

This is the most flexible of all change words: You give an arbitrary form describing what the new value should be. But it still highlights the fact that structure modification is happening, and still lets the datum appear only once. `CHANGE` sets *DATUM* to the value of *FORM**, where *FORM** is constructed from *FORM* by substituting the datum expression for every occurrence of the symbol *DATUM*. For example,

`(CHANGE (CAR X) (ITIMES DATUM 5))`

translates as

`(CAR (RPLACA X (ITIMES (CAR X) 5)))`.

INTERLISP-D REFERENCE MANUAL

`CHANGE` is useful for expressing modifications that are not built-in and are not common enough to justify defining a user-changeword.

You can define new change words. To define a change word, say `sub`, that subtracts items from the current value of the datum, you must put the property `CLISPPWORD`, value `(CHANGETRAN . sub)` on both the upper- and lower-case versions of `sub`:

```
(PUTPROP 'SUB 'CLISPPWORD '(CHANGETRAN . sub))
(PUTPROP 'sub 'CLISPPWORD '(CHANGETRAN . sub))
```

Then, you must put (on the *lower-case* version of `sub` only) the property `CHANGEWORD`, with value `FN`. `FN` is a function that will be applied to a single argument, the whole `sub` form, and must return a form that Chagnetran can translate into an appropriate expression. This form should be a list structure with the symbol `DATUM` used whenever you want an accessing expression for the current value of the datum to appear. The form `(DATUM← FORM)` (note that `DATUM←` is a single symbol) should occur once in the expression; this specifies that an appropriate storing expression into the datum should occur at that point. For example, `sub` could be defined as:

```
(PUTPROP 'sub 'CHANGEWORD
  '(LAMBDA (FORM)
    (LIST 'DATUM←
      (LIST 'IDIFFERENCE
        'DATUM
        (CONS 'IPLUS (CDDR FORM)))))))
```

If the expression `(sub (CAR X) A B)` were encountered, the arguments to `SUB` would first be dwimified, and then the `CHANGEWORD` function would be passed the list `(sub (CAR X) A B)`, and return `(DATUM← (IDIFFERENCE DATUM (IPLUS A B)))`, which Chagnetran would convert to `(CAR (RPLACA X (IDIFFERENCE (CAR X) (IPLUS A B))))`.

Note: The `sub` changeword as defined above will always use `IDIFFERENCE` and `IPLUS`; `add` uses the correct addition operation depending on the current CLISP declarations (see Chapter 21).

Built-In and User Data Types

Medley is a system for manipulating various kinds of data; it comes with a large set of built-in data types, which you can use to represent a variety of abstract objects; you can also define additional “user data types” that you can manipulate exactly like built-in data types.

Each data type in Medley has an associated “type name,” a symbol. Some of the type names of built-in data types are: `LITATOM`, `LISTP`, `STRINGP`, `ARRAYP`, `STACKP`, `SMALLP`, `FIXP`, and `FLOATP`. For user data types, the type name is specified when the data type is created.

(DATATYPES —) [Function]

Returns a list of all type names currently defined.

(USERDATATYPES) [Function]

Returns list of names of currently declared user data types.

(TYPENAME DATUM) [Function]

Returns the type name for the data type of `DATUM`.

RECORDS AND DATA STRUCTURES

(*TYPENAMEP DATUM TYPE*)

[Function]

Returns *T* if *DATUM* is an object with type name equal to *TYPE*, otherwise *NIL*.

In addition to built-in data-types like symbols, lists, arrays, etc., Medley provides a way to define completely *new* classes of objects, with a fixed number of fields determined by the definition of the data type. To define a new class of objects, you must supply a name for the new data type and specifications for each of its fields. Each field may contain either a pointer (i.e., any arbitrary Interlisp datum), an integer, a floating point number, or an *N*-bit integer.

Note: The most convenient way to define new user data types is via *DATATYPE* record declarations (see Chapter 8) which call the following functions.

(*DECLAREDATATYPE TYPENAME FIELDSPECS — —*)

[Function]

Defines a new user data type, with the name *TYPENAME*. *FIELDSPECS* is a list of “field specifications.” Each field specification may be one of the following:

POINTER Field may contain any Interlisp datum.

FIXP Field contains an integer.

FLOATP Field contains a floating point number.

(BITS N) Field contains a non-negative integer less than 2^N .

BYTE Equivalent to *(BITS 8)*.

WORD Equivalent to *(BITS 16)*.

SIGNEDWORD Field contains a 16 bit signed integer.

DECLAREDATATYPE returns a list of “field descriptors,” one for each element of *FIELDSPECS*. A field descriptor contains information about where within the datum the field is actually stored.

If *FIELDSPECS* is *NIL*, *TYPENAME* is “undeclared.” If *TYPENAME* is already declared as a data type, it is undeclared, and then re-declared with the new *FIELDSPECS*. An instance of a data type that has been undeclared has a type name of ***DEALLOC***.

(*FETCHFIELD DESCRIPTOR DATUM*)

[Function]

Returns the contents of the field described by *DESCRIPTOR* from *DATUM*. *DESCRIPTOR* must be a “field descriptor” as returned by *DECLAREDATATYPE* or *GETDESCRIPTORS*. If *DATUM* is not an instance of the datatype of which *DESCRIPTOR* is a descriptor, causes error *Datum of incorrect type*.

(*REPLACEFIELD DESCRIPTOR DATUM NEWVALUE*)

[Function]

Store *NEWVALUE* into the field of *DATUM* described by *DESCRIPTOR*. *DESCRIPTOR* must be a field descriptor as returned by *DECLAREDATATYPE*. If *DATUM* is not an instance of the datatype of which *DESCRIPTOR* is a descriptor, causes error *Datum of incorrect type*. Value is *NEWVALUE*.

INTERLISP-D REFERENCE MANUAL

(NCREATE TYPE OLDOBJ)

[Function]

Creates and returns a new instance of datatype *TYPE*.

If *OLDOBJ* is also a datum of datatype *TYPE*, the fields of the new object are initialized to the values of the corresponding fields in *OLDOBJ*.

NCREATE will not work for built-in datatypes, such as *ARRAYP*, *STRINGP*, etc. If *TYPE* is not the type name of a previously declared *user* data type, generates an error, *illegal data type*.

(GETFIELDSPECS TYPENAME)

[Function]

Returns a list which is *EQUAL* to the *FIELDSPECS* argument given to *DECLAREDATATYPE* for *TYPENAME*; if *TYPENAME* is not a currently declared data-type, returns *NIL*.

(GETDESCRIPTORS TYPENAME)

[Function]

Returns a list of field descriptors, *EQUAL* to the *value* of *DECLAREDATATYPE* for *TYPENAME*. If *TYPENAME* is not an atom, *(TYPENAME TYPENAME)* is used.

You can define how a user data type prints, using *DEFPRINT* (see Chapter 25), how they are to be evaluated by the interpreter via *DEFEVAL* (see Chapter 10), and how they are to be compiled by the compiler via *COMPILETYPELST* (see Chapter 18).

RECORDS AND DATA STRUCTURES

[This page intentionally left blank]

9. LISTS AND ITERATIVE STATEMENTS

Medley gives you a large number of predicates, conditional functions, and control functions. Also, there is a complex “iterative statement” facility which allows you to easily create complex loops and iterative constructs.

Data Type Predicates

Medley provides separate functions for testing whether objects are of certain commonly-used types:

(**LITATOM** *X*) [Function]

Returns T if *X* is a symbol; NIL otherwise. Note that a number is not a symbol.

(**SMALLP** *X*) [Function]

Returns *X* if *X* is a small integer; NIL otherwise. (The range of small integers is -65536 to +65535.

(**FIXP** *X*) [Function]

Returns *X* if *X* is a small or large integer; NIL otherwise.

(**FLOATP** *X*) [Function]

Returns *X* if *X* is a floating point number; NIL otherwise.

(**NUMBERP** *X*) [Function]

Returns *X* if *X* is a number of any type, NIL otherwise.

(**ATOM** *X*) [Function]

Returns T if *X* is an atom (i.e. a symbol or a number); NIL otherwise.

(**ATOM** *X*) is NIL if *X* is an array, string, etc. In Common Lisp, CL:ATOM is defined equivalent to the Interlisp function NLISTP.

(**LISTP** *X*) [Function]

Returns *X* if *X* is a list cell (something created by CONS); NIL otherwise.

(**NLISTP** *X*) [Function]

(NOT (LISTP *X*)). Returns T if *X* is not a list cell, NIL otherwise.

(**STRINGP** *X*) [Function]

Returns *X* if *X* is a string, NIL otherwise.

(**ARRAYP** *X*) [Function]

Returns *X* if *X* is an array, NIL otherwise.

INTERLISP-D REFERENCE MANUAL

(**HARRAYP** *X*) [Function]

Returns *X* if it is a hash array object; otherwise NIL.

HARRAYP returns NIL if *X* is a list whose CAR is an HARRAYP, even though this is accepted by the hash array functions.

Note: The empty list, () or NIL, is considered to be a symbol, rather than a list. Therefore, (LITATOM NIL) = (ATOM NIL) = T and (LISTP NIL) = NIL. Take care when using these functions if the object may be the empty list NIL.

Equality Predicates

Sometimes, there is more than one type of equality. For instance, given two lists, you can ask whether they are exactly the same object, or whether they are two distinct lists that contain the same elements. Confusion between these two types of equality is often the source of program errors.

(**EQ** *X Y*) [Function]

Returns T if *X* and *Y* are identical pointers; NIL otherwise. EQ should not be used to compare two numbers, unless they are small integers; use EQP instead.

(**NEQ** *X Y*) [Function]

The same as (NOT (EQ *X Y*))

(**NULL** *X*) [Function]

(**NOT** *X*) [Function]

The same as (EQ *X* NIL)

(**EQP** *X Y*) [Function]

Returns T if *X* and *Y* are EQ, or if *X* and *Y* are numbers and are equal in value; NIL otherwise. For more discussion of EQP and other number functions, see Chapter 7.

EQP also can be used to compare stack pointers (Section 11) and compiled code (Chapter 10).

(**EQUAL** *X Y*) [Function]

EQUAL returns T if *X* and *Y* are one of the following:

1. EQ
2. EQP, i.e., numbers with equal value
3. STREQUAL, i.e., strings containing the same sequence of characters
4. Lists and CAR of *X* is EQUAL to CAR of *Y*, and CDR of *X* is EQUAL to CDR of *Y*

EQUAL returns NIL otherwise. Note that EQUAL can be significantly slower than EQ.

A loose description of EQUAL might be to say that *X* and *Y* are EQUAL if they print out the same way.

CONDITIONALS AND ITERATIVE STATEMENTS

(EQUALALL X Y)

[Function]

Like EQUAL, except it descends into the contents of arrays, hash arrays, user data types, etc. Two non-EQ arrays may be EQUALALL if their respective components are EQUALALL.

Note: In general, EQUALALL descends all the way into all datatypes, both those you've defined and those built into the system. If you have a data structure with fonts and pointers to windows, EQUALALL will descend those also. If the data structures are circular, as windows are, EQUALALL can cause stack overflow.

Logical Predicates

(AND X X ... X)

[NLambda NoSpread Function]

Takes an indefinite number of arguments (including zero), that are evaluated in order. If any argument evaluates to NIL, AND immediately returns NIL, without evaluating the remaining arguments. If all of the arguments evaluate to non-NIL, the value of the last argument is returned. (AND) => T.

(OR X X ... X)

[NLambda NoSpread Function]

Takes an indefinite number of arguments (including zero), that are evaluated in order. If any argument is non-NIL, the value of that argument is returned by OR (without evaluating the remaining arguments). If all of the arguments evaluate to NIL, NIL is returned. (OR) => NIL.

AND and OR can be used as simple logical connectives, but note that they may not evaluate all of their arguments. This makes a difference if some of the arguments cause side-effects. This also means you can use AND and OR as simple conditional statements. For example: (AND (LISTP X) (CDR X)) returns the value of (CDR X) if X is a list cell; otherwise it returns NIL without evaluating (CDR X). In general, you should avoid this use of AND and OR in favor of more explicit conditional statements in order to make programs more readable.

COND Conditional Function

(COND CLAUSE CLAUSE ... CLAUSE)

[NLambda NoSpread Function]

COND takes an indefinite number of arguments, called clauses. Each CLAUSE is a list of the form (P C ... C), where P is the predicate, and C ... C are the consequents. The operation of COND can be paraphrased as:

IF P THEN C ... C ELSEIF P THEN C ... C ELSEIF P ...

The clauses are considered in sequence as follows: The predicate P of the clause CLAUSE is evaluated. If the value of P is "true" (non-NIL), the consequents C ... C are evaluated in order, and the value of the COND is the value of the last expression in the clause. If P is "false" (EQ to NIL), then the remainder of CLAUSE is ignored, and the next clause, CLAUSE , is considered. If no P is true for any clause, the value of the COND is NIL.

If a clause has no consequents, and has the form (P), then if P evaluates to non-NIL, it is returned as the value of the COND. It is only evaluated once.

INTERLISP-D REFERENCE MANUAL

Example:

```
← (DEFINEQ (DOUBLE (X)
  (COND ((NUMBERP X) (PLUS X X))
        ((STRINGP X) (CONCAT X X))
        ((ATOM X) (PACK* X X))
        (T (PRINT "unknown") X)
        (HORRIBLE-ERROR)))
  (DOUBLE)
  (DOUBLE 5)
  10
  (DOUBLE "FOO")
  "FOOFOO"
  (DOUBLE 'BAR)
  BARBAR
  (DOUBLE '(A B C))
  "unknown"
  (A B C))
```

A few points about this example: Notice that 5 is both a number and an atom, but it is “caught” by the NUMBERP clause before the ATOM clause. Also notice the predicate T, which is always true. This is the normal way to indicate a COND clause which will always be executed (if none of the preceeding clauses are true). (HORRIBLE-ERROR) will never be executed.

The IF Statement

The IF statement lets you write conditional expressions that are easier to read than using COND directly. CLISP translates expressions using IF, THEN, ELSEIF, or ELSE (or their lowercase versions) into equivalent CONDS. In general, statements of the form:

```
(if AAA then BBB elseif CCC then DDD else EEE)
```

are translated to:

```
(COND (AAA BBB)
      (CCC DDD)
      (T EEE))
```

The segment between IF or ELSEIF and the next THEN corresponds to the predicate of a COND clause, and the segment between THEN and the next ELSE or ELSEIF as the consequent(s). ELSE is the same as ELSEIF T THEN. These words are spelling corrected using the spelling list CLISPFIWORDSPLST. You may also use lower-case versions (if, then, elseif, else).

If there is nothing following a THEN, or THEN is omitted entirely, the resulting COND clause has a predicate but no consequent. For example, (if X then elseif ...) and (if X elseif ...) both translate to (COND (X) ...)—if X is not NIL, it is returned as the value of the COND.

Each predicate must be a single expression, but multiple expressions are allowed as the consequents after THEN or ELSE. Multiple consequent expressions are implicitly wrapped in a PROGN, and the value of the last one is returned as the value of the consequent. For example:

```
(if X then (PRINT "FOO") (PRINT "BAR") elseif Y then (PRINT "BAZ"))
```

CONDITIONALS AND ITERATIVE STATEMENTS

Selection Functions

(**SELECTQ** *X* *CLAUSE* *CLAUSE* ... *CLAUSE*
DEFAULT)

[NLambda NoSpread Function]

Selects a form or sequence of forms based on the value of *X*. Each clause *CLAUSE* is a list of the form (*S* *C* ... *C*) where *S* is the selection key. Think of **SELECTQ** as:

```
IF X = S THEN C ... C ELSEIF X = S
  THEN ... ELSE DEFAULT
```

If *S* is a symbol, the value of *X* is tested to see if it is EQ to *S* (which is *not* evaluated). If so, the expressions *C* ... *C* are evaluated in sequence, and the value of the **SELECTQ** is the value of the last expression.

If *S* is a list, the value of *X* is compared with each element (not evaluated) of *S*, and if *X* is EQ to any one of them, then *C* ... *C* are evaluated as above.

If *CLAUSE* is not selected in one of the two ways described, *CLAUSE* is tested, etc., until all the clauses have been tested. If none is selected, *DEFAULT* is evaluated, and its value is returned as the value of the **SELECTQ**. *DEFAULT* must be present.

An example of the form of a **SELECTQ** is:

```
[SELECTQ MONTH
  (FEBRUARY (if (LEAPYEARP) then 29 else 28))
  ((SEPTEMBER APRIL JUNE NOVEMBER) 30) 31]
```

If the value of *MONTH* is the symbol *FEBRUARY*, the **SELECTQ** returns 28 or 29 (depending on *(LEAPYEARP)*); otherwise if *MONTH* is *APRIL*, *JUNE*, *SEPTEMBER*, or *NOVEMBER*, the **SELECTQ** returns 30; otherwise it returns 31.

SELECTQ compiles open, and is therefore very fast; however, it will not work if the value of *X* is a list, a large integer, or floating point number, since **SELECTQ** uses EQ for all comparisons.

SELCHARQ (Chapter 2) is a version of **SELECTQ** that recognizes *CHARCODE* symbols.

(**SELECTC** *X* *CLAUSE* *CLAUSE* ... *CLAUSE*
DEFAULT)

[NLambda NoSpread Function]

"**SELECTQ-on-Constant.**" Like **SELECTQ**, but the selection keys are evaluated, and the result used as a **SELECTQ**-style selection key.

SELECTC is compiled as a **SELECTQ**, with the selection keys evaluated at compile-time. Therefore, the selection keys act like compile-time constants (see Chapter 18).

For example:

```
[SELECTC NUM
  ((for X from 1 to 9 collect (TIMES X X)) "SQUARE") "HIP"]
```

compiles as:

```
(SELECTQ NUM
  ((1 4 9 16 25 36 49 64 81) "SQUARE") "HIP")
```

PROG and Associated Control Functions

(**PROG1** *X X ... X*) [NLambda NoSpread Function]

Evaluates its arguments in order, and returns the value of its first argument *X*. For example, (PROG1 *X* (SETQ *X Y*)) sets *X* to *Y*, and returns *X*'s original value.

(**PROG2** *X X ... X*) [NoSpread Function]

Like PROG1. Evaluates its arguments in order, and returns the value of its second argument *X*.

(**PROGN** *X X ... X*) [NLambda NoSpread Function]

PROGN evaluates each of its arguments in order, and returns the value of its last argument. PROGN is used to specify more than one computation where the syntax allows only one, e.g., (SELECTQ ... (PROGN ...)) allows evaluation of several expressions as the default condition for a SELECTQ.

(**PROG** *VARLST E E ... E*) [NLambda NoSpread Function]

Lets you bind some variables while you execute a series of expressions. *VARLST* is a list of local variables (must be NIL if no variables are used). Each symbol in *VARLST* is treated as the name of a local variable and bound to NIL. *VARLST* can also contain lists of the form (*NAME FORM*). In this case, *NAME* is the name of the variable and is bound to the value of *FORM*. The evaluation takes place before any of the bindings are performed, e.g., (PROG ((*X Y*) (*Y X*)) ...) will bind local variable *X* to the value of *Y* (evaluated *outside* the PROG) and local variable *Y* to the value of *X* (outside the PROG). An attempt to use anything other than a symbol as a PROG variable will cause an error, Arg not symbol. An attempt to use NIL or T as a PROG variable will cause an error, Attempt to bind NIL or T.

The rest of the PROG is a sequence of forms and symbols (labels). The forms are evaluated sequentially; the labels serve only as markers. The two special functions, GO and RETURN, alter this flow of control as described below. The value of the PROG is usually specified by the function RETURN. If no RETURN is executed before the PROG "falls off the end," the value of the PROG is NIL.

(**GO** *L*) [NLambda NoSpread Function]

GO is used to cause a transfer in a PROG. (GO *L*) will cause the PROG to evaluate forms starting at the label *L* (GO does not evaluate its argument). A GO can be used at any level in a PROG. If the label is not found, GO will search higher progs *within the same function*, e.g., (PROG ... *A* ... (PROG ... (GO *A*))). If the label is not found in the function in which the PROG appears, an error is generated, Undefined or illegal GO.

(**RETURN** *X*) [Function]

A RETURN is the normal exit for a PROG. Its argument is evaluated and is immediately returned the value of the PROG in which it appears.

CONDITIONALS AND ITERATIVE STATEMENTS

Note: If a GO or RETURN is executed in an interpreted function which is not a PROG, the GO or RETURN will be executed in the last interpreted PROG entered if any, otherwise cause an error.

GO or RETURN inside of a compiled function that is not a PROG is not allowed, and will cause an error at compile time.

As a corollary, GO or RETURN in a functional argument, e.g., to SORT, will not work compiled. Also, since NLSETQ's and ERSETQ's compile as *separate* functions, a GO or RETURN *cannot* be used inside of a compiled NLSETQ or ERSETQ if the corresponding PROG is outside, i.e., above, the NLSETQ or ERSETQ.

(**LET** VARLIST E ... E) [Macro]

LET is essentially a PROG that can't contain GO's or RETURN's, and whose last form is the returned value.

(**LET*** VARLIST E ... E) [Macro]

(**PROG*** VARLIST E ... E) [Macro]

LET* and PROG* differ from LET and PROG only in that the binding of the bound variables is done "sequentially." Thus

```
(LET* ((A (LIST 5))
      (B (LIST A A)))
      (EQ A (CADR B)))
```

would evaluate to T; whereas the same form with LET might find A an unbound variable when evaluating (LIST A A).

The Iterative Statement

The various forms of the iterative statement (i.s.) let you write complex loops easily. Rather than writing PROG, MAPC, MAPCAR, etc., let Medley do it for you.

An iterative statement is a form consisting of a number of special words (known as i.s. operators or i.s.oprs), followed by operands. Many i.s.oprs (FOR, DO, WHILE, etc.) act like loops in other programming languages; others (COLLECT, JOIN, IN, etc.) do things useful in Lisp. You can also use lower-case versions of i.s.oprs (do, collect, etc.).

```
← (for X from 1 to 5 do (PRINT 'FOO))
FOO
FOO
FOO
FOO
FOO
FOO
NIL

← (for X from 2 to 10 by 2 collect (TIMES X X))
(4 16 36 64 100)

← (for X in '(A B 1 C 6.5 NIL (45)) count (NUMBERP X))
2
```

Iterative statements are implemented using CLISP, which translates them into the appropriate PROGS, MAPCARS, etc. They're translated using all CLISP declarations in effect (standard/fast/undoable/etc.); see Chapter 21. Misspelled i.s.oprs are recognized and corrected using the spelling list

INTERLISP-D REFERENCE MANUAL

CLISPFORWORDSPLST. Operators can appear in any order; CLISP scans the entire statement before it begins to translate.

If you define a function with the same name as an i.s.opr (WHILE, TO, etc.), that i.s.opr will no longer cause looping when it appears as CAR of a form, although it will continue to be treated as an i.s.opr if it appears in the interior of an iterative statement. To alert you, a warning message is printed, e.g., (While defined, therefore disabled in CLISP).

I.S. Types

Every iterative statement must have exactly one of the following operators in it (its “is.stype”), to specify what happens on each iteration. Its operand is called the “body” of the iterative statement.

DO *FORMS* [I.S. Operator]

Evaluate *FORMS* at each iteration. DO with no other operator specifies an infinite loop. If some explicit or implicit terminating condition is specified, the value of the loop is NIL. Translates to MAPC or MAP whenever possible.

COLLECT *FORM* [I.S. Operator]

The value of *FORM* at each iteration is collected in a list, which is returned as the value of the loop when it terminates. Translates to MAPCAR, MAPLIST or SUBSET whenever possible.

When COLLECT translates to a PROG (if UNTIL, WHILE, etc. appear in the loop), the translation employs an open TCONC using two pointers similar to that used by the compiler for compiling MAPCAR. To disable this translation, perform (CLDISABLE 'FCOLLECT).

JOIN *FORM* [I.S. Operator]

FORM returns a list; the lists from each iteration are concatenated using NCONC, forming one long list. Translates to MAPCONC or MAPCON whenever possible. /NCONC, /MAPCONC, and /MAPCON are used when the CLISP declaration UNDOABLE is in effect.

SUM *FORM* [I.S. Operator]

The values of *FORM* from each iteration are added together and returned as the value of the loop, e.g., (for I from 1 to 5 sum (TIMES I I)) returns 1+4+9+16+25 = 55. IPLUS, FPLUS, or PLUS will be used in the translation depending on the CLISP declarations in effect.

COUNT *FORM* [I.S. Operator]

Counts the number of times that *FORM* is true, and returns that count as the loop's value.

ALWAYS *FORM* [I.S. Operator]

Returns T if the value of *FORM* is non-NIL for all iterations. **Note:** Returns NIL as soon as the value of *FORM* is NIL).

CONDITIONALS AND ITERATIVE STATEMENTS

NEVER *FORM*

[I.S. Operator]

Like ALWAYS, but returns T if the value of *FORM* is *never* true. **Note:** Returns NIL as soon as the value of *FORM* is non-NIL.

Often, you'll want to set a variable each time through the loop; that's called the "iteration variable", or i.v. for short. The following i.s.types explicitly refer to the i.v. This is explained below under FOR.

THEREIS *FORM*

[I.S. Operator]

Returns the first value of the i.v. for which *FORM* is non-NIL, e.g., (for X in Y thereis (NUMBERP X)) returns the first number in Y.

Note: Returns the value of the i.v. as soon as the value of *FORM* is non-NIL.

LARGEST *FORM*

[I.S. Operator]

SMALLEST *FORM*

[I.S. Operator]

Returns the value of the i.v. that provides the largest/smallest value of *FORM*. \$\$EXTREME is always bound to the current greatest/smallest value, \$\$VAL to the value of the i.v. from which it came.

Iteration Variable I.s.oprs

You'll want to bind variables to use during the loop. Rather than putting the loop inside a PROG or LET, you can specify bindings like so:

BIND *VAR*

[I.S. Operator]

BIND *VARS*

[I.S. Operator]

Used to specify dummy variables, which are bound locally within the i.s.

Note: You can initialize a variable *VAR* by saying *VAR*←*FORM*:

(bind HEIGHT ← 0 WEIGHT ← 0 for SOLDIER in ...)

To specify iteration variables, use these operators:

FOR *VAR*

[I.S. Operator]

Specifies the iteration variable (i.v.) that is used in conjunction with IN, ON, FROM, TO, and BY. The variable is rebound within the loop, so the value of the variable outside the loop is not affected. Example:

```
←(SETQ X 55)
55
←(for X from 1 to 5 collect (TIMES X X))
(1 4 9 16 25)
←X
55
```

FOR *OLD VAR*

[I.S. Operator]

Like FOR, but *VAR* is *not* rebound, so its value outside the loop is changed. Example:

```
←(SETQ X 55)
55
```

INTERLISP-D REFERENCE MANUAL

```
←(for old X from 1 to 5 collect (TIMES X X))
(1 4 9 16 25)
←X
6
```

FOR VARS [I.S. Operator]

*VAR*s a list of variables, e.g., (for (X Y Z) in ...). The first variable is the i.v., the rest are dummy variables. See BIND above.

IN FORM [I.S. Operator]

FORM must evaluate to a list. The i.v. is set to successive elements of the list, one per iteration. For example, (for X in Y do ...) corresponds to (MAPC Y (FUNCTION (LAMBDA (X) ...))). If no i.v. has been specified, a dummy is supplied, e.g., (in Y collect CADR) is equivalent to (MAPCAR Y (FUNCTION CADR)).

ON FORM [I.S. Operator]

Same as IN, but the i.v. is reset to the corresponding *tail* at each iteration. Thus IN corresponds to MAPC, MAPCAR, and MAPCONC, while ON corresponds to MAP, MAPLIST, and MAPCON.

```
←(for X on ' (A B C) do (PRINT X))
(A B C)
(B C)
(C)
NIL
```

Note: For both IN and ON, *FORM* is evaluated before the main part of the i.s. is entered, i.e. *outside* of the scope of any of the bound variables of the i.s. For example, (for X bind (Y←' (1 2 3)) in Y ...) will map down the list which is the value of Y evaluated *outside* of the i.s., *not* (1 2 3).

IN OLD VAR [I.S. Operator]

Specifies that the i.s. is to iterate down *VAR*, with *VAR* itself being reset to the corresponding tail at each iteration, e.g., after (for X in old L do ... until ...) finishes, L will be some tail of its original value.

IN OLD (VAR←FORM) [I.S. Operator]

Same as IN OLD VAR, except *VAR* is first set to value of *FORM*.

ON OLD VAR [I.S. Operator]

Same as IN OLD VAR except the i.v. is reset to the current value of *VAR* at each iteration, instead of to (CAR *VAR*).

ON OLD (VAR←FORM) [I.S. Operator]

Same as ON OLD VAR, except *VAR* is first set to value of *FORM*.

CONDITIONALS AND ITERATIVE STATEMENTS

INSIDE *FORM*

[I.S. Operator]

Like **IN**, but treats first non-list, non-NIL tail as the last element of the iteration, e.g., **INSIDE** ' (A B C D . E) iterates five times with the i.v. set to E on the last iteration. **INSIDE** 'A is equivalent to **INSIDE** ' (A), which will iterate once.

FROM *FORM*

[I.S. Operator]

Specifies the initial value for a numerical i.v. The i.v. is automatically incremented by 1 after each iteration (unless **BY** is specified). If no i.v. has been specified, a dummy i.v. is supplied and initialized, e.g., (**from** 2 to 5 **collect** **SQRT**) returns (1.414 1.732 2.0 2.236).

TO *FORM*

[I.S. Operator]

Specifies the final value for a numerical i.v. If **FROM** is not specified, the i.v. is initialized to 1. If no i.v. has been specified, a dummy i.v. is supplied and initialized. If **BY** is not specified, the i.v. is automatically incremented by 1 after each iteration. When the i.v. is definitely being *incremented*, i.e., either **BY** is not specified, or its operand is a positive number, the i.s. terminates when the i.v. exceeds the value of *FORM*. Similarly, when the i.v. is definitely being decremented the i.s. terminates when the i.v. becomes *less* than the value of *FORM* (see description of **BY**).

FORM is evaluated only once, when the i.s. is first entered, and its value bound to a temporary variable against which the i.v. is checked each iteration. If the user wishes to specify an i.s. in which the value of the boundary condition is recomputed each iteration, he should use **WHILE** or **UNTIL** instead of **TO**.

When both the operands to **TO** and **FROM** are numbers, and **TO**'s operand is less than **FROM**'s operand, the i.v. is decremented by 1 after each iteration. In this case, the i.s. terminates when the i.v. becomes *less* than the value of *FORM*. For example, (**from** 10 to 1 **do** **PRINT**) prints the numbers from 10 down to 1.

BY *FORM* (without **IN** or **ON**)

[I.S. Operator]

If you aren't using **IN** or **ON**, **BY** specifies how the i.v. itself is reset at each iteration. If you're using **FROM** or **TO**, the i.v. is known to be numerical, so the new i.v. is computed by adding the value of *FORM* (which is reevaluated each iteration) to the current value of the i.v., e.g., (**for** N **from** 1 to 10 **by** 2 **collect** N) makes a list of the first five odd numbers.

If *FORM* is a positive number (*FORM* itself, not its value, which in general CLISP would have no way of knowing in advance), the loop stops when the value of the i.v. *exceeds* the value of **TO**'s operand. If *FORM* is a negative number, the loop stops when the value of the i.v. becomes *less* than **TO**'s operand, e.g., (**for** I **from** N to M **by** -2 **until** (**LESSP** I M) ...). Otherwise, the terminating condition for each iteration depends on the value of *FORM* for that iteration: if *FORM*<0, the test is whether the i.v. is less than **TO**'s operand, if *FORM*>0 the test is whether the i.v. exceeds **TO**'s operand; if *FORM* = 0, the loop terminates unconditionally.

INTERLISP-D REFERENCE MANUAL

If you didn't use FROM or TO and *FORM* is not a number, the i.v. is simply reset to the value of *FORM* after each iteration, e.g., (for I from N by (FOO) ...) sets I to the value of (FOO) on each loop after the first.

BY *FORM* (with IN or ON)

[I.S. Operator]

If you did use IN or ON, *FORM*'s value determines the *tail* for the next iteration, which in turn determines the value for the i.v. as described earlier, i.e., the new i.v. is CAR of the tail for IN, the tail itself for ON. In conjunction with IN, you can refer to the current tail within *FORM* by using the i.v. or the operand for IN/ON, e.g., (for Z in L by (CDDR Z) ...) or (for Z in L by (CDDR L) ...). At translation time, the name of the internal variable which holds the value of the current tail is substituted for the i.v. throughout *FORM*. For example, (for X in Y by (CDR (MEMB 'FOO (CDR X))) collect X) specifies that after each iteration, CDR of the current tail is to be searched for the atom FOO, and (CDR of) this latter tail to be used for the next iteration.

AS *VAR*

[I.S. Operator]

Lets you have more than one i.v. for a single loop, e.g., (for X in Y as U in V do ...) moves through the lisps Y and V in parallel (see MAP2C). The loop ends when any of the terminating conditions is met, e.g., (for X in Y as I from 1 to 10 collect X) makes a list of the first ten elements of Y, or however many elements there are on Y if less than 10.

The operand to AS, *VAR*, specifies the new i.v. For the remainder of the i.s., or until another AS is encountered, all operators refer to the new i.v. For example, (for I from 1 to N as J from 1 to N by 2 as K from N to 1 by -1 ...) terminates when I exceeds N, or J exceeds N, or K becomes less than 1. After each iteration, I is incremented by 1, J by 2, and K by -1.

OUTOF *FORM*

[I.S. Operator]

For use with generators. On each iteration, the i.v. is set to successive values returned by the generator. The loop ends when the generator runs out.

Condition I.S. Oprs

What if you want to do things only on certain times through the loop? You could make the loop body a big COND, but it's much more readable to use one of these:

WHEN *FORM*

[I.S. Operator]

Only run the loop body when *FORM*'s value is non-NIL. For example, (for X in Y collect X when (NUMBERP X)) collects only the elements of Y that are numbers.

UNLESS *FORM*

[I.S. Operator]

Opposite of WHEN: WHEN Z is the same as UNLESS (NOT Z).

WHILE *FORM*

[I.S. Operator]

WHILE *FORM* evaluates *FORM* before each iteration, and if the value is NIL, exits.

CONDITIONALS AND ITERATIVE STATEMENTS

UNTIL *FORM* [I.S. Operator]

Opposite of **WHILE**: Evaluates *FORM* *before* each iteration, and if the value is *not* **NIL**, exits.

REPEATWHILE *FORM* [I.S. Operator]

Same as **WHILE** except the test is performed *after* the loop body, but before the i.v. is reset for the next iteration.

REPEATUNTIL *FORM* [I.S. Operator]

Same as **UNTIL**, except the test is performed *after* the loop body.

Other I.S. Operators

FIRST *FORM* [I.S. Operator]

FORM is evaluated once before the first iteration, e.g., `(for X Y Z in L first (FOO Y Z) ...)`, and **FOO** could be used to initialize **Y** and **Z**.

FINALLY *FORM* [I.S. Operator]

FORM is evaluated after the loop terminates. For example, `(for X in L bind Y_0 do (if (ATOM X) then (SETQ Y (PLUS Y 1))) finally (RETURN Y))` will return the number of atoms in **L**.

EACHTIME *FORM* [I.S. Operator]

FORM is evaluated at the beginning of each iteration before, and regardless of, any testing. For example, consider,

```
(for I from 1 to N
  do (... (FOO I) ...)
  unless (... (FOO I) ...)
  until (... (FOO I) ...))
```

You might want to set a temporary variable to the value of `(FOO I)` in order to avoid computing it three times each iteration. However, without knowing the translation, you can't know whether to put the assignment in the operand to **DO**, **UNLESS**, or **UNTIL**. You can avoid this problem by simply writing `EACHTIME (SETQ J (FOO I))`.

DECLARE: *DECL* [I.S. Operator]

Inserts the form `(DECLARE DECL)` immediately following the **PROG** variable list in the translation, or, in the case that the translation is a mapping function rather than a **PROG**, immediately following the argument list of the lambda expression in the translation. This can be used to declare variables bound in the iterative statement to be compiled as local or special variables. For example `(for X in Y declare: (LOCALVARS X) ...)`. Several **DECLARE:s** can appear in the same i.s.; the declarations are inserted in the order they appear.

DECLARE *DECL* [I.S. Operator]

Same as **DECLARE:**.

INTERLISP-D REFERENCE MANUAL

Since `DECLARE` is also the name of a function, `DECLARE` cannot be used as an i.s. operator when it appears as `CAR` of a form, i.e. as the first i.s. operator in an iterative statement. However, `declare` (lowercase version) *can* be the first i.s. operator.

ORIGINAL *I.S.OPR OPERAND*

[I.S. Operator]

I.S.OPR will be translated using its original, built-in interpretation, independent of any user defined i.s. operators.

There are also a number of i.s.oprs that make it easier to create iterative statements that use the clock, looping for a given period of time. See timers, Chapter 12.

Miscellaneous Hints For Using I.S.Oprs

Lowercase versions of all i.s. operators are equivalent to the uppercase, e.g., `(for X in Y ...)` is equivalent to `(FOR X IN Y ...)`.

Each i.s. operator is of lower precedence than all Interlisp forms, so parentheses around the operands can be omitted, and will be supplied where necessary, e.g., `BIND (X Y Z)` can be written `BIND X Y Z`, `OLD (X_FORM)` as `OLD X_FORM`, etc.

`RETURN` or `GO` may be used in any operand. (In this case, the translation of the iterative statement will always be in the form of a `PROG`, never a mapping function.) `RETURN` means return from the loop (with the indicated value), *not* from the function in which the loop appears. `GO` refers to a label elsewhere in the function in which the loop appears, except for the labels `$$LP`, `$$ITERATE`, and `$$OUT` which are reserved, as described below.

In the case of `FIRST`, `FINALLY`, `EACHTIME`, `DECLARE`: or one of the i.s.types, e.g., `DO`, `COLLECT`, `SUM`, etc., the operand can consist of more than one form, e.g., `COLLECT (PRINT (CAR X)) (CDR X)`, in which case a `PROGN` is supplied.

Each operand can be the name of a function, in which case it is applied to the (last) i.v., e.g., `(for X in Y do PRINT when NUMBERP)` is the same as `(for X in Y do (PRINT X) when (NUMBERP X))`. Note that the i.v. need not be explicitly specified, e.g., `(in Y do PRINT when NUMBERP)` will work.

For i.s.types, e.g., `DO`, `COLLECT`, `JOIN`, the function is always applied to the first i.v. in the i.s., whether explicitly named or not. For example, `(in Y as I from 1 to 10 do PRINT)` prints elements on `Y`, not integers between 1 and 10.

Note that this feature does not make much sense for `FOR`, `OLD`, `BIND`, `IN`, or `ON`, since they “operate” before the loop starts, when the i.v. may not even be bound.

In the case of `BY` in conjunction with `IN`, the function is applied to the current *tail* e.g., `(for X in Y by CDDR ...)` is the same as `(for X in Y by (CDDR X) ...)`.

While the exact translation of a loop depends on which operators are present, a `PROG` will always be used whenever the loop specifies dummy variables—if `BIND` appears, or there is more than one variable specified by a `FOR`, or a `GO`, `RETURN`, or a reference to the variable `$$VAL` appears in any of the operands. When `PROG` is used, the form of the translation is:

```
(PROG VARIABLES
  {initialize})
```

CONDITIONALS AND ITERATIVE STATEMENTS

```
$$LP {eachtime}
    {test}
    {body}
$$ITERATE
    {aftertest}
    {update}
    (GO $$LP)
$$OUT {finalize}
    (RETURN $$VAL))
```

where {test} corresponds to that part of the loop that tests for termination and also for those iterations for which {body} is not going to be executed, (as indicated by a WHEN or UNLESS); {body} corresponds to the operand of the i.s.type, e.g., DO, COLLECT, etc.; {aftertest} corresponds to those tests for termination specified by REPEATWHILE or REPEATUNTIL; and {update} corresponds to that part that resets the tail, increments the counter, etc. in preparation for the next iteration. {initialize}, {finalize}, and {eachtime} correspond to the operands of FIRST, FINALLY, and EACHTIME, if any.

Since {body} always appears at the top level of the PROG, you can insert labels in {body}, and GO to them from within {body} or from other i.s. operands, e.g., (for X in Y first (GO A) do (FOO) A (FIE)). However, since {body} is dwimified as a list of forms, the label(s) should be added to the dummy variables for the iterative statement in order to prevent their being dwimified and possibly “corrected”, e.g., (for X in Y bind A first (GO A) do (FOO) A (FIE)). You can also GO to \$\$LP, \$\$ITERATE, or \$\$OUT, or explicitly set \$\$VAL.

Errors in Iterative Statements

An error will be generated and an appropriate diagnostic printed if any of the following conditions hold:

1. Operator with null operand, i.e., two adjacent operators, as in (for X in Y until do ...)
2. Operand consisting of more than one form (except as operand to FIRST, FINALLY, or one of the i.s.types), e.g., (for X in Y (PRINT X) collect ...).
3. IN, ON, FROM, TO, or BY appear twice in same i.s.
4. Both IN and ON used on same i.v.
5. FROM or TO used with IN or ON on same i.v.
6. More than one i.s.type, e.g., a DO and a SUM.

In 3, 4, or 5, an error is not generated if an intervening AS occurs.

If an error occurs, the i.s. is left unchanged.

If no DO, COLLECT, JOIN or any of the other i.s.types are specified, CLISP will first attempt to find an operand consisting of more than one form, e.g., (for X in Y (PRINT X) when ATOM X ...), and in this case will insert a DO after the first form. (In this case, condition 2 is not considered to be met, and an error is not generated.) If CLISP cannot find such an operand, and no WHILE or UNTIL appears in the i.s., a warning message is printed: NO DO, COLLECT, OR JOIN: followed by the i.s.

INTERLISP-D REFERENCE MANUAL

Similarly, if no terminating condition is detected, i.e., no IN, ON, WHILE, UNTIL, TO, or a RETURN or GO, a warning message is printed: Possible non-terminating iterative statement: followed by the iterative statement. However, since the user may be planning to terminate the i.s. via an error, Control-E, or a RETFROM from a lower function, the i.s. is still translated.

Note: The error message is not printed if the value of CLISPI . S . GAG is T (initially NIL).

Defining New Iterative Statement Operators

The following function is available for defining new or redefining existing iterative statement operators:

(I . S . OPR *NAME FORM OTHERS EVALFLG*) [Function]

NAME is the name of the new i.s.opr. If *FORM* is a list, *NAME* will be a new *i.s.type*, and *FORM* its body.

OTHERS is an (optional) list of additional i.s. operators and operands which will be added to the i.s. at the place where *NAME* appears. If *FORM* is NIL, *NAME* is a new i.s.opr defined entirely by *OTHERS*.

In both *FORM* and *OTHERS*, the atom \$\$VAL can be used to reference the value to be returned by the i.s., I . V. to reference the current i.v., and BODY to reference *NAME*'s operand. In other words, the current i.v. will be substituted for all instances of I . V. and *NAME*'s operand will be substituted for all instances of BODY throughout *FORM* and *OTHERS*.

If *EVALFLG* is T, *FORM* and *OTHERS* are evaluated at translation time, and their values used as described above. A dummy variable for use in translation that does not clash with a dummy variable already used by some other i.s. operators can be obtained by calling (GETDUMMYVAR). (GETDUMMYVAR T) will return a dummy variable and also insure that it is bound as a PROG variable in the translation.

If *NAME* was previously an i.s.opr and is being redefined, the message (*NAME* REDEFINED) will be printed (unless DFNFLG=T), and all expressions using the i.s.opr *NAME* that have been translated will have their translations discarded.

The following are some examples of how I . S . OPR could be called to define some existing i.s.oprs, and create some new ones:

```
COLLECT (I . S . OPR 'COLLECT
        ' (SETQ $$VAL (NCONC1 $$VAL BODY)))

SUM (I . S . OPR 'SUM
    ' (SETQ $$VAL (PLUS $$VAL BODY)
      ' (FIRST (SETQ $$VAL0)))

NEVER (I . S . OPR 'NEVER
    ' (if BODY then
      (SETQ $$VAL NIL) (GO $$OUT)))
```

Note: (if BODY then (RETURN NIL)) would exit from the i.s. immediately and therefore not execute the operations specified via a FINALLY (if any).

CONDITIONALS AND ITERATIVE STATEMENTS

```

THEREIS  (I.S.OPER 'THEREIS
          '(if BODY then
            (SETQ $$VAL I.V.) (GO $$OUT)))

RCOLLECT To define RCOLLECT, a version of COLLECT which uses CONS
         instead of NCONC1 and then reverses the list of values:

          (I.S.OPER 'RCOLLECT
            '(FINALLY (RETURN
                       (DREVERSE $$VAL)))]

TCOLLECT To define TCOLLECT, a version of COLLECT which uses TCONC:

          (I.S.OPER 'TCOLLECT
            '(TCONC $$VAL BODY)
            '(FIRST (SETQ $$VAL (CONS))
                   FINALLY (RETURN
                             (CAR $$VAL)))]

PRODUCT  (I.S.OPER 'PRODUCT
          '(SETQ $$VAL $$VAL*BODY)
          '(FIRST ($$VAL 1)))]

UPTO     To define UPTO, a version of TO whose operand is evaluated only
         once:

          (I.S.OPER 'UPTO
            NIL
            '(BIND $$FOO←BODY TO $$FOO)]

TO        To redefine TO so that instead of recomputing FORM each
         iteration, a variable is bound to the value of FORM, and then that
         variable is used:

          (I.S.OPER 'TO
            NIL
            '(BIND $$END FIRST
              (SETQ $$END BODY)
              ORIGINALTO $$END)]

```

Note the use of ORIGINAL to redefine TO in terms of its original definition. ORIGINAL is intended for use in redefining built-in operators, since their definitions are not accessible, and hence not directly modifiable. Thus if the operator had been defined by the user via I.S.OPER, ORIGINAL would not obtain its original definition. In this case, one presumably would simply modify the i.s.oper definition.

I.S.OPER can also be used to define synonyms for already defined i.s. operators by calling I.S.OPER with *FORM* an atom, e.g., (I.S.OPER 'WHERE 'WHEN) makes WHERE be the same as WHEN. Similarly, following (I.S.OPER 'ISTHERE 'THEREIS), one can write (ISTHERE ATOM IN Y), and following (I.S.OPER 'FIND 'FOR) and (I.S.OPER 'SUCHTHAT 'THEREIS), one can write (find X in Y suchthat X member Z). In the current system, WHERE is synonymous with WHEN, SUCHTHAT and ISTHERE with THEREIS, FIND with FOR, and THRU with TO.

INTERLISP-D REFERENCE MANUAL

If *FORM* is the atom `MODIFIER`, then *NAME* is defined as an i.s.opr which can immediately follow another i.s. operator (i.e., an error will not be generated, as described previously). *NAME* will not terminate the scope of the previous operator, and will be stripped off when `DWIMIFY` is called on its operand. `OLD` is an example of a `MODIFIER` type of operator. The `MODIFIER` feature allows the user to define i.s. operators similar to `OLD`, for use in conjunction with some other user defined i.s.opr which will produce the appropriate translation.

The file package command `I.S.OPRS` (Chapter 17) will dump the definition of i.s.oprs. (`I.S.OPRS PRODUCT UPTO`) as a file package command will print suitable expressions so that these iterative statement operators will be (re)defined when the file is loaded.

CONDITIONALS AND ITERATIVE STATEMENTS

[This page intentionally left blank]

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

Medley is designed to help you define and debug functions. Developing an applications program with Medley involves defining a number of functions in terms of the system primitives and other user-defined functions. Once defined, your functions may be used exactly like Interlisp primitive functions, so the programming process can be viewed as extending the Interlisp language to include the required functionality.

A function's definition specifies if the function has a fixed or variable number of arguments, whether these arguments are evaluated or not, the function argument names, and a series of forms which define the behavior of the function. For example:

```
(LAMBDA (X Y) (PRINT X) (PRINT Y))
```

This function has two evaluated arguments, `x` and `y`, and it will execute `(PRINT X)` and `(PRINT Y)` when evaluated. Other types of function definitions are described below.

A function is defined by putting an expr definition in the function definition cell of a symbol. There are a number of functions for accessing and setting function definition cells, but one usually defines a function with `DEFINEQ` (see the Defining Functions section below). For example:

```
← (DEFINEQ (FOO (LAMBDA (X Y) (PRINT X) (PRINT Y)))) (FOO)
```

The expression above will define the function `FOO` to have the expr definition `(LAMBDA (X Y) (PRINT X) (PRINT Y))`. After being defined, this function may be evaluated just like any system function:

```
← (FOO 3 (IPLUS 3 4))
3
7
7
```

Not all function definition cells contain expr definitions. The compiler (see the first page of Chapter 18) translates expr definitions into compiled code objects, which execute much faster. Interlisp provides a number of "function type functions" which determine how a given function is defined, the number and names of function arguments, etc. See the Function Type Functions section below.

Usually, functions are evaluated automatically when they appear within another function or when typed into Interlisp. However, sometimes it is useful to invoke the Interlisp interpreter explicitly to apply a given "functional argument" to some data. There are a number of functions which will apply a given function repeatedly. For example, `MAPCAR` will apply a function (or an expr definition) to all of the elements of a list, and return the values returned by the function:

```
← (MAPCAR '(1 2 3 4 5) '(LAMBDA (X) (ITIMES X X)))
(1 4 9 16 25)
```

When using functional arguments, there are a number of problems which can arise, related to accessing free variables from within a function argument. Many times these problems can be solved using the function `FUNCTION` to create a `FUNARG` object.

The macro facility provides another way of specifying the behavior of a function (see the Macros section below). Macros are very useful when developing code which should run very quickly, which should be compiled differently than when it is interpreted, or which should run differently in different implementations of Interlisp.

Function Types

Interlisp functions are defined using list expressions called “expr definitions.” An expr definition is a list of the form `(LAMBDA-WORD ARG-LIST FORM ... FORM)`. *LAMBDA-WORD* determines whether the arguments to this function will be evaluated or not. *ARG-LIST* determines the number and names of arguments. *FORM ... FORM* are a series of forms to be evaluated after the arguments are bound to the local variables in *ARG-LIST*.

If *LAMBDA-WORD* is the symbol `LAMBDA`, then the arguments to the function are evaluated. If *LAMBDA-WORD* is the symbol `NLAMBDA`, then the arguments to the function are not evaluated. Functions which evaluate or don’t evaluate their arguments are therefore known as “lambda” or “nlambda” functions, respectively.

If *ARG-LIST* is `NIL` or a list of symbols, this indicates a function with a fixed number of arguments. Each symbol is the name of an argument for the function defined by this expression. The process of binding these symbols to the individual arguments is called “spreading” the arguments, and the function is called a “spread” function. If the argument list is any symbol other than `NIL`, this indicates a function with a variable number of arguments, known as a “nospread” function.

If *ARG-LIST* is anything other than a symbol or a list of symbols, such as `(LAMBDA "FOO" ...)`, attempting to use this expr definition will generate an `Arg not symbol` error. In addition, if `NIL` or `T` is used as an argument name, the error `Attempt to bind NIL or T` is generated.

These two parameters (lambda/nlambda and spread/nospread) may be specified independently, so there are four main function types, known as lambda-spread, nlambda-spread, lambda-nospread, and nlambda-nospread functions. Each one has a different form and is used for a different purpose. These four function types are described more fully below.

For lambda-spread, lambda-nospread, or nlambda-spread functions, there is an upper limit to the number of arguments that a function can have, based on the number of arguments that can be stored on the stack on any one function call. Currently, the limit is 80 arguments. If a function is called with more than that many arguments, the error `Too many arguments` occurs. However, nlambda-nospread functions can be called with an arbitrary number of arguments, since the arguments are not individually saved on the stack.

Lambda-Spread Functions

Lambda-spread functions take a fixed number of evaluated arguments. This is the most common function type. A lambda-spread expr definition has the form:

```
(LAMBDA (ARG ... ARG) FORM ... FORM)
```

The argument list `(ARG ... ARG)` is a list of symbols that gives the number and names of the formal arguments to the function. If the argument list is `()` or `NIL`, this indicates that the function takes no arguments. When a lambda-spread function is applied to some arguments, the arguments are evaluated, and bound to the local variables `ARG ... ARG`. Then, `FORM ... FORM` are evaluated in order, and the value of the function is the value of `FORM`.

```
← (DEFINEQ (FOO (LAMBDA (X Y) (LIST X Y))))
  (FOO)
← (FOO 99 (PLUS 3 4))
  (99 7)
```

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

In the above example, the function `FOO` defined by `(LAMBDA (X Y) (LIST X Y))` is applied to the arguments `99` and `(PLUS 3 4)`. These arguments are evaluated (giving `99` and `7`), the local variable `x` is bound to `99` and `y` to `7`, `(LIST X Y)` is evaluated, returning `(99 7)`, and this is returned as the value of the function.

A standard feature of the Interlisp system is that no error occurs if a spread function is called with too many or too few arguments. If a function is called with too many arguments, the extra arguments are evaluated but ignored. If a function is called with too few arguments, the unsupplied ones will be delivered as `NIL`. In fact, a spread function cannot distinguish between being given `NIL` as an argument, and not being given that argument, e.g., `(FOO)` and `(FOO NIL)` are exactly the same for spread functions. If it is necessary to distinguish between these two cases, use an `nlambda` function and explicitly evaluate the arguments with the `eval` function.

Nlambda-Spread Functions

Nlambda-spread functions take a fixed number of unevaluated arguments. An `nlambda-spread` expr definition has the form:

```
(NLAMBDA (ARG ... ARG) FORM ... FORM)
```

Nlambda-spread functions are evaluated similarly to `lambda-spread` functions, except that the arguments are not evaluated before being bound to the variables `ARG ... ARG`.

```
← (DEFINEQ (FOO (NLAMBDA (X Y) (LIST X Y))))  
  (FOO)  
← (FOO 99 (PLUS 3 4))  
  (99 (PLUS 3 4))
```

In the above example, the function `FOO` defined by `(NLAMBDA (X Y) (LIST X Y))` is applied to the arguments `99` and `(PLUS 3 4)`. These arguments are unevaluated to `x` and `y`. `(LIST X Y)` is evaluated, returning `(99 (PLUS 3 4))`, and this is returned as the value of the function.

Functions can be defined so that all of their arguments are evaluated (`lambda` functions) or none are evaluated (`nlambda` functions). If it is desirable to write a function which only evaluates some of its arguments (e.g., `SETQ`), the functions should be defined as an `nlambda`, with some arguments explicitly evaluated using the function `eval`. If this is done, the user should put the symbol `eval` on the property list of the function under the property `INFO`. This informs various system packages, such as `DWIM`, `CLISP`, and `Masterscope`, that this function in fact does evaluate its arguments, even though it is an `nlambda`.

Warning: A frequent problem that occurs when evaluating arguments to `nlambda` functions with `eval` is that the form being evaluated may reference variables that are not accessible within the `nlambda` function. This is usually not a problem when interpreting code, but when the code is compiled, the values of “local” variables may not be accessible on the stack (see Chapter 18). The system `nlambda` functions that evaluate their arguments (such as `SETQ`) are expanded in-line by the compiler, so this is not a problem. Using the macro facility is recommended in cases where it is necessary to evaluate some arguments to an `nlambda` function.

Lambda-Nospread Functions

Lambda-nospread functions take a variable number of evaluated arguments. A `lambda-nospread` expr definition has the form:

```
(LAMBDA VAR FORM ... FORM)
```

INTERLISP-D REFERENCE MANUAL

VAR may be any symbol, except *NIL* and *T*. When a lambda-nospread function is applied to some arguments, each of these arguments is evaluated and the values stored on the stack. *VAR* is then bound to the number of arguments which have been evaluated. For example, if *FOO* is defined by (LAMBDA *X* ...), when (FOO *A B C*) is evaluated, *A*, *B*, and *C* are evaluated and *x* is bound to 3. *VAR* should never be reset

The following functions are used for accessing the arguments of lambda-nospread functions.

(**ARG** *VAR M*) [NLambda Function]

Returns the *M*th argument for the lambda-nospread function whose argument list is *VAR*. *VAR* is the name of the atomic argument list to a lambda-nospread function, and is not evaluated. *M* is the number of the desired argument, and is evaluated. The value of *ARG* is undefined for *M* less than or equal to 0 or greater than the value of *VAR*.

(**SETARG** *VAR M X*) [NLambda Function]

Sets the *M*th argument for the lambda-nospread function whose argument list is *VAR* to *X*. *VAR* is not evaluated; *M* and *X* are evaluated. *M* should be between 1 and the value of *VAR*.

In the example below, the function *FOO* is defined to collect and return a list of all of the evaluated arguments it is given (the value of the *for* statement).

```
← (DEFINEQ (FOO
  (LAMBDA X (for ARGNUM from 1 to X collect (ARG X ARGNUM))
  (FOO))
← (FOO 99 (PLUS 3 4))
  (99 7)
← (FOO 99 (PLUS 3 4) (TIMES 3 4))
  (99 7 12)
```

NLambda-Nospread Functions

Nlambda-nospread functions take a variable number of unevaluated arguments. An nlambda-nospread *expr* definition has the form:

(NLAMBDA *VAR FORM* ... *FORM*)

VAR may be any symbol, except *NIL* and *T*. Though similar in form to lambda-nospread *expr* definitions, an nlambda-nospread is evaluated quite differently. When an nlambda-nospread function is applied to some arguments, *VAR* is simply bound to a list of the unevaluated arguments. The user may pick apart this list, and evaluate different arguments.

In the example below, *FOO* is defined to return the reverse of the list of arguments it is given (unevaluated):

```
← (DEFINEQ (FOO (NLAMBDA X (REVERSE X))))
  (FOO)
← (FOO 99 (PLUS 3 4))
  ((PLUS 3 4) 99)
← (FOO 99 (PLUS 3 4) (TIMES 3 4))
  (TIMES 3 4) (PLUS 3 4) 99)
```

The warning about evaluating arguments to nlambda functions also applies to nlambda-nospread function.

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

Compiled Functions

Functions defined by expr definitions can be compiled by the Interlisp compiler (see Chapter 18). The compiler produces compiled code objects (of data type `CCODEP`) which execute more quickly than the corresponding expr definition code. Functions defined by compiled code objects may have the same four types as expr definitions (lambda/nlambda, spread/nospread). Functions created by the compiler are referred to as compiled functions.

Function Type Functions

There are a variety of functions used for examining the type, argument list, etc. of functions. These functions may be given either a symbol (in which case they obtain the function definition from the definition cell), or a function definition itself.

(FNTYP FN) [Function]

Returns `NIL` if *FN* is not a function definition or the name of a defined function. Otherwise, *FNTYP* returns one of the following symbols, depending on the type of function definition.

`EXPR` Lambda-spread expr definition
`CEXP` Lambda-spread compiled definition
`FEXPR` Nlambda-spread expr definition
`CFEXPR` Nlambda-spread compiled definition
`EXPR*` Lambda-nospread expr definition
`CEXP*` Lambda-nospread compiled definition
`FEXPR*` Nlambda-nospread expr definition
`CFEXPR*` Nlambda-nospread compiled definition
`FUNARG` *FNTYP* returns the symbol *FUNARG* if *FN* is a *FUNARG* expression.

`EXP`, `FEXPR`, `EXPR*`, and `FEXPR*` indicate that *FN* is defined by an expr definition. `CEXP`, `CFEXPR`, `CEXP*`, and `CFEXPR*` indicate that *FN* is defined by a compiled definition, as indicated by the prefix `c`. The suffix `*` indicates that *FN* has an indefinite number of arguments, i.e., is a nospread function. The prefix `f` indicates unevaluated arguments. Thus, for example, a `CFEXPR*` is a compiled nospread nlambda function.

(EXPRP FN) [Function]

Returns `T` if *(FNTYP FN)* is `EXPR`, `FEXPR`, `EXPR*`, or `FEXPR*`; `NIL` otherwise. However, *(EXPRP FN)* is also true if *FN* is (has) a list definition, even if it does not begin with `LAMBDA` or `NLAMBDA`. In other words, *EXPRP* is not quite as selective as *FNTYP*.

(CCODEP FN) [Function]

Returns `T` if *(FNTYP FN)* is either `CEXP`, `CFEXPR`, `CEXP*`, or `CFEXPR*`; `NIL` otherwise.

(ARGTYPE FN) [Function]

FN is the name of a function or its definition. *ARGTYPE* returns 0, 1, 2, or 3, or `NIL` if *FN* is not a function. *ARGTYPE* corresponds to the rows of *FNTYP*s. The interpretation of this value is as follows:

- 0 Lambda-spread function (`EXPR`, `CEXP`)
- 1 Nlambda-spread function (`FEXPR`, `CFEXPR`)
- 2 Lambda-nospread function (`EXPR*`, `CEXP*`)

INTERLISP-D REFERENCE MANUAL

3 Nlambda-nospread function (FEXPR*, CFEXPR*)

(**NARGS** *FN*) [Function]

Returns the number of arguments of *FN*, or `NIL` if *FN* is not a function. If *FN* is a nospread function, the value of `NARGS` is 1.

(**ARGLIST** *FN*) [Function]

Returns the “argument list” for *FN*. Note that the “argument list” is a symbol for nospread functions. Since `NIL` is a possible value for `ARGLIST`, the error `Args not available` is generated if *FN* is not a function.

If *FN* is a compiled function, the argument list is constructed, i.e., each call to `ARGLIST` requires making a new list. For functions defined by expr definitions, lists beginning with `LAMBDA` or `NLAMBDA`, the argument list is simply `CADR` of `GETD`. If *FN* has an expr definition, and `CAR` of the definition is not `LAMBDA` or `NLAMBDA`, `ARGLIST` will check to see if `CAR` of the definition is a member of `LAMBDA$PLST` (see Chapter 20). If it is, `ARGLIST` presumes this is a function object the user is defining via `DWIMUSERFORMS`, and simply returns `CADR` of the definition as its argument list. Otherwise `ARGLIST` generates an error as described above.

(**SMARTARGLIST** *FN EXPLAINFLG TAIL*) [Function]

A “smart” version of `ARGLIST` that tries various strategies to get the arglist of *FN*.

First `SMARTARGLIST` checks the property list of *FN* under the property `ARGNAMES`. For spread functions, the argument list itself is stored. For nospread functions, the form is `(NIL ARGLIST . ARGLIST)`, where `ARGLIST` is the value `SMARTARGLIST` should return when `EXPLAINFLG` = `T`, and `ARGLIST` the value when `EXPLAINFLG` = `NIL`. For example, `(GETPROP 'DEFINEQ 'ARGNAMES) = (NIL (X1 X1 ... XN) . X)`. This allows the user to specify special argument lists.

Second, if *FN* is not defined as a function, `SMARTARGLIST` attempts spelling correction on *FN* by calling `FNCHECK` (see Chapter 20), passing *TAIL* to be used for the call to `FIXSPELL`. If unsuccessful, the `FN Not a function` error will be generated.

Third, if *FN* is known to the file package (see Chapter 17) but not loaded in, `SMARTARGLIST` will obtain the arglist information from the file.

Otherwise, `SMARTARGLIST` simply returns `(ARGLIST FN)`.

`SMARTARGLIST` is used by `BREAK` (see Chapter 15) and `ADVISE` with `EXPLAINFLG` = `NIL` for constructing equivalent expr definitions, and by the `TTYIN` in-line command `?=` (see Chapter 26), with `EXPLAINFLG` = `T`.

Defining Functions

Function definitions are stored in a “function definition cell” associated with each symbol. This cell is directly accessible via the two functions `PUTD` and `GETD` (see below), but it is usually easier to define functions with `DEFINEQ`:

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

(DEFINEQ X X ... X)

[NLambda NoSpread Function]

DEFINEQ is the function normally used for defining functions. It takes an indefinite number of arguments which are not evaluated. Each *X* must be a list defining one function, of the form (NAME DEFINITION). For example:

```
(DEFINEQ (DOUBLE (LAMBDA (X) (IPLUS X X))))
```

The above expression will define the function **DOUBLE** with the expr definition (LAMBDA (X) (IPLUS X X)). *X* may also have the form (NAME ARGS . DEF-BODY), in which case an appropriate lambda expr definition will be constructed. Therefore, the above expression is exactly the same as:

```
(DEFINEQ (DOUBLE (X) (IPLUS X X)))
```

Note that this alternate form can only be used for lambda functions. The first form must be used to define an nlambda function.

DEFINEQ returns a list of the names of the functions defined.

(DEFINE X →)

[Function]

Lambda-spread version of **DEFINEQ**. Each element of the list *X* is itself a list either of the form (NAME DEFINITION) or (NAME ARGS . DEF-BODY). **DEFINE** will generate an error, Incorrect defining form on encountering an atom where a defining list is expected.

DEFINE and **DEFINEQ** operate correctly if the function is already defined and **BROKEN**, **ADVISED**, or **BROKEN-IN**.

For expressions involving type-in only, if the time stamp facility is enabled (see the Time Stamps section of Chapter 16), both **DEFINE** and **DEFINEQ** stamp the definition with your initials and date.

UNSAFE.TO.MODIFY.FNS

[Variable]

Value is a list of functions that should not be redefined, because doing so may cause unusual bugs (or crash the system!). If you try to modify a function on this list (using **DEFINEQ**, **TRACE**, etc), the system prints Warning: XXX may be unsafe to modify -- continue? If you type **Yes**, the function is modified, otherwise an error occurs. This provides a measure of safety for novices who may accidentally redefine important system functions. You can add your own functions onto this list.

By convention, all functions starting with the character backslash (“\”) are system internal functions, which you should never redefine or modify. Backslash functions are not on **UNSAFE.TO.MODIFY.FNS**, so trying to redefine them will not cause a warning.

DFNFLG

[Variable]

DFNFLG is a global variable that affects the operation of **DEFINEQ** and **DEFINE**. If **DFNFLG**=NIL, an attempt to *redefine* a function *FN* will cause **DEFINE** to print the message (FN REDEFINED) and to save the old definition of *FN* using **SAVEDEF** (see the Functions for Manipulating Typed Definitions section of Chapter 17) before redefining it (except if the old and new definitions are **EQUAL**, in which case the effect is simply a no-op). If **DFNFLG**=T, the function is simply redefined. If **DFNFLG**=PROP or **ALLPROP**, the new definition is stored on the property list under the property *EXPR*. **ALLPROP** also affects the operation of **RPAQQ** and **RPAQ** (see the Functions Used Within Source Files section of Chapter 17). **DFNFLG** is initially NIL.

INTERLISP-D REFERENCE MANUAL

`DFNFLG` is reset by `LOAD` (see the Loading Files section of Chapter 17) to enable various ways of handling the defining of functions and setting of variables when loading a file. For most applications, the user will not reset `DFNFLG` directly.

Note: The compiler does *not* respect the value of `DFNFLG` when it redefines functions to their compiled definitions (see the first page of Chapter 18). Therefore, if you set `DFNFLG` to `PROP` to completely avoid inadvertently redefining something in your running system, you *must* use compile mode `F`, not `ST`.

Note that the functions `SAVEDEF` and `UNSAVEDEF` (see the Functions for Manipulating Typed Definitions section of Chapter 17) can be useful for “saving” and restoring function definitions from property lists.

(**GETD** *FN*) [Function]

Returns the function definition of *FN*. Returns `NIL` if *FN* is not a symbol, or has no definition.

`GETD` of a compiled function constructs a pointer to the definition, with the result that two successive calls do not necessarily produce `EQ` results. `EQP` or `EQUAL` must be used to compare compiled definitions.

(**PUTD** *FN DEF* →) [Function]

Puts *DEF* into *FN*'s function cell, and returns *DEF*. Generates an error, `Arg not symbol`, if *FN* is not a symbol. Generates an error, `illegal arg`, if *DEF* is a string, number, or a symbol other than `NIL`.

(**MOVED** *FROM TO COPYFLG* →) [Function]

Moves the definition of *FROM* to *TO*, i.e., redefines *TO*. If *COPYFLG* = `T`, a `COPY` of the definition of *FROM* is used. *COPYFLG* = `T` is only meaningful for `expr` definitions, although `MOVED` works for compiled functions as well. `MOVED` returns *TO*.

`COPYDEF` (see the Functions for Manipulating Typed Definitions section of Chapter 17) is a higher-level function that not only moves `expr` definitions, but works also for variables, records, etc.

(**MOVED?** *FROM TO COPYFLG* →) [Function]

If *TO* is not defined, same as `(MOVED FROM TO COPYFLG)`. Otherwise, does nothing and returns `NIL`.

Function Evaluation

Usually, function application is done automatically by the Interlisp interpreter. If a form is typed into Interlisp whose `CAR` is a function, this function is applied to the arguments in the `CDR` of the form. These arguments are evaluated or not, and bound to the function parameters, as determined by the type of the function, and the body of the function is evaluated. This sequence is repeated as each form in the body of the function is evaluated.

There are some situations where it is necessary to explicitly call the evaluator, and Interlisp supplies a number of functions that will do this. These functions take “functional arguments,” which may either

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

be symbols with function definitions, or expr definition forms such as `(LAMBDA (X) ...)`, or `FUNARG` expressions.

(APPLY FN ARGLIST →) [Function]

Applies the function *FN* to the arguments in the list *ARGLIST*, and returns its value. `APPLY` is a lambda function, so its arguments are evaluated, but the individual elements of *ARGLIST* are not evaluated. Therefore, lambda and nlambda functions are treated the same by `APPLY`—lambda functions take their arguments from *ARGLIST* without evaluating them. For example:

```
← (APPLY 'APPEND ' ((PLUS 1 2 3) (4 5 6)))
  (PLUS 1 2 3 4 5 6)
```

Note that *FN* may explicitly evaluate one or more of its arguments itself. For example, the system function `SETQ` is an nlambda function that explicitly evaluates its second argument. Therefore, `(APPLY 'SETQ ' (FOO (ADD1 3)))` will set `FOO` to 4, instead of setting it to the expression `(ADD1 3)`.

`APPLY` can be used for manipulating expr definitions. For example:

```
← (APPLY ' (LAMBDA (X Y) (ITIMES X Y)) ' (3 4))
  12
```

(APPLY* FN ARG ARG ... ARG) [NoSpread Function]

Nospread version of `APPLY`. Applies the function *FN* to the arguments *ARG ARG ... ARG*. For example:

```
← (APPLY 'APPEND ' (PLUS 1 2 3) (4 5 6))
  (PLUS 1 2 3 4 5 6)
```

(EVAL X→) [Function]

`EVAL` evaluates the expression *X* and returns this value, i.e., `EVAL` provides a way of calling the Interlisp interpreter. Note that `EVAL` is itself a lambda function, so its argument is first evaluated, e.g.:

```
← (SETQ FOO 'ADD1 3))
  (ADD1 3)
← (EVAL FOO)
  4
← (EVAL 'FOO)
  (ADD1 3)
```

(QUOTE X) [Nlambda NoSpread Function]

`QUOTE` prevents its arguments from being evaluated. Its value is *X* itself, e.g., `(QUOTE FOO)` is `FOO`.

Interlisp functions can either evaluate or not evaluate their arguments. `QUOTE` can be used in those cases where it is desirable to specify arguments unevaluated.

The single-quote character `(')` is defined with a read macro so it returns the next expression, wrapped in a call to `QUOTE` (see Chapter 25). For example, `'FOO` reads as `(QUOTE FOO)`. This is the form used for examples in this manual.

Since giving `QUOTE` more than one argument is almost always a parentheses error, and one that would otherwise go undetected, `QUOTE` itself generates an error in this case, `Parenthesis error`.

INTERLISP-D REFERENCE MANUAL

(KWOTE X)

[Function]

Value is an expression which, when evaluated, yields *X*. If *X* is `NIL` or a number, this is *X* itself. Otherwise `(LIST (QUOTE QUOTE) X)`. For example:

```
(KWOTE 5) => 5
(KWOTE (CONS 'A 'B)) => (QUOTE (A.B))
```

(NLAMBDA.ARGS X)

[Function]

This function interprets its argument as a list of unevaluated `nlambda` arguments. If any of the elements in this list are of the form `(QUOTE...)`, the enclosing `QUOTE` is stripped off. Actually, `NLAMBDA.ARGS` stops processing the list after the first non-quoted argument. Therefore, whereas `(NLAMBDA.ARGS '((QUOTE FOO) BAR)) -> (FOO BAR)`, `(NLAMBDA.ARGS '(FOO (QUOTE BAR))) -> (FOO (QUOTE BAR))`.

`NLAMBDA.ARGS` is called by a number of `nlambda` functions in the system, to interpret their arguments. For instance, the function `BREAK` calls `NLAMBDA.ARGS` so that `(BREAK 'FOO)` will break the function `FOO`, rather than the function `QUOTE`.

(EVALA X A)

[Function]

Simulates association list variable lookup. *X* is a form, *A* is a list of the form:

```
((NAME . VAL) (NAME . VAL) ... (NAME . VAL))
```

The variable names and values in *A* are “spread” on the stack, and then *X* is evaluated. Therefore, any variables appearing free in *X* that also appears as `CAR` of an element of *A* will be given the value on the `CDR` of that element.

(DEFEVAL TYPE FN)

[Function]

Specifies how a datum of a particular type is to be evaluated. Intended primarily for user-defined data types, but works for all data types except lists, literal atoms, and numbers. *TYPE* is a type name. *FN* is a function object, i.e., name of a function or a lambda expression. Whenever the interpreter encounters a datum of the indicated type, *FN* is applied to the datum and its value returned as the result of the evaluation. `DEFEVAL` returns the previous evaluating function for this type. If *FN* = `NIL`, `DEFEVAL` returns the current evaluating function without changing it. If *FN* = `T`, the evaluating functions is set back to the system default (which for all data types except lists is to return the datum itself).

`COMPILETYPELST` (see Chapter 18) permits the user to specify how a datum of a particular type is to be compiled.

(EVALHOOK FORM EVALHOOKFN)

[Function]

`EVALHOOK` evaluates the expression *FORM*, and returns its value. While evaluating *FORM*, the function `EVAL` behaves in a special way. Whenever a list other than *FORM* itself is to be evaluated, whether implicitly or via an explicit call to `EVAL`, *EVALHOOKFN* is invoked (it should be a function), with the form to be evaluated as its argument. *EVALHOOKFN* is then responsible for evaluating the form. Whatever is returned is assumed to be the result of evaluating the form. During the execution of *EVALHOOKFN*, this special evaluation is turned off. (Note that `EVALHOOK` does not affect the evaluations of variables, only of lists).

Here is an example of a simple tracing routine that uses the `EVALHOOK` feature:

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

```
← (DEFINQ (PRINTHOOK (FORM)
  (printout T "eval: "FORM T)
  (EVALHOOK FORM (FUNCTION PRINTHOOK
    (PRINTHOOK))
```

Using PRINTHOOK, one might see the following interaction:

```
← (EVALHOOK ' (LIST (CONS 1 2) (CONS 3 4)) 'PRINTHOOK)
eval: (CONS 1 2)
eval: (CONS 3 4)
((1.2) (3.4))
```

Iterating and Mapping Functions

The functions below are used to evaluate a form or apply a function repeatedly. `RPT`, `RPTQ`, and `FRPTQ` evaluate an expression a specified number of time. `MAP`, `MAPCAR`, `MAPLIST`, etc., apply a given function repeatedly to different elements of a list, possibly constructing another list.

These functions allow efficient iterative computations, but they are difficult to use. For programming iterative computations, it is usually better to use the CLISP Iterative Statement facility (see Chapter 9), which provides a more general and complete facility for expressing iterative statements. Whenever possible, CLISP translates iterative statements into expressions using the functions below, so there is no efficiency loss.

(RPT *N FORM*) [Function]

Evaluates the expression *FORM*, *N* times. Returns the value of the last evaluation. If *N* is less than or equal to 0, *FORM* is not evaluated, and `RPT` returns `NIL`.

Before each evaluation, the local variable `RPTN` is bound to the number of evaluations yet to take place. This variable can be referenced within *FORM*. For example, `(RPT 10 '(PRINT RPTN))` will print the numbers 10, 9...1, and return 1.

(RPTQ *N FORM FORM ... FORM*) [NLambda NoSpread Function]

Nlambda-nospread version of `RPT`: *N* is evaluated, *FORM* are not. Returns the value of the last evaluation of *FORM*.

(FRPTQ *N FORM FORM ... FORM*) [NLambda NoSpread Function]

Faster version of `RPTQ`. Does not bind `RPTN`.

(MAP MAP MAPFN MAPFN) [Function]

If *MAPFN* is `NIL`, `MAP` applies the function *MAPFN* to successive tails of the list *MAP*. That is, first it computes `(MAPFN MAP)`, and then `(MAPFN (CDR MAP))`, etc., until *MAP* becomes a non-list. If *MAPFN* is provided, `(MAPFN MAP)` is used instead of `(CDR MAP)` for the next call for *MAPFN*, e.g., if *MAPFN* were `CDDR`, alternate elements of the list would be skipped. `MAP` returns `NIL`.

(MAPC MAP MAPFN MAPFN) [Function]

Identical to `MAP`, except that `(MAPFN (CAR MAP))` is computed at each iteration instead of `(MAPFN MAP)`, i.e., `MAPC` works on elements, `MAP` on tails. `MAPC` returns `NIL`.

INTERLISP-D REFERENCE MANUAL

(MAPLIST MAP MAPFN MAPFN) [Function]

Successively computes the same values that `MAP` would compute, and returns a list consisting of those values.

(MAPCAR MAP MAPFN MAPFN) [Function]

Computes the same values that `MAPC` would compute, and returns a list consisting of those values, e.g., `(MAPCAR X 'FNTYP)` is a list of `FNTYPS` for each element on `x`.

(MAPCON MAP MAPFN MAPFN) [Function]

Computes the same values that `MAP` and `MAPLIST` but `NCONCS` these values to form a list which it returns.

(MAPCONC MAP MAPFN MAPFN) [Function]

Computes the same values that `MAPC` and `MAPCAR`, but `NCONCS` the values to form a list which it returns.

Note that `MAPCAR` creates a new list which is a mapping of the old list in that each element of the new list is the result of applying a function to the corresponding element on the original list. `MAPCONC` is used when there are a variable number of elements (including none) to be inserted at each iteration. Examples:

```
(MAPCONC '(A B C NIL D NIL) '(LAMBDA (Y) (if (NULL Y) then NIL
else (LIST Y)))) => (A B C D)
```

This `MAPCONC` returns a list consisting of `MAP` with all `NILs` removed.

```
(MAPCONC '((A B) C (D E F) (G) H I) '(LAMBDA (Y) (if (LISTP Y) then Y
else NIL))) => (A B D E F G)
```

This `MAPCONC` returns a linear list consisting of all the lists on `MAP`.

Since `MAPCONC` uses `NCONC` to string the corresponding lists together, in this example the original list will be altered to be `((A B C D E F G) C (D E F G) (G) H I)`. If this is an undesirable side effect, the functional argument to `MAPCONC` should return instead a top level copy of the lists, i.e., `(LAMBDA (Y) (if (LISTP Y) then (APPEND Y) else NIL))`.

(MAP2C MAP MAP MAPFN MAPFN) [Function]

Identical to `MAPC` except `MAPFN` is a function of two arguments, and `(MAPFN (CAR MAP) (CAR MAP))` is computed at each iteration. Terminates when either `MAP` or `MAP` is a non-list.

`MAPFN` is still a function of one argument, and is applied twice on each iteration; `(MAPFN MAP)` gives the new `MAP`, `(MAPFN MAP)` the new `MAP`. `CDR` is used if `MAPFN` is not supplied, i.e., is `NIL`.

(MAP2CAR MAP MAP MAPFN MAPFN) [Function]

Identical to `MAPCAR` except `MAPFN` is a function of two arguments, and `(MAPFN (CAR MAP) (CAR MAP))` is used to assemble the new list. Terminates when either `MAP` or `MAP` is a non-list.

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

(SUBSET MAP MAPFN MAPFN)

[Function]

Applies *MAPFN* to elements of *MAP* and returns a list of those elements for which this application is non-NIL, e.g.:

```
(SUBSET '(A B 3 C 4) 'NUMBERP) => (3 4)
```

MAPFN plays the same role as with *MAP*, *MAPC*, et al.

(EVERY EVERY EVERYFN EVERYFN)

[Function]

Returns T if the result of applying *EVERYFN* to each element in *EVERY* is true, otherwise NIL. For example, (EVERY '(X Y Z) 'ATOM) => T.

EVERY operates by evaluating (*EVERYFN* (CAR *EVERY*) *EVERY*). The second argument is passed to *EVERYFN* so that it can look at the next element on *EVERY* if necessary. If *EVERYFN* yields NIL, *EVERY* immediately returns NIL. Otherwise, *EVERY* computes (*EVERYFN* *EVERY*), or (CDR *EVERY*) if *EVERYFN* = NIL, and uses this as the “new” *EVERY*, and the process continues. For example (EVERY X 'ATOM 'CDDR) is true if every other element of X is atomic.

(SOME SOME SOMEFN SOMEFN)

[Function]

Returns the tail of *SOME* beginning with the first element that satisfies *SOMEFN*, i.e., for which *SOMEFN* applied to that element is true. Value is NIL if no such element exists.

(SOME X '(LAMBDA (Z) (EQUAL Z Y))) is equivalent to (MEMBER Y X). *SOME* operates analogously to *EVERY*. At each stage, (*SOMEFN* (CAR *SOME*) *SOME*) is computed, and if this not NIL, *SOME* is returned as the value of *SOME*. Otherwise, (*SOMEFN* *SOME*) is computed, or (CDR *SOME*) if *SOMEFN* = NIL, and used for the next *SOME*.

(NOTANY SOME SOMEFN SOMEFN)

[Function]

```
(NOT (SOME SOME SOMEFN SOMEFN)).
```

(NOTEVERY EVERY EVERYFN EVERYFN)

[Function]

```
(NOT (EVERY EVERY EVERYFN EVERYFN)).
```

(MAPRINT LST FILE LEFT RIGHT SEP PFN LISPXPRINTFLG)

[Function]

A general printing function. For each element of the list *LST*, applies *PFN* to the element, and *FILE*. If *PFN* is NIL, *PRIN1* is used. Between each application *MAPRINT* performs *PRIN1* of *SEP* (or "" if *SEP* = NIL). If *LEFT* is given, it is printed (using *PRIN1*) initially; if *RIGHT* is given, it is printed (using *PRIN1*) at the end.

For example, (MAPRINT X NIL '%('%)) is equivalent to *PRIN1* for lists. To print a list with commas between each element and a final “.” one could use (MAPRINT X T NIL '%. '%.).

If *LISPXPRINTFLG* = T, *LISPXPRIN1* (see Chapter 13) is used instead of *PRIN1*.

Functional Arguments

The functions that call the Interlisp-D evaluator take “functional arguments,” which may be symbols with function definitions, or expr definition forms such as (LAMBDA (X) ...).

INTERLISP-D REFERENCE MANUAL

The following functions are useful when one wants to supply a functional argument which will always return `NIL`, `T`, or `0`. Note that the arguments `X ... X` to these functions are evaluated, though they are not used.

`(NIL X ... X)` [NoSpread Function]

Returns `NIL`.

`(TRUE X ... X)` [NoSpread Function]

Returns `T`.

`(ZERO X ... X)` [NoSpread Function]

Returns `0`.

When using `expr` definitions as function arguments, they should be enclosed within the function `FUNCTION` rather than `QUOTE`, so that they will be compiled as separate functions.

`(FUNCTION FN ENV)` [NLambda Function]

If `ENV` = `NIL`, `FUNCTION` is the same as `QUOTE`, except that it is treated differently when compiled. Consider the function definition:

```
(DEFINEQ (FOO (LST) (FIE LST (FUNCTION (LAMBDA (Z) (ITIMES Z Z)))))
```

`FOO` calls the function `FIE` with the value of `LST` and the `expr` definition `(LAMBDA (Z) (LIST (CAR Z)))`.

If `FOO` is run interpreted, it does not make any difference whether `FUNCTION` or `QUOTE` is used. However, when `FOO` is compiled, if `FUNCTION` is used the compiler will define and compile the `expr` definition as an auxiliary function (see Chapter 18). The compiled `expr` definition will run considerably faster, which can make a big difference if it is applied repeatedly.

Compiling `FUNCTION` will *not* create an auxiliary function if it is a functional argument to a function that compiles open, such as most of the mapping functions (`MAPCAR`, `MAPLIST`, etc.).

If `ENV` is not `NIL`, it can be a list of variables that are (presumably) used freely by `FN`. `ENV` can also be an atom, in which case it is evaluated, and the value interpreted as described above.

Macros

Macros provide an alternative way of specifying the action of a function. Whereas function definitions are evaluated with a “function call”, which involves binding variables and other housekeeping tasks, macros are evaluated by *translating* one Interlisp form into another, which is then evaluated.

A symbol may have both a function definition and a macro definition. When a form is evaluated by the interpreter, if the `CAR` has a function definition, it is used (with a function call), otherwise if it has a macro definition, then that is used. However, when a form is compiled, the `CAR` is checked for a macro definition first, and only if there isn’t one is the function definition compiled. This allows functions that behave differently when compiled and interpreted. For example, it is possible to define a function that, when interpreted, has a function definition that is slow and has a lot of error checks, for

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

use when debugging a system. This function could also have a macro definition that defines a fast version of the function, which is used when the debugged system is compiled.

Macro definitions are represented by lists that are stored on the property list of a symbol. Macros are often used for functions that should be compiled differently in different Interlisp implementations, and the exact property name a macro definition is stored under determines whether it should be used in a particular implementation. The global variable `MACROPROPS` contains a list of all possible macro property names which should be saved by the `MACROS` file package command. Typical macro property names are `DMACRO` for Interlisp-D, `10MACRO` for Interlisp-10, `VAXMACRO` for Interlisp-VAX, `JMACRO` for Interlisp-Jerico, and `MACRO` for “implementation independent” macros. The global variable `COMPILERMACROPROPS` is a list of macro property names. Interlisp determines whether a symbol has a macro definition by checking these property names, in order, and using the first non-`NIL` property value as the macro definition. In Interlisp-D this list contains `DMACRO` and `MACRO` in that order so that `DMACROS` will override the implementation-independent `MACRO` properties. In general, use a `DMACRO` property for macros that are to be used only in Interlisp-D, use `10MACRO` for macros that are to be used only in Interlisp-10, and use `MACRO` for macros that are to affect both systems.

Macro definitions can take the following forms:

(LAMBDA ...)

(NLAMBDA ...)

A function can be made to compile open by giving it a macro definition of the form `(LAMBDA ...)` or `(NLAMBDA ...)`, e.g., `(LAMBDA (X) (COND ((GREATERP X 0) X) (T (MINUS X))))` for `ABS`. The effect is as if the macro definition were written in place of the function wherever it appears in a function being compiled, i.e., it compiles as a `lambda` or `nlambda` expression. This saves the time necessary to call the function at the price of more compiled code generated in-line.

(NIL EXPRESSION)

(LIST EXPRESSION)

“Substitution” macro. Each argument in the form being evaluated or compiled is substituted for the corresponding atom in `LIST`, and the result of the substitution is used instead of the form. For example, if the macro definition of `ADD1` is `((X) (IPLUS X 1))`, then, `(ADD1 (CAR Y))` is compiled as `(IPLUS (CAR Y) 1)`.

Note that `ABS` could be defined by the substitution macro `((X) (COND ((GREATERP X 0) X) (T (MINUS X))))`. In this case, however, `(ABS (FOO X))` would compile as

```
(COND ((GREATERP (FOO X) 0)
      (FOO X))
      (T (MINUS (FOO X))))
```

and `(FOO X)` would be evaluated two times. (Code to evaluate `(FOO X)` would be generated three times.)

(OPENLAMBDA ARGS BODY)

This is a cross between substitution and `LAMBDA` macros. When the compiler processes an `OPENLAMBDA`, it attempts to substitute the actual arguments for the formals wherever this preserves the frequency and order of evaluation that would have resulted from a `LAMBDA` expression, and produces a `LAMBDA` binding only for those that require it.

INTERLISP-D REFERENCE MANUAL

Note: `OPENLAMBDA` assumes that it can substitute literally the actual arguments for the formal arguments in the body of the macro if the actual is side-effect free or a constant. Thus, you should be careful to use names in `ARGS` which don't occur in `BODY` (except as variable references). For example, if `FOO` has a macro definition of

```
(OPENLAMBDA (ENV) (FETCH (MY-RECORD-TYPE ENV) OF BAR))
```

then `(FOO NIL)` will expand to

```
(FETCH (MY-RECORD-TYPE NIL) OF BAR)
```

- T** When a macro definition is the atom `T`, it means that the compiler should ignore the macro, and compile the function definition; this is a simple way of turning off other macros. For example, the user may have a function that runs in both Interlisp-D and Interlisp-10, but has a macro definition that should only be used when compiling in Interlisp-10. If the `MACRO` property has the macro specification, a `DMACRO` of `T` will cause it to be ignored by the Interlisp-D compiler. This `DMACRO` would not be necessary if the macro were specified by a `10MACRO` instead of a `MACRO`.

`(= . OTHER-FUNCTION)`

A simple way to tell the compiler to compile one function exactly as it would compile another. For example, when compiling in Interlisp-D, `FRPLACAS` are treated as `RPLACAS`. This is achieved by having `FRPLACA` have a `DMACRO` of `(= . RPLACA)`.

`(LITATOM EXPRESSION)`

If a macro definition begins with a symbol other than those given above, this allows *computation* of the Interlisp expression to be evaluated or compiled in place of the form. `LITATOM` is bound to the `CDR` of the calling form, `EXPRESSION` is evaluated, and the result of this evaluation is evaluated or compiled in place of the form. For example, `LIST` could be compiled using the computed macro:

```
[X (LIST 'CONS (CAR X) (AND (CDR X) (CONS 'LIST (CDR X))
```

This would cause `(LIST X Y Z)` to compile as `(CONS X (CONS Y (CONS Z NIL)))`. Note the recursion in the macro expansion.

If the result of the evaluation is the symbol `IGNOREMACRO`, the macro is ignored and the compilation of the expression proceeds as if there were no macro definition. If the symbol in question is normally treated specially by the compiler (`CAR`, `CDR`, `COND`, `AND`, etc.), and also has a macro, if the macro expansion returns `IGNOREMACRO`, the symbol will still be treated specially.

In Interlisp-10, if the result of the evaluation is the atom `INSTRUCTIONS`, no code will be generated by the compiler. It is then assumed the evaluation was done for effect and the necessary code, if any, has been added. This is a way of giving direct instructions to the compiler if you understand it.

It is often useful, when constructing complex macro expressions, to use the `BQUOTE` facility (see the Read Macros section of Chapter 25).

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

The following function is quite useful for debugging macro definitions:

`(EXPANDMACRO EXP QUIETFLG -)` [Function]

Takes a form whose `CAR` has a macro definition and expands the form as it would be compiled. The result is prettyprinted, unless `QUIETFLG=T`, in which case the result is simply returned.

Note: `EXPANDMACRO` only works on Interlisp macros. Use `CL:MACROEXPAND-1` to expand Interlisp macros visible to the Common Lisp interpreter and compiler.

DEFMACRO

Macros defined with the function `DEFMACRO` are much like “computed” macros (see the above section), in that they are defined with a form that is evaluated, and the result of the evaluation is used (evaluated or compiled) in place of the macro call. However, `DEFMACRO` macros support complex argument lists with optional arguments, default values, and keyword arguments as well as argument list destructuring.

`(DEFMACRO NAME ARGS FORM)` [NLambda NoSpread Function]

Defines `NAME` as a macro with the arguments `ARGS` and the definition form `FORM` (`NAME`, `ARGS`, and `FORM` are unevaluated). If an expression starting with `NAME` is evaluated or compiled, arguments are bound according to `ARGS`, `FORM` is evaluated, and the value of `FORM` is evaluated or compiled instead. The interpretation of `ARGS` is described below.

Note: Like the function `DEFMACRO` in Common Lisp, this function currently removes any function definition for `NAME`.

`ARGS` is a list that defines how the argument list passed to the macro `NAME` is interpreted. Specifically, `ARGS` defines a set of variables that are set to various arguments in the macro call (unevaluated), that `FORM` can reference to construct the macro form.

In the simplest case, `ARGS` is a simple list of variable names that are set to the corresponding elements of the macro call (unevaluated). For example, given:

```
(DEFMACRO FOO (A B) (LIST 'PLUS A B B))
```

The macro call `(FOO X (BAR Y Z))` will expand to `(PLUS X (BAR Y Z) (BAR Y Z))`.

“&-keywords” (beginning with the character “&”) that are used to set variables to particular items from the macro call form, as follows:

&OPTIONAL Used to define optional arguments, possibly with default values. Each element on `ARGS` after `&OPTIONAL` until the next &-keyword or the end of the list defines an optional argument, which can either be a symbol or a list, interpreted as follows:

`VAR`

If an optional argument is specified as a symbol, that variable is set to the corresponding element of the macro call (unevaluated).

`(VAR DEFAULT)`

INTERLISP-D REFERENCE MANUAL

If an optional argument is specified as a two element list, *VAR* is the variable to be set, and *DEFAULT* is a form that is evaluated and used as the default if there is no corresponding element in the macro call.

```
(VAR DEFAULT VARSETP)
```

If an optional argument is specified as a three element list, *VAR* and *DEFAULT* are the variable to be set and the default form, and *VARSETP* is a variable that is set to *T* if the optional argument is given in the macro call, *NIL* otherwise. This can be used to determine whether the argument was not given, or whether it was specified with the default value.

For example, after

```
(DEFMACRO FOO (&OPTIONAL A (B 5) (C 6 CSET)) FORM)
```

 expanding the macro call

```
(FOO)
```

 would cause *FORM* to be evaluated with *A* set to *NIL*, *B* set to 5, *C* set to 6, and *CSET* set to *NIL*.

```
(FOO 4 5 6)
```

 would be the same, except that *A* would be set to 4 and *CSET* would be set to *T*.

&REST

&BODY Used to get a list of all additional arguments from the macro call. Either **&REST** or **&BODY** should be followed by a single symbol, which is set to a list of all arguments to the macro after the position of the **&**-keyword. For example, given

```
(DEFMACRO FOO (A B &REST C) FORM)
```

expanding the macro call

```
(FOO 1 2 3 4 5)
```

 would cause *FORM* to be evaluated with *A* set to 1, *B* set to 2, and *C* set to

```
(3 4 5)
```

.

If the macro calling form contains keyword arguments (see **&KEY** below), these are included in the **&REST** list.

&KEY

Used to define keyword arguments, that are specified in the macro call by including a “keyword” (a symbol starting with the character “:”) followed by a value.

Each element on *ARGS* after **&KEY** until the next **&**-keyword or the end of the list defines a keyword argument, which can either be a symbol or a list, interpreted as follows:

```
VAR  
(VAR)  
((KEYWORD VAR))
```

If a keyword argument is specified by a single symbol *VAR*, or a one-element list containing *VAR*, it is set to the value of a keyword argument, where the keyword used is created by adding the character “:” to the front of *VAR*. If a keyword argument is specified by a single-element list containing a two-element list, *KEYWORD* is interpreted as the keyword (which should start with the letter “:”), and *VAR* is the variable to set.

```
(VAR DEFAULT)  
((KEYWORD VAR) DEFAULT)  
(VAR DEFAULT VARSETP)  
((KEYWORD VAR) DEFAULT VARSETP)
```

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

If a keyword argument is specified by a two- or three-element list, the first element of the list specifies the keyword and variable to set as above. Similar to `&OPTIONAL` (above), the second element *DEFAULT* is a form that is evaluated and used as the default if there is no corresponding element in the macro call, and the third element *VARSETP* is a variable that is set to `T` if the optional argument is given in the macro call, `NIL` otherwise.

For example, the form

```
(DEFMACRO FOO (&KEY A (B 5 BSET) ((:BAR C) 6 CSET)) FORM)
```

Defines a macro with keys `:A`, `:B` (defaulting to `5`), and `:BAR`. Expanding the macro call `(FOO :BAR 2 :A 1)` would cause `FORM` to be evaluated with `A` set to `1`, `B` set to `5`, `BSET` set to `NIL`, `C` set to `2`, and `CSET` set to `T`.

&ALLOW-OTHER-KEYS

It is an error for any keywords to be supplied in a macro call that are not defined as keywords in the macro argument list, unless either the `&-keyword` `&ALLOW-OTHER-KEYS` appears in *ARGS*, or the keyword `:ALLOW-OTHER-KEYS` (with a non-`NIL` value) appears in the macro call.

&AUX

Used to bind and initialize auxiliary variables, using a syntax similar to `PROG` (see the `PROG` and Associated Control Functions section of Chapter 9). Any elements after `&AUX` should be either symbols or lists, interpreted as follows:

`VAR`

Single symbols are interpreted as auxiliary variables that are initially bound to `NIL`.

`(VAR EXP)`

If an auxiliary variable is specified as a two element list, `VAR` is a variable initially bound to the result of evaluating the form *EXP*.

For example, given

```
(DEFMACRO FOO (A B &AUX C (D 5)) FORM)
```

`C` will be bound to `NIL` and `D` to `5` when *FORM* is evaluated.

&WHOLE

Used to get the whole macro calling form. Should be the first element of *ARGS*, and should be followed by a single symbol, which is set to the entire macro calling form. Other `&-keywords` or arguments can follow. For example, given

```
(DEFMACRO FOO (&WHOLE X A B) FORM)
```

Expanding the macro call `(FOO 1 2)` would cause `FORM` to be evaluated with `X` set to `(FOO 1 2)`, `A` set to `1`, and `B` set to `2`.

`DEFMACRO` macros also support argument list “destructuring,” a facility for accessing the structure of individual arguments to a macro. Any place in an argument list where a symbol is expected, an argument list (in the form described above) can appear instead. Such an embedded

INTERLISP-D REFERENCE MANUAL

argument list is used to match the corresponding parts of that particular argument, which should be a list structure in the same form. In the simplest case, where the embedded argument list does not include &-keywords, this provides a simple way of picking apart list structures passed as arguments to a macro. For example, given

```
(DEFMACRO FOO (A (B (C . D)) E) FORM)
```

Expanding the macro call `(FOO 1 (2 (3 4 5)) 6)` would cause `FORM` to be evaluated with `A` set to 1, `B` set to 2, `C` set to 3, `D` set to `(4 5)`, and `E` set to 6. Note that the embedded argument list `(B (C . D))` has an embedded argument list `(C . D)`. Also notice that if an argument list ends in a dotted pair, that the final symbol matches the rest of the arguments in the macro call.

An embedded argument list can also include &-keywords, for interpreting parts of embedded list structures as if they appeared in a top-level macro call. For example, given

```
(DEFMACRO FOO (A (B &OPTIONAL (C 6)) D) FORM)
```

Expanding the macro call `(FOO 1 (2) 3)` would cause `FORM` to be evaluated with `A` set to 1, `B` set to 2, `C` set to 6 (because of the default value), and `D` set to 3.

Warning: Embedded argument lists can only appear in positions in an argument list where a list is otherwise not accepted. In the above example, it would not be possible to specify an embedded argument list after the `&OPTIONAL` keyword, because it would be interpreted as an optional argument specification (with variable name, default value, set variable). However, it would be possible to specify an embedded argument list as the first element of an optional argument specification list, as so:

```
(DEFMACRO FOO (A (B &OPTIONAL ((X (Y) Z)
                               '(1 (2) 3))) D) FORM)
```

In this case, `x`, `y`, and `z` default to 1, 2, and 3, respectively. Note that the “default” value has to be an appropriate list structure. Also, in this case either the whole structure `(X (Y) Z)` can be supplied, or it can be defaulted (i.e., is not possible to specify `x` while letting `y` default).

Interpreting Macros

When the interpreter encounters a form `CAR` of which is an undefined function, it tries interpreting it as a macro. If `CAR` of the form has a macro definition, the macro is expanded, and the result of this expansion is evaluated in place of the original form. `CLISPTRAN` (see the Miscellaneous Functions and Variables section of Chapter 21) is used to save the result of this expansion so that the expansion only has to be done once. On subsequent occasions, the translation (expansion) is retrieved from `CLISPARRAY` the same as for other `CLISP` constructs.

Note: Because of the way that the evaluator processes macros, if you have a macro on `FOO`, then typing `(FOO 'A 'B)` will work, but `FOO(A B)` will not work.

FUNCTION DEFINITION, MANIPULATION AND EVALUATION

[This page intentionally left blank]

INTERLISP-D REFERENCE MANUAL

11. VARIABLE BINDINGS AND THE STACK

Medley uses “deep binding.” Every time a function is entered, a basic frame containing the new variables is put on top of the stack. Therefore, any variable reference requires searching the stack for the first instance of that variable, which makes free variable use somewhat more expensive than in a shallow binding scheme. On the other hand, spaghetti stack operations are considerably faster. Some other tricks involving copying freely-referenced variables to higher frames on the stack are also used to speed up the search.

The basic frames are allocated on a stack; for most user purposes, these frames should be thought of as containing the variable names associated with the function call, and the *current* values for that frame. The descriptions of the stack functions in below are presented from this viewpoint. Both interpreted and compiled functions store both the names and values of variables so that interpreted and compiled functions are compatible and can be freely intermixed, i.e., free variables can be used with no SPECVAR declarations necessary. However, it is possible to *suppress* storing of names in compiled functions, either for efficiency or to avoid a clash, via a LOCALVAR declaration (see the Local Variables and Special Variables section of Chapter 18). The names are also very useful in debugging, for they make possible a complete symbolic backtrace in case of error.

In addition to the binding information, additional information is associated with each function call: access information indicating the path to search the basic frames for variable bindings, control information, and temporary results are also stored on the stack in a block called the frame extension. The interpreter also stores information about partially evaluated expressions as described in the Stack and Interpreter section of Chapter 11.

Spaghetti Stack

The Bobrow/Wegbreit paper, “A Model and Stack Implementation for Multiple Environments” (*Communications of the ACM*, Vol. 16, 10, October 1973.), describes an access and control mechanism more general than a simple linear stack. The access and control mechanism used by Interlisp is a slightly modified version of the one proposed by Bobrow and Wegbreit. This mechanism is called the “spaghetti stack.”

The spaghetti system presents the access and control stack as a data structure composed of “frames.” The functions described below operate on this structure. These primitives allow user functions to manipulate the stack in a machine independent way. Backtracking, coroutines, and more sophisticated control schemes can be easily implemented with these primitives.

The evaluation of a function requires the allocation of storage to hold the values of its local variables during the computation. In addition to variable bindings, an activation of a function requires a return link (indicating where control is to go after the completion of the computation) and room for temporaries needed during the computation. In the spaghetti system, one “stack” is used for storing all this information, but it is best to view this stack as a tree of linked objects called frame extensions (or simply frames).

A frame extension is a variable sized block of storage containing a frame name, a pointer to some variable bindings (the BLINK), and two pointers to other frame extensions (the ALINK and CLINK). In

INTERLISP-D REFERENCE MANUAL

addition to these components, a frame extension contains other information (such as temporaries and reference counts) that does not interest us here.

The block of storage holding the variable bindings is called a basic frame. A basic frame is essentially an array of pairs, each of which contains a variable name and its value. The reason frame extensions point to basic frames (rather than just having them “built in”) is so that two frame extensions can share a common basic frame. This allows two processes to communicate via shared variable bindings.

The chain of frame extensions which can be reached via the successive ALINKs from a given frame is called the “access chain” of the frame. The first frame in the access chain is the starting frame. The chain through successive CLINKs is called the “control chain”.

A frame extension completely specifies the variable bindings and control information necessary for the evaluation of a function. Whenever a function (or in fact, any form which generally binds local variables) is evaluated, it is associated with some frame extension.

In the beginning there is precisely one frame extension in existence. This is the frame in which the top-level call to the interpreter is being run. This frame is called the “top-level” frame.

Since precisely one function is being executed at any instant, exactly one frame is distinguished as having the “control bubble” in it. This frame is called the active frame. Initially, the top-level frame is the active frame. If the computation in the active frame invokes another function, a new basic frame and frame extension are built. The frame name of this basic frame will be the name of the function being called. The ALINK, BLINK, and CLINK of the new frame all depend on precisely how the function is invoked. The new function is then run in this new frame by passing control to that frame, i.e., it is made the active frame.

Once the active computation has been completed, control normally returns to the frame pointed to by the CLINK of the active frame. That is, the frame in the CLINK becomes the active frame.

In most cases, the storage associated with the basic frame and frame extension just abandoned can be reclaimed. However, it is possible to obtain a pointer to a frame extension and to “hold on” to this frame even after it has been exited. This pointer can be used later to run another computation in that environment, or even “continue” the exited computation.

A separate data type, called a stack pointer, is used for this purpose. A stack pointer is just a cell that literally points to a frame extension. Stack pointers print as #ADR/FRAMENAME, e.g., #1,13636/COND. Stack pointers are returned by many of the stack manipulating functions described below. Except for certain abbreviations (such as “the frame with such-and-such a name”), stack pointers are the only way you can reference a frame extension. As long as you have a stack pointer which references a frame extension, that frame extension (and all those that can be reached from it) will not be garbage collected.

Two stack pointers referencing the same frame extension are *not* necessarily EQ, i.e., (EQ (STKPOS 'FOO) (STKPOS 'FOO)) = NIL. However, EQP can be used to test if two different stack pointers reference the same frame extension (see the Equality Predicates section of Chapter 9).

It is possible to evaluate a form with respect to an access chain other than the current one by using a stack pointer to refer to the head of the access chain desired. Note, however, that this can be very expensive when using a shallow binding scheme such as that in Interlisp-10. When evaluating the form, since all references to variables under the shallow binding scheme go through the variable's value cell, the values in the value cells must be adjusted to reflect the values appropriate to the desired

VARIABLE BINDINGS AND THE STACK

access chain. This is done by changing all the bindings on the current access chain (all the name-value pairs) so that they contain the value current at the time of the call. Then along the new access path, all bindings are made to contain the previous value of the variable, and the current value is placed in the value cell. For that part of the access path which is shared by the old and new chain, no work has to be done. The context switching time, i.e. the overhead in switching from the current, active, access chain to another one, is directly proportional to the size of the two branches that are not shared between the access contexts. This cost should be remembered in using generators and coroutines (see the Generators section below).

Stack Functions

In the descriptions of the stack functions below, when we refer to an argument as a stack descriptor, we mean that it is one of the following:

- A stack pointer An object that points to a frame on the stack. Stack pointers are returned by many of the stack manipulating functions described below.
- NIL Specifies the active frame; that is, the frame of the stack function itself.
- T Specifies the top-level frame.
- A symbol Specifies the first frame (along the control chain from the active frame) that has the frame name LITATOM. Equivalent to (STKPOS LITATOM -1).
- A list of symbols Specifies the first frame (along the control chain from the active frame) whose frame name is included in the list.
- A number *N* Specifies the *N*th frame back from the active frame. If *N* is negative, the control chain is followed, otherwise the access chain is followed. Equivalent to (STKNTH *N*).

In the stack functions described below, the following errors can occur: The error `Illegal stack arg` occurs when a stack descriptor is expected and the supplied argument is either not a legal stack descriptor (i.e., not a stack pointer, symbol, or number), or is a symbol or number for which there is no corresponding stack frame, e.g., (STKNTH -1 'FOO) where there is no frame named FOO in the active control chain or (STKNTH -10 'EVALQT). The error `Stack pointer has been released` occurs whenever a released stack pointer is supplied as a stack descriptor argument for any purpose other than as a stack pointer to re-use.

Note: The creation of a single stack pointer can result in the retention of a large amount of stack space. Therefore, one should try to release stack pointers when they are no longer needed (see the Releasing and Reusing Stack Pointers section below).

In Lisp there is a fixed amount of space allocated for the stack. When most of this space is exhausted, the `STACK OVERFLOW` error occurs and the debugger will be invoked. You will still have a little room on the stack to use inside the debugger. If you use up this last little bit of stack you will encounter a “hard” stack overflow. A “hard” stack overflow will put you into URaid (see the documentation on URaid).

Searching the Stack

(**STKPOS** *FRAMENAME N POS OLDPOS*) [Function]

Returns a stack pointer to the *N*th frame with frame name *FRAMENAME*. The search begins with (and includes) the frame specified by the stack descriptor *POS*. The search proceeds along the control chain from *POS* if *N* is negative, or along the access chain if *N* is positive. If *N* is NIL, -1 is used. Returns a stack pointer to the frame if such a frame exists, otherwise returns NIL. If *OLDPOS* is supplied and is a stack pointer, it is reused. If *OLDPOS* is supplied and is a stack pointer and *STKPOS* returns NIL, *OLDPOS* is released. If *OLDPOS* is not a stack pointer it is ignored.

(**STKNTH** *N POS OLDPOS*) [Function]

Returns a stack pointer to the *N*th frame back from the frame specified by the stack descriptor *POS*. If *N* is negative, the control chain from *POS* is followed. If *N* is positive the access chain is followed. If *N* equals 0, *STKNTH* returns a stack pointer to *POS* (this provides a way to copy a stack pointer). Returns NIL if there are fewer than *N* frames in the appropriate chain. If *OLDPOS* is supplied and is a stack pointer, it is reused. If *OLDPOS* is not a stack pointer it is ignored.

Note: (*STKNTH* 0) causes an error, *Illegal stack arg*; it is not possible to create a stack pointer to the active frame.

(**STKNAME** *POS*) [Function]

Returns the frame name of the frame specified by the stack descriptor *POS*.

(**SETSTKNAME** *POS NAME*) [Function]

Changes the frame name of the frame specified by *POS* to be *NAME*. Returns *NAME*.

(**STKNTHNAME** *N POS*) [Function]

Returns the frame name of the *N*th frame back from *POS*. Equivalent to (*STKNAME* (*STKNTH N POS*)) but avoids creation of a stack pointer.

In summary, *STKPOS* converts function names to stack pointers, *STKNTH* converts numbers to stack pointers, *STKNAME* converts stack pointers to function names, and *STKNTHNAME* converts numbers to function names.

Variable Bindings in Stack Frames

The following functions are used for accessing and changing bindings. Some of functions take an argument, *N*, which specifies a particular binding in the basic frame. If *N* is a literal atom, it is assumed to be the name of a variable bound in the basic frame. If *N* is a number, it is assumed to reference the *N*th binding in the basic frame. The first binding is 1. If the basic frame contains no binding with the given name or if the number is too large or too small, the error *Illegal arg* occurs.

VARIABLE BINDINGS AND THE STACK

(**STKSCAN** *VAR IPOS OPOS*) [Function]

Searches beginning at *IPOS* for a frame in which a variable named *VAR* is bound. The search follows the access chain. Returns a stack pointer to the frame if found, otherwise returns NIL. If *OPOS* is a stack pointer it is reused, otherwise it is ignored.

(**FRAMESCAN** *ATOM POS*) [Function]

Returns the relative position of the binding of *ATOM* in the basic frame of *POS*. Returns NIL if *ATOM* is not found.

(**STKARG** *N POS* —) [Function]

Returns the value of the binding specified by *N* in the basic frame of the frame specified by the stack descriptor *POS*. *N* can be a literal atom or number.

(**STKARGNAME** *N POS*) [Function]

Returns the name of the binding specified by *N*, in the basic frame of the frame specified by the stack descriptor *POS*. *N* can be a literal atom or number.

(**SETSTKARG** *N POS VAL*) [Function]

Sets the value of the binding specified by *N* in the basic frame of the frame specified by the stack descriptor *POS*. *N* can be a literal atom or a number. Returns *VAL*.

(**SETSTKARGNAME** *N POS NAME*) [Function]

Sets the variable name to *NAME* of the binding specified by *N* in the basic frame of the frame specified by the stack descriptor *POS*. *N* can be a literal atom or a number. Returns *NAME*. This function does not work for interpreted frames.

(**STKNARGS** *POS* —) [Function]

Returns the number of arguments bound in the basic frame of the frame specified by the stack descriptor *POS*.

(**VARIABLES** *POS*) [Function]

Returns a list of the variables bound at *POS*.

(**STKARGS** *POS* —) [Function]

Returns a list of the values of the variables bound at *POS*.

Evaluating Expressions in Stack Frames

The following functions are used to evaluate an expression in a different environment:

(**ENVEVAL** *FORM APOS CPOS AFLG CFLG*) [Function]

Evaluates *FORM* in the environment specified by *APOS* and *CPOS*. That is, a new active frame is created with the frame specified by the stack descriptor *APOS* as its ALINK, and the frame specified by the stack descriptor *CPOS* as its CLINK. Then *FORM* is evaluated. If

INTERLISP-D REFERENCE MANUAL

AFLG is not NIL, and *APOS* is a stack pointer, then *APOS* will be released. Similarly, if *CFLG* is not NIL, and *CPOS* is a stack pointer, then *CPOS* will be released.

(**ENVAPPLY** *FN* *ARGS* *APOS* *CPOS* *AFLG* *CFLG*) [Function]

APPLYs *FN* to *ARGS* in the environment specified by *APOS* and *CPOS*. *AFLG* and *CFLG* have the same interpretation as with ENVEVAL.

(**EVALV** *VAR* *POS* *RELFLG*) [Function]

Evaluates *VAR*, where *VAR* is assumed to be a symbol, in the access environment specified by the stack descriptor *POS*. If *VAR* is unbound, EVALV returns NOBIND and does not generate an error. If *RELFLG* is non-NIL and *POS* is a stack pointer, it will be released after the variable is looked up. While EVALV could be defined as (ENVEVAL *VAR* *POS* NIL *RELFLG*) it is in fact somewhat faster.

(**STKEVAL** *POS* *FOR* *L*)
→) [Function]

Evaluates *FORM* in the access environment of the frame specified by the stack descriptor *POS*. If *FLG* is not NIL and *POS* is a stack pointer, releases *POS*. The definition of STKEVAL is (ENVEVAL *FORM* *POS* NIL *FLG*).

(**STKAPPLY** *POS* *FN* *ARGS* *FLG*) [Function]

Like STKEVAL but applies *FN* to *ARGS*.

Altering Flow of Control

The following functions are used to alter the normal flow of control, possibly jumping to a different frame on the stack. RETEVAL and RETAPPLY allow evaluating an expression in the specified environment first.

(**RETFROM** *POS* *VAL* *FLG*) [Function]

Return from the frame specified by the stack descriptor *POS*, with the value *VAL*. If *FLG* is not NIL, and *POS* is a stack pointer, then *POS* is released. An attempt to RETFROM the top level (e.g., (RETFROM T)) causes an error, Illegal stack arg. RETFROM can be written in terms of ENVEVAL as follows:

```
(RETFROM
  (LAMBDA (POS VAL FLG)
    (ENVEVAL (LIST 'QUOTE VAL)
      NIL
      (if (STKNTH -1 POS
        (if FLG then POS))
      else (ERRORX (LIST 19 POS)))
      NIL
      T)))
```

(**RETTO** *POS* *VAL* *FLG*) [Function]

Like RETFROM, but returns to the frame specified by *POS*.

VARIABLE BINDINGS AND THE STACK

(RETEVAL POS FORM FLG →) [Function]

Evaluates *FORM* in the access environment of the frame specified by the stack descriptor *POS*, and then returns from *POS* with that value. If *FLG* is not NIL and *POS* is a stack pointer, then *POS* is released. The definition of RETEVAL is equivalent to (ENVEVAL *FORM POS (STKNTH -1 POS) FLG T*), but RETEVAL does not create a stack pointer.

(RETAPPLY POS FN ARGS FLG) [Function]

Like RETEVAL but applies *FN* to *ARGS*.

Releasing and Reusing Stack Pointers

The following functions and variables are used for manipulating stack pointers:

(STACKP X) [Function]

Returns *X* if *X* is a stack pointer, otherwise returns NIL.

(RELSTK POS) [Function]

Release the stack pointer *POS* (see below). If *POS* is not a stack pointer, does nothing. Returns *POS*.

(RELSTKP X) [Function]

Returns T if *X* is a released stack pointer, NIL otherwise.

(CLEARSTK FLG) [Function]

If *FLG* is T, returns a list of all the active (unreleased) stack pointers. If *FLG* is NIL, this call is a no-op. The ability to clear all stack pointers is inconsistent with the modularity implicit in a multi processing environment.

CLEARSTKLST [Variable]

A variable used by the top-level executive. Every time the top-level executive is re-entered (e.g., following errors, or Control-D), CLEARSTKLST is checked. If its value is T, all active stack pointers are released using CLEARSTK. If its value is a list, then all stack pointers on that list are released. If its value is NIL, nothing is released. CLEARSTKLST is initially T.

NOCLEARSTKLST [Variable]

A variable used by the top-level executive. If CLEARSTKLST is T (see above) all active stack pointers *except* those on NOCLEARSTKLST are released. NOCLEARSTKLST is initially NIL.

Creating a single stack pointer can cause the retention of a large amount of stack space. Furthermore, this space will not be freed until the next garbage collection, *even if the stack pointer is no longer being used*, unless the stack pointer is explicitly released or reused. If there is sufficient amount of stack space tied up in this fashion, a STACK OVERFLOW condition can occur, even in the simplest of computations. For this reason, you should consider releasing a stack pointer when the environment referenced by the stack pointer is no longer needed.

INTERLISP-D REFERENCE MANUAL

The effects of releasing a stack pointer are:

1. The link between the stack pointer and the stack is broken by setting the contents of the stack pointer to the “released mark”. A released stack pointer prints as #*ADR*/*#0*.
2. If this stack pointer was the last remaining reference to a frame extension; that is, if no other stack pointer references the frame extension and the extension is not contained in the active control or access chain, then the extension may be reclaimed, and is reclaimed immediately. The process repeats for the access and control chains of the reclaimed extension so that all stack space that was reachable only from the released stack pointer is reclaimed.

A stack pointer may be released using the function `RELSTK`, but there are some cases for which `RELSTK` is not sufficient. For example, if a function contains a call to `RETFROM` in which a stack pointer was used to specify where to return to, it would not be possible to simultaneously release the stack pointer. (A `RELSTK` appearing in the function following the call to `RETFROM` would not be executed!) To permit release of a stack pointer in this situation, the stack functions that relinquish control have optional flag arguments to denote whether or not a stack pointer is to be released (`AFLG` and `CFLG`). Note that in this case releasing the stack pointer will *not* cause the stack space to be reclaimed immediately because the frame referenced by the stack pointer will have become part of the active environment.

Another way to avoid creating new stack pointers is to *reuse* stack pointers that are no longer needed. The stack functions that create stack pointers (`STKPOS`, `STKNTH`, and `STKSCAN`) have an optional argument that is a stack pointer to reuse. When a stack pointer is reused, two things happen. First the stack pointer is released (see above). Then the pointer to the new frame extension is deposited in the stack pointer. The old stack pointer (with its new contents) is returned as the value of the function. Note that the reused stack pointer will be released even if the function does not find the specified frame.

Even if stack pointers are explicitly being released, *creating* many stack pointers can cause a garbage collection of stack pointer space. Thus, if your application requires creating many stack pointers, you definitely should take advantage of reusing stack pointers.

Backtrace Functions

The following functions perform a “backtrace,” printing information about every frame on the stack. Arguments allow only backtracing a selected range of the stack, skipping selected frames, and printing different amounts of information about each frame.

(**BACKTRACE** *IPOS EPOS FLAGS FILE PRINTFN*) [Function]

Performs a backtrace beginning at the frame specified by the stack descriptor *IPOS*, and ending with the frame specified by the stack descriptor *EPOS*. *FLAGS* is a number in which the options of the `BACKTRACE` are encoded. If a bit is set, the corresponding information is included in the backtrace.

1Q - print arguments of non-SUBRs

2Q - print temporaries of the interpreter

4Q - print SUBR arguments and local variables

10Q - omit printing of `UNTRACE:` and function names

20Q - follow access chain instead of control chain

VARIABLE BINDINGS AND THE STACK

40Q - print temporaries, i.e. the blips (see the stack and interpreter section below)

For example: If *FLAGS* = 47Q, everything is printed. If *FLAGS* = 21Q, follows the access chain, prints arguments.

FILE is the file that the backtrace is printed to. *FILE* must be open. *PRINTFN* is used when printing the values of variables, temporaries, blips, etc. *PRINTFN* = NIL defaults to PRINT.

(**BAKTRACE** *IPOS EPOS SKIPFNS FLAGS FILE*) [Function]

Prints a backtrace from *IPOS* to *EPOS* onto *FILE*. *FLAGS* specifies the options of the backtrace, e.g., do/don't print arguments, do/don't print temporaries of the interpreter, etc., and is the same as for BACKTRACE.

SKIPFNS is a list of functions. As BAKTRACE scans down the stack, the stack name of each frame is passed to each function in *SKIPFNS*, and if any of them returnS non-NIL, *POS* is skipped (including all variables).

BAKTRACE collapses the sequence of several function calls corresponding to a call to a system package into a single "function" using BAKTRACELST as described below. For example, any call to the editor is printed as **EDITOR**, a break is printed as **BREAK**, etc.

BAKTRACE is used by the BT, BTv, BTv+, BTv*, and BTv! break commands, with *FLAGS* = 0, 1, 5, 7, and 47Q respectively.

If *SYSPRETTYFLG* = T, the values arguments and local variables will be prettyprinted.

BAKTRACELST [Variable]

Used to tell BAKTRACE (therefore, the BT, BTv, etc. commands) to abbreviate various sequences of function calls on the stack by a single key, e.g. **BREAK**, **EDITOR**, etc.

Each entry on BAKTRACELST is a list of the form (*FRAMENAME KEY . PATTERN*) or (*FRAMENAME (KEY . PATTERN) ... (KEY . PATTERN)*), where a pattern is a list of elements that are either atoms, which match a single frame, or lists, which are interpreted as a list of alternative patterns, e.g. (PROGN **BREAK** EVAL ((ERRORSET BREAK1A BREAK1) (BREAK1)))

BAKTRACE operates by scanning up the stack and, at each point, comparing the current frame name, with the frame names on BAKTRACELST, i.e. it does an ASSOC. If the frame name does appear, BAKTRACE attempts to match the stack as of that point with (one of) the patterns. If the match is successful, BAKTRACE prints the corresponding key, and continues with where the match left off. If the frame name does not appear, or the match fails, BAKTRACE simply prints the frame name and continues with the next higher frame (unless the *SKIPFNS* applied to the frame name are non-NIL as described above).

Matching is performed by comparing symbols in the pattern with the current frame name, and matching lists as patterns, i.e. sequences of function calls, always working up the stack. For example, either of the sequence of function calls "... BREAK1 BREAK1A ERRORSET EVAL PROGN ..."

INTERLISP-D REFERENCE MANUAL

or "... BREAK1 EVAL PROGN ..." would match with the sample entry given above, causing **BREAK** to be printed.

Special features:

- The symbol & can be used to match any frame.
- The pattern "-" can be used to match nothing. - is useful for specifying an optional match, e.g. the example above could also have been written as (PROGN **BREAK** EVAL ((ERRORSET BREAK1A) -) BREAK1).
- It is not necessary to provide in the pattern for matching dummy frames, i.e. frames for which DUMMYFRAMEP (see below) is true. When working on a match, the matcher automatically skips over these frames when they do not match.
- If a match succeeds and the KEY is NIL, nothing is printed. For example, (*PROG*LAM NIL EVALA *ENV). This sequence will occur following an error which then causes a break if some of the function's arguments are LOCALVARS.

Other Stack Functions

(**DUMMYFRAMEP** POS) [Function]

Returns T if you never wrote a call to the function at POS, e.g. in Interlisp-10, DUMMYFRAMEP is T for *PROG*LAM, *ENV*, and FOOBLOCK frames (see the Block Compiling section of Chapter 18).

REALFRAMEP and REALSTKNTH can be used to write functions which manipulate the stack and work on either interpreted or compiled code:

(**REALFRAMEP** POS INTERPFLG) [Function]

Returns POS if POS is a "real" frame, i.e. if POS is not a dummy frame and POS is a frame that does not disappear when compiled (such as COND); otherwise NIL. If INTERPFLG = T, returns T if POS is not a dummy frame. For example, if (STKNAME POS) = COND, (REALFRAMEP POS) is NIL, but (REALFRAMEP POS T) is T.

(**REALSTKNTH** N POS INTERPFLG OLDPOS) [Function]

Returns a stack pointer to the Nth (or -Nth) frames for which (REALFRAMEP POS INTERPFLG) is POS.

(**MAPDL** MAPDLFN MAPDLPOS) [Function]

Starts at MAPDLPOS and applies the function MAPDLFN to two arguments (the frame name and a stack pointer to the frame), for each frame until the top of the stack is reached. Returns NIL. For example,

```
[MAPDL (FUNCTION (LAMBDA (X POS)
  (if (IGREATERP (STKNARGS POS) 2) then (PRINT X))
```

will print all functions of more than two arguments.

VARIABLE BINDINGS AND THE STACK

(**SEARCHPDL** *SRCHF* *SRCHPOS*)

[Function]

Like MAPDL, but searches the stack starting at position *SRCHPOS* until it finds a frame for which *SRCHF*, a function of two arguments applied to the *name* of the frame and the frame itself, is not NIL. Returns (*NAME* . *FRAME*) if such a frame is found, otherwise NIL.

The Stack and the Interpreter

In addition to the names and values of arguments for functions, information regarding partially-evaluated expressions is kept on the push-down list. For example, consider the following definition of the function FACT (intentionally faulty):

```
(FACT
  [LAMBDA (N)
    (COND
      ((ZEROP N)
       L)
      (T (ITIMES N (FACT (SUB1 N)))))
```

In evaluating the form (FACT 1), as soon as FACT is entered, the interpreter begins evaluating the implicit PROG following the LAMBDA. The first function entered in this process is COND. COND begins to process its list of clauses. After calling ZEROP and getting a NIL value, COND proceeds to the next clause and evaluates T. Since T is true, the evaluation of the implicit PROG that is the consequent of the T clause is begun. This requires calling the function ITIMES. However before ITIMES can be called, its arguments must be evaluated. The first argument is evaluated by retrieving the current binding of N from its value cell; the second involves a recursive call to FACT, and another implicit PROG, etc.

At each stage of this process, some portion of an expression has been evaluated, and another is awaiting evaluation. The output below (from Interlisp-10) illustrates this by showing the state of the push-down list at the point in the computation of (FACT 1) when the unbound atom L is reached.

```
← FACT(1)
u.b.a. L {in FACT} in ((ZEROP NO L)
(L broken)
:BTV!
  *TAIL* (L)
  *ARG1 ((ZEROP N) L) (T (ITIMES N (FACT (SUB1 N)))))
COND
  *FORM* (COND ((ZEROP N) L) (T (ITIMES N (FACT (SUB1 N)))))
  *TAIL* ((COND ((ZEROP N) L) (T (ITIMES N (FACT (SUB1 N)))))
  N 0
FACT
  *FORM* (FACT (SUB1 N))
  *FN* ITIMES
  *TAIL* ((FACT (SUB1 N)))
  *ARGVAL* 1
  *FORM* (ITIMES N (FACT (SUB1 N)))
  *TAIL* ((ITIMES N (FACT (SUB1 N))))
  *ARG1 ((ZEROP N) L) (T (ITIMES N (FACT (SUB1 N)))))
COND
  *FORM* (COND ((ZEROP N) L) (T (ITIMES N (FACT (SUB1 N)))))
  *TAIL* ((COND ((ZEROP N) L) (T (ITIMES N (FACT (SUB1 N)))))
  N 1
FACT
```

INTERLISP-D REFERENCE MANUAL

****TOP****

Internal calls to EVAL, e.g., from COND and the interpreter, are marked on the push-down list by a special mark or blip which the backtrace prints as **FORM**. The genealogy of **FORM**'s is thus a history of the computation. Other temporary information stored on the stack by the interpreter includes the tail of a partially evaluated implicit PROG (e.g., a cond clause or lambda expression) and the tail of a partially evaluated form (i.e., those arguments not yet evaluated), both indicated on the backtrace by **TAIL**, the values of arguments that have already been evaluated, indicated by **ARGVAL**, and the names of functions waiting to be called, indicated by **FN**. **ARG1**, . . . , **ARGn** are used by the backtrace to indicate the (unnamed) arguments to SUBRS.

Note that a function is not actually entered and does not appear on the stack, until its arguments have been evaluated (except for lambda functions, of course). Also note that the **ARG1**, **FORM**, **TAIL**, etc. "bindings" comprise the actual working storage. In other words, in the above example, if a (lower) function changed the value of the **ARG1** binding, the COND would continue interpreting the new binding as a list of COND clauses. Similarly, if the **ARGVAL** binding were changed, the new value would be given to ITIMES as its first argument after its second argument had been evaluated, and ITIMES was actually called.

FORM, **TAIL**, **ARGVAL**, etc., do not actually appear as variables on the stack, i.e., evaluating **FORM** or calling STKSCAN to search for it will not work. However, the functions BLIPVAL, SETBLIPVAL, and BLIPSCAN described below are available for accessing these internal blips. These functions currently know about four different types of blips:

FN The name of a function about to be called

ARGVAL An argument for a function about to be called

FORM A form in the process of evaluation

TAIL The tail of a COND clause, implicit PROG, PROG, etc.

(**BLIPVAL** *BLIPTYP IPOS FLG*)

[Function]

Returns the value of the specified blip of type *BLIPTYP*. If *FLG* is a number N, finds the Nth blip of the desired type, searching the control chain beginning at the frame specified by the stack descriptor *IPOS*. If *FLG* is NIL, 1 is used. If *FLG* is T, returns the number of blips of the specified type at *IPOS*.

(**SETBLIPVAL** *BLIPTYP IPOS N VAL*)

[Function]

Sets the value of the specified blip of type *BLIPTYP*. Searches for the Nth blip of the desired type, beginning with the frame specified by the stack descriptor *IPOS*, and following the control chain.

(**BLIPSCAN** *BLIPTYP IPOS*)

[Function]

Returns a stack pointer to the frame in which a blip of type *BLIPTYP* is located. Search begins at the frame specified by the stack descriptor *IPOS* and follows the control chain.

Generators

A *generator* is like a subroutine except that it retains information about previous times it has been called. Some of this state may be data (for example, the seed in a random number generator), and some may be in program state (as in a recursive generator which finds all the atoms in a list structure). For example, if LISTGEN is defined by:

```
(DEFINEQ (LISTGEN (L)
  (if L then (PRODUCE (CAR L))
    (LISTGEN (CDR L))))
```

we can use the function GENERATOR (described below) to create a generator that uses LISTGEN to produce the elements of a list one at a time, e.g.,

```
(SETQ GR (GENERATOR (LISTGEN ' (A B C))))
```

creates a generator, which can be called by

```
(GENERATE GR)
```

to produce as values on successive calls, A, B, C. When GENERATE (not GENERATOR) is called the first time, it simply starts evaluating (LISTGEN ' (A B C)). PRODUCE gets called from LISTGEN, and pops back up to GENERATE with the indicated value after saving the state. When GENERATE gets called again, it continues from where the last PRODUCE left off. This process continues until finally LISTGEN completes and returns a value (it doesn't matter what it is). GENERATE then returns GR itself as its value, so that the program that called GENERATE can tell that it is finished, i.e., there are no more values to be generated.

(**GENERATOR** *FORM COMVAR*)

[NLambda Function]

An nlambda function that creates a generator which uses *FORM* to compute values. GENERATOR returns a *generator handle* which is represented by a dotted pair of stack pointers.

COMVAR is optional. If its value (EVAL of) is a generator handle, the list structure and stack pointers will be reused. Otherwise, a new generator handle will be constructed.

GENERATOR compiles open.

(**PRODUCE** *VAL*)

[Function]

Used from within a generator to return *VAL* as the value of the corresponding call to GENERATE.

(**GENERATE** *HANDLE VAL*)

[Function]

Restarts the generator represented by *HANDLE*. *VAL* is returned as the value of the PRODUCE which last suspended the operation of the generator. When the generator runs out of values, GENERATE returns *HANDLE* itself.

Examples:

The following function will go down recursively through a list structure and produce the atoms in the list structure one at a time.

```
(DEFINEQ (LEAVESG (L)
  (if (ATOM L)
```

INTERLISP-D REFERENCE MANUAL

```
      then (PRODUCE L)
    else (LEAVESG (CAR L))
          (if (CDR L)
              then (LEAVESG (CDR L))]
```

The following function prints each of these atoms as it appears. It illustrates how a loop can be set up to use a generator.

```
(DEFINEQ (PLEAVESG1 (L)
  (PROG (X LHANDLE)
    (SETQ LHANDLE (GENERATOR (LEAVESG L)))
    LP (SETQ X (GENERATE LHANDLE))
      (if (EQ X LHANDLE)
          then (RETURN NIL))
      (PRINT X)
      (GO LP)))]
```

The loop terminates when the value of the generator is EQ to the dotted pair which is the value produced by the call to GENERATOR. A CLISP iterative operator, OUTOF, is provided which makes it much easier to write the loop in PLEAVESG1. OUTOF (or outof) can precede a form which is to be used as a generator. On each iteration, the iteration variable will be set to successive values returned by the generator; the loop will be terminated automatically when the generator runs out. Therefore, the following is equivalent to the above program PLEAVESG1:

```
(DEFINEQ (PLEAVESG2 (L) (for X outof (LEAVESG L) do (PRINT X)))]
```

Here is another example; the following form will print the first N atoms.

```
(for X outof (MAPATOMS (FUNCTION PRODUCE)) as I from 1 to N do (PRINT X))
```

Coroutines

This package provides facilities for the creation and use of fully general coroutine structures. It uses a stack pointer to preserve the state of a coroutine, and allows arbitrary switching between *N* different coroutines, rather than just a call to a generator and return. This package is slightly more efficient than the generator package described above, and allows more flexibility on specification of what to do when a coroutine terminates.

(**COROUTINE** *CALLPTR COROUTPTR COROUTFORM ENDFORM*) [NLambda Function]

This nlambda function is used to create a coroutine and initialize the linkage. *CALLPTR* and *COROUTPTR* are the names of two variables, which will be set to appropriate stack pointers. If the values of *CALLPTR* or *COROUTPTR* are already stack pointers, the stack pointers will be reused. *COROUTFORM* is the form which is evaluated to start the coroutine; *ENDFORM* is a form to be evaluated if *COROUTFORM* actually returns when it runs out of values.

COROUTINE compiles open.

(**RESUME** *FROMPTR TOPTR VAL*) [Function]

Used to transfer control from one coroutine to another. *FROMPTR* should be the stack pointer for the current coroutine, which will be smashed to preserve the current state. *TOPTR* should be the stack pointer which has preserved the state of the coroutine to be transferred to, and *VAL* is the value that is to be returned to the latter coroutine as the value of the RESUME which suspended the operation of that coroutine.

VARIABLE BINDINGS AND THE STACK

For example, the following is the way one might write the LEAVES program using the coroutine package:

```
(DEFINEQ (LEAVESC (L COROUTPTR CALLPTR)
  (if (ATOM L)
    then (RESUME COROUTPTR CALLPTR L)
    else (LEAVESC (CAR L) COROUTPTR CALLPTR)
          (if (CDR L) then (LEAVESC (CDR L) COROUTPTR CALLPTR))))))
```

A function PLEAVESC which uses LEAVESC can be defined as follows:

```
(DEFINEQ (PLEAVESC (L)
  (bind PLHANDLE LHANDLE
    first (COROUTINE PLHANDLE LHANDLE
      (LEAVESC L LHANDLE PLHANDLE)
      (RETFROM 'PLEAVESC))
    do (PRINT (RESUME PLHANDLE LHANDLE))))]
```

By RESUMEing LEAVESC repeatedly, this function will print all the leaves of list L and then return out of PLEAVESC via the RETFROM. The RETFROM is necessary to break out of the non-terminating do-loop. This was done to illustrate the additional flexibility allowed through the use of ENDFORM.

We use two coroutines working on two trees in the example EQLEAVES, defined below. EQLEAVES tests to see whether two trees have the same leaf set in the same order, e.g., (EQLEAVES ' (A B C) ' (A B (C))) is true.

```
(DEFINEQ (EQLEAVES (L1 L2)
  (bind LHANDLE1 LHANDLE2 PE EL1 EL2
    first (COROUTINE PE LHANDLE1 (LEAVESC L1 LHANDLE1 PE) 'NO-MORE)
          (COROUTINE PE LHANDLE2 (LEAVESC L2 LHANDLE2 PE) 'NO-MORE)
    do (SETQ EL1 (RESUME PE LHANDLE1))
      (SETQ EL2 (RESUME PE LHANDLE2))
      (if (NEQ EL1 EL2)
        then (RETURN NIL))
    repeatuntil (EQ EL1 'NO-MORE)
    finally (RETURN T))))]
```

Possibilities Lists

A possibilities list is the interface between a generator and a consumer. The possibilities list is initialized by a call to POSSIBILITIES, and elements are obtained from it by using TRYNEXT. By using the spaghetti stack to maintain separate environments, this package allows a regime in which a generator can put a few items in a possibilities list, suspend itself until they have been consumed, and be subsequently aroused and generate some more.

(POSSIBILITIES *FORM*)

[NLambda Function]

This nlambda function is used for the initial creation of a possibilities list. *FORM* will be evaluated to create the list. It should use the functions NOTE and AU-REVOIR described below to generate possibilities. Normally, one would set some variable to the possibilities list which is returned, so it can be used later, e.g.:

```
(SETQ PLIST (POSSIBILITIES (GENERFN V1 V2))).
```

POSSIBILITIES compiles open.

INTERLISP-D REFERENCE MANUAL

(**NOTE** *VAL* *LSTFLG*) [Function]

Used within a generator to put items on the possibilities list being generated. If *LSTFLG* is equal to *NIL*, *VAL* is treated as a single item. If *LSTFLG* is non-*NIL*, then the list *VAL* is *NCONC*ed on the end of the possibilities list. Note that it is perfectly reasonable to create a possibilities list using a second generator, and *NOTE* that list as possibilities for the current generator with *LSTFLG* equal to *T*. The lower generator will be resumed at the appropriate point.

(**AU-REVOIR** *VAL*) [NoSpread Function]

Puts *VAL* on the possibilities list if it is given, and then suspends the generator and returns to the consumer in such a fashion that control will return to the generator at the *AU-REVOIR* if the consumer exhausts the possibilities list.

NIL is not put on the possibilities list unless it is explicitly given as an argument to *AU-REVOIR*, i.e., (*AU-REVOIR*) and (*AU-REVOIR* *NIL*) are *not* the same. *AU-REVOIR* and *ADIEU* are lambda nospreads to enable them to distinguish these two cases.

(**ADIEU** *VAL*) [NoSpread Function]

Like *AU-REVOIR* but releases the generator instead of suspending it.

(**TRYNEXT** *PLST* *ENDFORM* *VAL*) [NLambda Function]

This nlambda function allows a consumer to use a possibilities list. It removes the first item from the possibilities list named by *PLST* (i.e. *PLST* must be an atom whose value is a possibilities list), and returns that item, provided it is not a generator handle. If a generator handle is encountered, the generator is reawakened. When it returns a possibilities list, this list is added to the front of the current list. When a call to *TRYNEXT* causes a generator to be awakened, *VAL* is returned as the value of the *AU-REVOIR* which put that generator to sleep. If *PLST* is empty, it evaluates *ENDFORM* in the caller's environment.

TRYNEXT compiles open.

(**CLEANPOSLST** *PLST*) [Function]

This function is provided to release any stack pointers which may be left in the *PLST* which was not used to exhaustion.

For example, *FIB* is a generator for fibonnaci numbers. It starts out by *NOTE*ing its two arguments, then suspends itself. Thereafter, on being re-awakened, it will *NOTE* two more terms in the series and suspends again. *PRINTFIB* uses *FIB* to print the first *N* fibonacci numbers.

```
(DEFINEQ (FIB (F1 F2)
  (do (NOTE F1)
    (NOTE F2)
    (SETQ F1 (IPLUS F1 F2))
    (SETQ F2 (IPLUS F1 F2))
    (AU-REVOIR) ]
```

Note that this *AU-REVOIR* just suspends the generator and adds nothing to the possibilities list except the generator.

VARIABLE BINDINGS AND THE STACK

```
(DEFINEQ (PRINTFIB (N)
  (PROG ((FL (POSSIBILITIES (FIB 0 1))))
    (RPTQ N (PRINT (TRYNEXT FL)))
    (CLEANPOSTLST FL)]
```

Note that FIB itself will never terminate.

INTERLISP-D REFERENCE MANUAL

[This page intentionally left blank]

Greeting and Initialization Files

Many of the features of Medley are controlled by variables that you can adjust to your own taste. In addition, you can modify the action of system functions in ways not specifically provided for by using `ADVISE` (see the Advise Functions section of Chapter 15). To encourage customizing Medley's environment, it includes a facility for automatically loading initialization files (or "init files") when it is first started. Each user can have a separate "user init file" that customizes Medley's environment to his/her tastes. In addition, there can be a "site init file" that applies to all users at a given physical site, setting system variables that are the same for all users such as the name of the nearest printer, etc.

The process of loading init files, also known as "greeting", occurs when a Medley system created by `MAKESYS` (see the Saving Virtual Memory State section below) is started for the first time. The user can also explicitly invoke the greeting operation at any time via the function `GREET` (below). The process of greeting includes the following steps:

1. Any previous greeting operation is undone. The side effects of the greeting operation are stored on a global variable as well as on the history list, thus enabling the previous greeting to be undone even if it has dropped off of the bottom of the history list.
2. All of the items on the list `PREGREETFORMS` are evaluated.
3. The site init file is loaded. `GREET` looks for a file by the name `{DSK}INIT.LISP`. If this is found, it is loaded. If it is not found, the system prints `Please enter name of system init file (e.g. {server}<directory>INIT.extension):` and waits for you to type a file name, followed by a carriage return. If you just type a carriage return without typing a file name, no site init file is loaded. **Note:** The site init file is loaded with `LDFLG` set to `SYSLOAD`, so that no file package information is saved, and nothing is printed out.
4. The user init file is loaded. The user init file is found by using the variable `USERGREETFILES` (described below), which is normally set in the site init file. The user init file is loaded with normal file package settings, but under errorset protection and with `PRETTYHEADER` set to `NIL` to suppress the `File created` message.
5. All of the items on the list `POSTGREETFORMS` are evaluated.
6. The greeting "Hello, XXX." is printed, where `XXX` is the value of the variable `FIRSTNAME` (if non-`NIL`). The variable `GREETDATES` (below) can be set to modify this greeting for particular dates.

(**GREET** *NAME* —)

[Function]

Performs the greeting for person whose username is *NAME* (if *NAME* = `NIL`, uses the login name). When Medley first starts up, it performs (`GREET`).

MEDLEY REFERENCE MANUAL

(GREETFILENAME *USER*) [Function]

If *USER* is T, GREETFILENAME returns the file name of the site init file. If the file name doesn't exist, you are prompted for it. Otherwise, *USER* is interpreted to be a user's system name, and GREETFILENAME returns the file name for the user init file (if it exists).

USERGREETFILES [Variable]

USERGREETFILES specifies a series of file names to try as the user init file. The value of USERGREETFILES is a list, where each element is a list of symbols. For each item in USERGREETFILES, the user name is substituted for the symbol USER and the value of COMPILE.EXT (see the Cimpiler Functions section of Chapter 18) is substituted for the symbol COM, and the symbols are packed into a single file name. The first such file that is found is the user init file.

For example, suppose that the value of USERGREETFILES was

```
(( {ERIS}< USER >LISP>INIT. COM)
  {ERIS}< USER >LISP>INIT)
  {ERIS}< USER >INIT. COM)
  {ERIS}< USER >INIT) )
```

If the user name was JONES, and the value of COMPILE.EXT was DCOM, then this would search for the files {ERIS}<JONES>LISP>INIT.DCOM, {ERIS}<JONES>LISP>INIT, {ERIS}<JONES>INIT.DCOM, and {ERIS}<JONES>INIT.

Note: The file name “specifications” in USERGREETFILES should be fully qualified, including all host and directory information. The directory search path (the value of DIRECTORIES, see the Searching File Directories section of Chapter 24) is *not* used to find the user greet files.

GREETDATES [Variable]

The value of GREETDATES can be used to specify special greeting messages for various dates. GREETDATES is a list of elements of the form (*DATESTRING* . *STRING*), e.g. ("25-DEC" . "Merry Christmas"). The user can add entries to this list in his/her INIT.LISP file by using a ADDVARS file package command like (ADDVARS (GREETDATES ("8-FEB" . "Happy Birthday"))). On the specified date, the GREET will use the indicated salutation.

It is impossible to give a complete list of all of the variables and functions you may want to set in your init files. The default values for system variables are chosen in the hope that they will be correct for the majority of users, so many users get along with very small init files. The following describes some of the variables that users may want to reset in their init files:

Directories The variables DIRECTORIES and LISPUSERSDIRECTORIES (see the Searching File Directories section of Chapter 24) contain lists of directories used when searching for files. LOGINHOST/DIR (see the Incomplete File Names section of Chapter 24) determines the default directory used when you call CONN with no argument.

MISCELLANEOUS

- Fonts and Printing** The variables `DISPLAYFONTDIRECTORIES`, `DISPLAYFONTEXTENSIONS`, `INTERPRESSFONTDIRECTORIES`, and `PRESSFONTWIDTHSFILES` (see the Font Files and Font Directories section of Chapter 27) must be set before fonts can be automatically loaded from files. `DEFAULTPRINTINGHOST` (see Chapter 29) should be set before attempting to generate hardcopy to a printer.
- Network Systems** `CH.DEFAULT.ORGANIZATION` and `CH.DEFAULT.DOMAIN` (see the Name and Address Conventions section of Chapter 31) should be set to the default NS organization and domain, when using NS network communications. If `CH.NET.HINT` (see the Clearinghouse Functions section of Chapter 31) is set, it can reduce the amount of time spent searching for a clearinghouse.
- Medley Executive** The variable `PROMPT#FLG` (see the Changing the Programmer's Assistant section of Chapter 13) determines whether an "event number" is printed at the beginning of every input line. The function `CHANGESLICE` (see the Changing the Programmer's Assistant section of Chapter 13) can be used to change the number of events that are remembered on the history list.
- Copyright Notices** `COPYRIGHTFLG`, `COPYRIGHTOWNERS`, and `DEFAULTCOPYRIGHTOWNER` (see the Copyright Notices section of Chapter 17) control the inclusion of copyright notices on source files.
- Printing Functions** `**COMMENT**FLG` (see the Comment Feature section of Chapter 26) determines how program comments are printed. `FIRSTCOL`, `PRETTYFLG`, and `CLISPIFYPRETTYFLG` (see the Special Prettyprint Controls section of Chapter 26) are among the many variables controlling how functions are pretty printed.
- List Structure Editor** The variable `INITIALSLST` (see the Time Stamps section of Chapter 16) is used when "time-stamps" are inserted in a function when it is edited. `EDITCHARACTERS` (see the Time Stamps section of Chapter 16) is used to set the read macros used in the teletype editor.

Idle Mode

The Medley environment runs on small single-user computers, usually located in users' offices. Often, users leave their computers up and running for days, which can cause several problems. First, the phosphor in the video display screen can be permanently marked if the same pattern is displayed for a long time (weeks). Second, if you go away, leaving a Medley system running, another person could possibly walk up and use the environment, taking advantage of any passwords that have been entered. To solve these problems, Medley implements the concept of "idle mode."

If no keyboard or mouse action has occurred for a specified time, Medley automatically enters idle mode. While idle mode is on, the display screen is blacked out, to protect the phosphor. Idle mode also runs a program to display some moving pattern on the black screen, so the screen does not appear to be broken. Usually, idle mode can be exited by pressing any key on the keyboard or mouse. However, you can optionally specify that idle mode should erase the current password cache when it is entered, and require the next user to supply a password to exit idle mode.

MEDLEY REFERENCE MANUAL

If either shift key is pressed while Medley is in idle mode, the current user name and the amount of time spent idling are displayed in the prompt window while the key is depressed.

Idle mode can also be entered by calling the function `IDLE`, or by selecting the Idle menu command from the background menu (see Chapter 28). The Idle menu command has subitems that allow you to interactively set the idle options (display program, erasing password, etc.) specified by the variable `IDLE.PROFILE`.

IDLE.PROFILE

[Variable]

The value of this variable is a property list (see Chapter 3) which controls most aspects of idle mode. The following properties are recognized:

TIMEOUT Value is a number that determines how long (in minutes) Medley will wait before automatically entering idle mode. If `NIL`, idle mode will never be entered automatically. Default is 10 minutes.

FORGET If this is the symbol `FIRST`, your password will be erased when idle mode is entered. If non-`NIL`, your password will be erased when idle mode is exited. Initial value is `T` (erase password on exit).

If the password is erased on entry to idle mode (value `FIRST`), any programs left running when idle mode is entered will fail if they try doing anything requiring passwords (such as accessing file servers).

ALLOWED.LOGINS The value of this property can either be a list or a non-list. If the value is `NIL` or any other non-list, idle mode is exited without requesting login.

If the value is a list the members of the list should be interpreted as follows:

- * If the value is a list containing * as it's element, login is required but anyone can exit idle mode. This will overwrite the previous user's user name and password each time idle mode is exited.

- T Let the previous user (as determined by `USERNAME`) exit idle mode. If the username has not been set, this is equivalent to *

- user name Let this specific user exit idle mode.

- group name Let any member of this group (an NS clearinghouse group) exit idle mode.

AUTHENTICATE The value of this property determines the method used for logging in. The value can be one of the following:

- T or NS Use the NS authentication protocol. This requires that you have an NS authentication server accessible on your net.

MISCELLANEOUS

GV Authenticate the login via the GrapeVine protocol.

UNIX Use the unix login mechanism.

Note: Unix is case sensitive. If you try to login but fail, you may have typed the password with the caps-lock on.

LOGIN.TIMEOUT This is the number of seconds idle will wait for a login before resuming idle mode again.

DISPLAYFN The value of this property, which should be a function name or lambda expression, is called to display a moving pattern on the screen while in idle mode. This function is called with one argument, a window covering the whole screen. The default is IDLE.BOUNCING.BOX (below).

Any function used as a DISPLAYFN should call BLOCK (see Chapter 23) frequently, so other programs can run during idle mode.

SAVEVM Value is a number that determines how long (in minutes) after idle mode is entered that SAVEVM will be called to save the virtual memory. If NIL, SAVEVM is never called automatically from idle mode. Default is 10 minutes.

SUSPEND.PROCESS.NAMES Value is a list of names. For each name on this list, if a process by that name is found, it will be suspended upon entering idle mode and woken upon exiting idle mode.

IDLE.RESETVARS [Variable]

The value of this variable is a list of two-element lists: ((VAR EXP) (VAR EXP) ...). On entering idle mode, each variable VAR is bound to the value of the corresponding expression EXP. When idle mode is exited, each variable VAR is reset to its original value.

IDLE.SUSPEND.PROCESS.NAMES [Variable]

Value is a list of names. For each name on this list, if a process by that name is found, it will be suspended upon entering idle mode and woken upon exiting idle mode.

IDLE.PROFILE [Variable]

The value of this variable determines the menu raised by selecting the Display subitem of the Idle background menu command. It should be in the format used for the ITEMS field of a menu (see Chapter 28), with the selection of an item returning the appropriate display function.

(IDLE.BOUNCING.BOX WINDOW BOX WAIT) [Variable]

This is the default display function used for idle mode. BOX is bounded about WINDOW, with bounces taking place every WAIT milliseconds. BOX can be a string, a bitmap, a window (whose image will be bounced about), or a list containing any number of these (which will be cycled through). BOX defaults to the value of the variable

MEDLEY REFERENCE MANUAL

`IDLE.BOUNCING.BOX`, which is initially a bitmap of the Venue logo. `WAIT` defaults to 1000 (one second).

Saving Virtual Memory State

Medley storage allocation occurs within a virtual memory space that is usually much larger than the physical memory on the computer. The virtual memory is stored as a large file on the computer's hard disk, called the virtual memory file. Medley controls the swapping of pages between this file and the real memory, swapping in virtual memory pages as they are accessed, and swapping out pages that have been modified. At any moment, the total state of the Medley virtual memory is stored partially in the virtual memory file, and partially in the real physical memory.

Medley provides facilities for saving the total state of the virtual memory, either on the virtual memory file, or in a file on an arbitrary file device. The function `LOGOUT` is used to write all altered (dirty) pages from the real memory to the virtual memory file and stop Medley, so that Medley can be restarted from the state of the `LOGOUT`. `SAVEVM` updates the virtual memory file without stopping Medley, which puts the virtual memory file into a consistent state (temporarily), so it could be restarted if the system crashes. `SYSOUT` and `MAKESYS` are used to save a copy of the total virtual memory state on a file, which can be loaded into another machine to restore Medley's state. `VMEM.PURE.STATE` can be used to "freeze" the current state of the virtual memory, so that Medley will come up in that state if it is restarted.

(LOGOUT *FAST*)

[Function]

Stops Medley, and returns control to the operating system. If Medley is restarted, it should come up in the same state as when the `LOGOUT` was called. `LOGOUT` will not affect the state of open files.

`LOGOUT` writes out all altered pages from real memory to the virtual memory file. If *FAST* is `T`, Medley is stopped without updating the virtual memory file. Note that after doing `(LOGOUT T)` it will not be possible to restart Medley from the point of the `LOGOUT`, and it may not be possible to restart it at all. Typing `(LOGOUT T)` is preferable to just booting the machine, because it also does other cleanup operations (closing network connections, etc.).

If *FAST* is the symbol `?`, `LOGOUT` acts like `FLG = T` if the virtual memory file is consistent, otherwise it acts like `FLG = NIL`. This insures that the virtual memory image can be restarted as of *some* previous state, not necessarily as of the `LOGOUT`.

(SAVEVM —)

[Function]

This function is similar to logging out and continuing, but faster. It takes about as long as a logout, which can be as brief as 10 seconds or so if you have already written out most of your dirty pages by virtue of being idle a while. After the `SAVEVM`, and until the pagefault handler is next forced to write out a dirty page, your virtual memory image will be continuable (as of the `SAVEVM`) should there be a system crash or other disaster.

If the system has been idle long enough (no keyboard or mouse activity), there are dirty pages to be written, and there are few enough dirty pages left to write that a `SAVEVM` would be quick, `SAVEVM` is automatically called. When `SAVEVM` is called automatically, the cursor is changed to a special cursor: ^{SAV}⏏, stored in the variable `SAVINGCURSOR`. You

MISCELLANEOUS

can control how often SAVEVM is automatically called by setting the following two global variables:

SAVEVMWAIT	[Variable]
SAVEVMMAX	[Variable]

The system will call SAVEVM after being idle for SAVEVMWAIT seconds (initially 300) if there are fewer than SAVEVMMAX pages dirty (initially 600). These values are fairly conservative. If you want to be extremely wary, you can set SAVEVMWAIT = 0 and SAVEVMMAX = 10000, in which case SAVEVM will be called the first chance available after the first dirty page has been written.

The function SYSOUT saves the current state of Medley's virtual memory on a file, known as a "sysout file", or simply a "sysout". The file package can be used to save particular function definitions and other arbitrary objects on files, but SYSOUT saves the *total* state of the system. This capability can be useful in many situations: for creating customized systems for other people to use, or to save a particular system state for debugging purposes. Note that a sysout file can be very large (thousands of pages), and can take a long time to create, so it is not to be done lightly. The file produced by SYSOUT can be loaded into Medley's virtual memory and restarted to restore the virtual memory to the exact state that it had when the sysout file was made. The exact method of loading a sysout depend on the implementation. For more information on loading sysout files, see the users guide for your computer.

(SYSOUT FILE)	[Function]

Saves the current state of Medley's virtual memory on the file *FILE*, in a form that can be subsequently restarted. The current state of program execution is saved in the sysout file, so (PROGN (SYSOUT 'FOO) (PRINT 'HELLO)) will cause HELLO to be printed after the sysout file is restarted.

SYSOUT can take a very long time (ten or fifteen minutes), particularly when storing a file on a remote file server. To display some indication that something is happening, the cursor is changed to:  Also, as the sysout file is being written, the cursor is inverted line by line, to show that activity is taking place, and how much of the sysout has completed. For example, after the SYSOUT is about two-thirds done, the cursor would look like: . The SYSOUT cursor is stored in the variable SYSOUTCURSOR.

If *FILE* is non-NIL, the variable SYSOUTFILE is set to the body of *FILE*. If *FILE* is NIL, then the value of SYSOUTFILE instead. Therefore, (SYSOUT) will save the current state on the next higher version of a file with the same name as the previous SYSOUT. Also, if the extension for *FILE* is not specified, the value of SYSOUT.EXT is used. SYSOUT sets SYSOUTDATE (see the System Version Information section below) to (DATE), the time and date that the SYSOUT was performed.

If SYSOUT was not able to create the sysout file, because of disk or computer error, or because there was not enough space on the directory, SYSOUT returns NIL. Otherwise it returns the full file name of *FILE*.

Actually, SYSOUT "returns" twice; when the sysout file is first created, and when it is subsequently restarted. In the latter case, SYSOUT returns a list whose CAR is the full file

MEDLEY REFERENCE MANUAL

name of *FILE*. For example, (if (LISTP (SYSOUT 'FOO)) then (PRINT 'HELLO)) will cause HELLO to be printed when the sysout file is restarted, but not when SYSOUT is initially performed.

Note: SYSOUT does not save the state of any open files. Use WHENCLOSE (see the Closing and Reopening Files section in Chapter 24) to associate certain operations with open files so that when a SYSOUT is started up, these files will be reopened, and file pointers repositioned.

SYSOUT evaluates the expressions on BEFORESYSOUTFORMS (see also AROUNDEXITFNS) before creating the sysout file. This variable initially includes expressions to:

1. Set the variables SYSOUTDATE and SYSOUTFILE as described above
2. Default the sysout file name *FILE* according to the values of the variables SYSOUTFILE and SYSOUT.EXT, as described above
3. Perform any necessary operations on open files as specified by calls to WHENCLOSE.

After a sysout file is restarted (but *not* when it is initially created), SYSOUT evaluates the expressions on AFTERSYSOUTFORMS (see also AROUNDEXITFNS). This initially includes expressions to:

1. Perform any necessary operations on previously-opened files as specified by calls to WHENCLOSE
2. Possibly print a message, as determined by the value of SYSOUTGAG (see below)
3. Call SETINITIALS to reset the initials used for time-stamping (see the Time Stamps section of Chapter 16).

AROUNDEXITFNS

[Variable]

This variable provides a way to “advise” the system on cleanup and restoration activities to perform around LOGOUT, SYSOUT, MAKESYS and SAVEVM; It subsumes the functionality of BEFORESYSOUTFORMS, AFTERLOGOUTFORMS, etc. Its value is a list of functions (names) to call around every “exit” of the system. Each function is called with one argument, a symbol indicating which particular event is occurring. The symbols are:

BEFORLOGOUT The system is about to perform a LOGOUT.

BEFORESYSOUT
BEFOREMAKESYS

BEFORESAVEVM The system is about to perform a SYSOUT, MAKESYS or a SAVEVM.

AFTERLOGOUT
AFTERSYSOUT
AFTERMAKESYS

AFTERSAVEVM The system is starting up an image that was saved by performing a LOGOUT, SYSOUT, etc.

AFTERDOSYSOUT
AFTERDOMAKESYS

MISCELLANEOUS

AFTERDOSAVEVM The system just made a copy of the virtual memory and saved it to disk. The image continues to run. These events only exist to allow you to negate the effects of saving a copy of the virtual memory.

SYSOUTGAG [Variable]

The value of SYSOUTGAG determines what is printed when a sysout file is restarted. If the value of SYSOUTGAG is a list, the list is evaluated, and no additional message is printed. This allows you to print a message. If SYSOUTGAG is non-NIL and not a list, no message is printed. Finally, if SYSOUTGAG is NIL (its initial value), and the sysout file is being restarted by the same user that made the sysout originally, you are greeted by printing the value of HERALDSTRING (see below) followed by a greeting message. If the sysout file was made by a different user, a message is printed, warning that the currently-loaded user init file may be incorrect for the current user (see the Greeting and Initialization Files section above).

(MAKESYS FILE NAME) [Function]

Used to store a new Medley system on the “makesys file” *FILE*. Like SYSOUT, but before the file is made, the system is “initialized” by undoing the greet history, and clearing the display.

When the system is first started up, a “herald” is printed identifying the system, typically “Medley-XX DATE . . .”. If *NAME* is non-NIL, MAKESYS will use it instead of Medley-XX in the herald. MAKESYS sets HERALDSTRING to the herald string printed out.

MAKESYS also sets the variable MAKESYSDATE (see the next section below) to (DATE), i.e. the time and date the system was made.

Medley contains a routine that writes out dirty pages of the virtual memory during I/O wait, assuming that swapping has caused at least one dirty page to be written back into the virtual memory file (making it non-continuable). The frequency with which this routine runs is determined by:

BACKGROUNDPAGEFREQ [Variable]

This variable determines how often the routine that writes out dirty pages is run. The *higher* BACKGROUNDPAGEFREQ is set, the *greater* the time between running the dirty page writing routine. Initially it is set to 4. The lower BACKGROUNDPAGEFREQ is set, the less responsiveness you get at typein, so it may not be desirable to set it all the way down to 1.

(VMEM.PURE.STATE X) [NoSpread Function]

VMEM.PURE.STATE modifies the swapper’s page replacement algorithm so that dirty pages are only written at the end of the virtual memory backing file. This “freezes” a given virtual memory state, so that Medley will come up in that state whenever it is restarted. This can be used to set up a “clean” environment on a pool machine, allowing each user to initialize the system simply by rebooting the computer.

The way to use VMEM.PURE.STATE is to set up the environment as you wish it to be “frozen,” evaluate (VMEM.PURE.STATE T), and then call any function that saves the virtual memory state (LOGOUT, SAVEVM, SYSOUT, or MAKESYS). From that point on,

MEDLEY REFERENCE MANUAL

whenever the system is restarted, it will return to the state as of the saving operation. Future LOGOUT, SAVEVM, etc. operations will not reset this state.

Note: When the system is running in “pure state” mode, it uses a significant amount of the virtual memory backing file to save the “frozen” memory image, so this will reduce the amount of virtual memory space available for use.

(VMEM.PURE.STATE) returns T if the system is running in “pure state” mode, NIL otherwise.

(**REALMEMORYSIZE**) [Function]

Returns the number of real memory pages in the computer.

(**VMEMSIZE**) [Function]

Returns the number of pages in use in the virtual memory. This is the roughly the same as the number of pages required to make a sysout file on the local disk (see SYSOUT, above).

(**LASTVMEMFILEPAGE**) [Variable]

Value is the total size of the virtual memory backing file. This variable is set when the system is started. You should *not* set it..

Note: When the virtual memory expands to the point where the virtual memory backing file is almost full, a break will occur with the warning message “Your virtual memory backing file is almost full. Save your work & reload asap.” When this happens, it is strongly suggested that you save any important work and reload the system. If you continue working past this point, the system will start slowing down considerably, and it will eventually stop working.

System Version Information

Medley runs on a number of different machines, with many possible hardware configurations. There have been a number of different releases of the Medley software. These facts make it difficult to answer the important question “what software/hardware environment are you running?” when reporting bugs. The following functions allow the novice to collect this information.

(**PRINT-LISP-INFO** *STREAM FILESTRING*) [NoSpread Function]

Prints out a summary of the software and hardware environment that Medley is running in, and a list of all loaded patch files:

```
Venue Medley version
Medley 2.0 sysout of 7-Oct-92 15:18:52 on mips,
Emulator created: 20-Nov-92, memory size: 0,
machine d022899 mo
based on Envos Medley version Medley 2.0 sysout of 7-Oct-
92 15:18:52,
Make-init dates: 7-Oct-92 11:07:17, 7-Oct-92 11:26:22
Patch files: NIL
```

STREAM is the stream used to print the summary. If not given, it defaults to T.

MISCELLANEOUS

FILESTRING is a string used to determine what loaded files should be listed as “patch files.” All file names on *LOADEDFILELIST* (see the Noticing Files section of Chapter 17) that have *FILESTRING* as a substring as listed. If *FILESTRING* is not given, it defaults to the string “PATCH”.

(CL:LISP-IMPLEMENTATION-TYPE) [Function]

Returns a string identifying the type of implementation that is running, e.g., “Medley”.

(CL:LISP-IMPLEMENTATION-VERSION) [Function]

Returns a string identifying the version that is running. Currently gives the system name and date, e.g., “KOTO of 10-Sep-85 08:25:46”.

This uses the variables *MAKESYSNAME* and *MAKESYSDATE* (below), so it will change if you use *MAKESYS* (see the Saving Virtual Memory State section above) to create a custom sysout file, or explicitly changes these variables.

(CL:SOFTWARE-TYPE) [Function]

Returns a string identifying the operating system that Interlisp is running under. Currently returns the string “Envos Medley”.

(CL:SOFTWARE-VERSION) [Function]

Returns a string identifying the version of the operating system that Interlisp is running under. Currently, this returns the date that the Medley release was originally created, so it doesn’t change over *MAKESYS* or *SYSOUT*.

(CL:MACHINE-TYPE) [Function]

Returns a string identifying the type of computer hardware that Medley is running on, i.e., “1108”, “1132”, “1186”, “mips”, etc.

(CL:MACHINE-VERSION) [Function]

Returns a string identifying the version of the computer hardware that Medley is running on. Currently returns the microcode version and real memory size.

(CL:MACHINE-INSTANCE) [Function]

Returns a string identifying the particular machine that Medley is running on. Currently returns the machine’s NS address.

(CL:SHORT-SITE-NAME) [Function]

Returns a short string identifying the site where the machine is located. Currently returns (*ETHERHOSTNAME*) (if non-NIL) or the string “unknown”.

(CL:LONG-SITE-NAME) [Function]

Returns a long string identifying the site where the machine is located. Currently returns the same as *SHORT-SITE-NAME*.

MEDLEY REFERENCE MANUAL

SYSOUTDATE [Variable]

Value is set by `SYSOUT` (see the Saving Virtual Memory State section above) to the date before generating a virtual memory image file.

MAKESYSDATE [Variable]

Value is set by `MAKESYS` (see the Saving Virtual Memory State section above) to the date before generating a virtual memory image file.

MAKESYSNAME [Variable]

Value is a symbol identifying the release name of the current Medley system, e.g., `:MEDLEY`.

(SYSTEMTYPE) [Function]

Allows programmers to write system-dependent code. `SYSTEMTYPE` returns a symbol corresponding to the implementation of Interlisp: `D` (for Medley), `TOPS-20`, `TENEX`, `JERICO`, or `VAX`.

In Medley, `(SELECTQ (SYSTEMTYPE) ...)` expressions are expanded at compile time so that this is an effective way to perform conditional compilation.

(MACHINETYPE) [Function]

Returns the type of machine that Medley is running on: either `DORADO` (for the Xerox 1132), `DOLPHIN` (for the Xerox 1100), `DANDELION` (for the Xerox 1108), `DOVE` (for the Xerox 1186), or `MAIKO` (for Unix, DOS, etc).

Date And Time Functions

(DATE *FORMAT*) [Function]

Returns the current date and time as a string with format `"DD-MM-YY HH:MM:SS"`, where *DD* is day, *MM* is month, *YY* year, *HH* hours, *MM* minutes, *SS* seconds, e.g., `"7-Jun-85 15:49:34"`.

If *FORMAT* is a date format as returned by `DATEFORMAT` (below), it is used to modify the format of the date string returned by `DATE`.

(IDATE *STR*) [Function]

STR is a date and time string. `IDATE` returns *STR* converted to a number such that if *DATE1* is before (earlier than) *DATE2*, then `(IDATE DATE1) < (IDATE DATE2)`. If *STR* is `NIL`, the current date and time is used.

Different Interlisp implementations can have different internal date formats. However, `IDATE` always has the essential property that `(IDATE X)` is less than `(IDATE Y)` if *X* is before *Y*, and `(IDATE (GDATE N))` equals *N*. Programs which do arithmetic other than numerical comparisons between `IDATE` numbers may not work when moved from one implementation to another.

MISCELLANEOUS

Generally, it is possible to increment an `IDATE` number by an integral number of days by computing a “1 day” constant, the difference between two convenient `IDATES`, e.g. `(IDIFFERENCE (IDATE "2-JAN-80 12:00") (IDATE "1-JAN-80 12:00"))`. This “1 day” constant can be evaluated at compile time.

`IDATE` is guaranteed to accept as input the dates that `DATE` will output. It will ignore the parenthesized day of the week (if any). `IDATE` also correctly handles time zone specifications for those time zones registered in the list `TIME.ZONES` (below).

(GDATE DATE FRMAT —) [Function]

Like `DATE`, except that `DATE` can be a number in internal date and time format as returned by `IDATE`. If `DATE` is `NIL`, the current time and date is used.

(DATEFORMAT KEY ... KEY) [NLambda NoSpread Function]

`DATEFORMAT` returns a date format suitable as a parameter to `DATE` and `GDATE`. `KEY ... KEY` are a set of keywords (unevaluated). Each keyword affects the format of the date independently (except for `SLASHES` and `SPACES`). If the date returned by `(DATE)` with the default formatting was 7-Jun-85 15:49:34, the keywords would affect the formatting as follows:

<code>NO.DATE</code>	Doesn't include the date information, e.g. "15:49:34".
<code>NUMBER.OF.MONTH</code>	Shows the month as a number instead of a name, e.g. "7-06-85 15:49:34".
<code>YEAR.LONG</code>	Prints the year using four digits, e.g. "7-Jun-1985 15:49:34".
<code>SLASHES</code>	Separates the day, month, and year fields with slashes, e.g. "7/Jun/85 15:49:34".
<code>SPACES</code>	Separates the day, month, and year fields with spaces, e.g. "7 Jun 85 15:49:34".
<code>NO.LEADING.SPACES</code>	By default, the day field will always be two characters long. If <code>NO.LEADING.SPACES</code> is specified, the day field will be one character for dates earlier than the 10th, e.g. "7-Jun-85 15:49:34" instead of "7-Jun-85 15:49:34".
<code>NO.TIME</code>	Doesn't include the time information, e.g. "7-Jun-85".
<code>TIME.ZONE</code>	Includes the time zone in the time specification, e.g. "7-Jun-85".
<code>NO.SECONDS</code>	Doesn't include the seconds, e.g. "7-Jun-85 15:49".
<code>DAY.OF.WEEK</code>	Includes the day of the week in the time specification, e.g. "7-Jun-85 15:49:34 PDT (Friday)".

MEDLEY REFERENCE MANUAL

DAY.SHORT If **DAY.OF.WEEK** is specified to include the day of the week, the week day is shortened to the first three letters, e.g. "7-Jun-85 15:49:34 PDT (Fri)". Note that **DAY.SHORT** has no effect unless **DAY.OF.WEEK** is also specified.

(CLOCK N —) [Function]

If $N = 0$, **CLOCK** returns the current value of the time of day clock i.e., the number of milliseconds since last system start up.

If $N = 1$, returns the value of the time of day clock when you started up this Interlisp, i.e., difference between **(CLOCK 0)** and **(CLOCK 1)** is number of milliseconds (real time) since this Interlisp system was started.

If $N = 2$, returns the number of milliseconds of *compute* time since user started up this Interlisp (garbage collection time is subtracted off).

If $N = 3$, returns the number of milliseconds of compute time spent in garbage collections (all types).

(SETTIME DT) [Function]

Sets the internal time-of-day clock. If $DT = \text{NIL}$, **SETTIME** attempts to get the time from the communications net; if it fails, you are prompted for the time. If DT is a string in a form that **IDATE** recognizes, it is used to set the time.

The following variables are used to interpret times in different time zones. **\TimeZoneComp**, **\BeginDST**, and **\EndDST** are normally set automatically if your machine is connected to a network with a time server. For standalone machines, it may be necessary to set them by hand (or in your init file, see the first section of this chapter) if you are not in the Pacific time zone.

TIME.ZONES [Variable]

Value is an association list that associates time zone specifications (PDT, EST, GMT, etc.) with the number of hours west of Greenwich (negative if east). If the time zone specification is a single letter, it is appended to "DT" or "ST" depending on whether daylight saving time is in effect. Initially set to:

((8 . P) (7 . M) (6 . C) (5 . E) (0 . GMT))

This list is used by **DATE** and **GDATE** when generating a date with the **TIME.ZONE** format is specified, and by **IDATE** when parsing dates.

\TimeZoneComp [Variable]

This variable should be initialized to the number of hours west of Greenwich (negative if east). For the U.S. west coast it is 8. For the east coast it is 5.

\BeginDST [Variable]

\EndDST [Variable]

\BeginDST is the day of the year on or before which Daylight Savings Time takes effect (i.e., the Sunday on or immediately preceding this day); **\EndDST** is the day on or before which Daylight Savings Time ends. Days are numbered with 1 being January 1, and

counting the days as for a leap year. In the USA where Daylight Savings Time is observed, \BeginDST = 121 and \EndDST = 305. In a region where Daylight Savings Time is not observed at all, set \BeginDST to 367.

Timers and Duration Functions

Often one needs to loop over some code, stopping when a certain interval of time has passed. Some systems provide an “alarm clock” facility, which provides an asynchronous interrupt when a time interval runs out. This is not particularly feasible in the current Medley environment, so the following facilities are supplied for efficiently testing for the expiration of a time interval in a loop context.

Three functions are provided: `SETUPTIMER`, `SETUPTIMER.DATE`, and `TIMEREXPIRED?`. There are also several new i.s.oprs: `forDuration`, `during`, `untilDate`, `timerUnits`, `usingTimer`, and `resourceName` (reasonable variations on upper/lower case are permissible).

These functions use an object called a timer, which encodes a future clock time at which a signal is desired. A timer is constructed by the functions `SETUPTIMER` and `SETUPTIMER.DATE`, and is created with a basic clock “unit” selected from among `SECONDS`, `MILLISECONDS`, or `TICKS`. The first two timer units provide a machine/system independent interface, and the latter provides access to the “real”, basic strobe unit of the machine’s clock on which the program is running. The default unit is `MILLISECONDS`.

Currently, the `TICKS` unit depends on what machine Medley is running on. The Xerox 1132 has about 1680 ticks per millisecond; the Xerox 1108 has about 34.746 ticks per millisecond; the Xerox 1185 and 1186 have about 62.5 ticks per millisecond. The advantage of using `TICKS` rather than one of the uniform interfaces is primarily speed; e.g., it may take over 400 microseconds to read the milliseconds clock (a software facility that uses the real clock), whereas reading the real clock itself may take less than ten microseconds. The disadvantage of the `TICKS` unit is its short roll-over interval (about 20 minutes) compared to the `MILLISECONDS` roll-over interval (about two weeks), and also the dependency on particular machine parameters.

(**SETUPTIMER** *INTERVAL* *OldTimer?* *timerUnits* *intervalUnits*) [Function]

`SETUPTIMER` returns a timer that will “go off” (as tested by `TIMEREXPIRED?`) after a specified time-interval measured from the current clock time. `SETUPTIMER` has one required and three optional arguments:

INTERVAL must be a integer specifying how long an interval is desired. *timerUnits* specifies the units of measure for the interval (defaults to `MILLISECONDS`).

If *OldTimer?* is a timer, it will be reused and returned, rather than allocating a new timer. *intervalUnits* specifies the units in which the *OldTimer?* is expressed (defaults to the value of *timerUnits*).

(**SETUPTIMER.DATE** *DTS* *OldTimer?*) [Function]

`SETUPTIMER.DATE` returns a timer (using the `SECONDS` time unit) that will “go off” at a specified date and time. *DTS* is a Date/Time string such as `IDATE` accepts (see the above section). If *OldTimer?* is a timer, it will be reused and returned, rather than allocating a new timer.

MEDLEY REFERENCE MANUAL

SETUPTIMER.DATE operates by first subtracting (IDATE) from (IDATE DTS), so there may be some large integer creation involved, even if *OLDTIMER?* is given.

(**TIMEREXPIRED?** *TIMER ClockValue.or.timerUnits*) [Function]

If *TIMER* is a timer, and *ClockValue.or.timerUnits* is the time-unit of *TIMER*, **TIMEREXPIRED?** returns true if *TIMER* has “gone off”.

ClockValue.or.timerUnits can also be a timer, in which case **TIMEREXPIRED?** compares the two timers (which must be in the same timer units). If *X* and *Y* are timers, then (**TIMEREXPIRED?** *X Y*) is true if *X* is set for an *earlier* time than *Y*.

There are a number of i.s.oprs that make it easier to use timers in iterative statements (see the Iterative Statement section of Chapter 9). These i.s.oprs are given below in the “canonical” form, with the second “word” capitalized, but the all-caps and all-lower-case versions are also acceptable.

forDuration *INTERVAL* [I.S. Operator]

during *INTERVAL* [I.S. Operator]

INTERVAL is an integer specifying an interval of time during which the iterative statement will loop.

timerUnits *UNITS* [I.S. Operator]

UNITS specifies the time units of the *INTERVAL* specified in **forDuration**.

untilDate *DTS* [I.S. Operator]

DTS is a Date/Time string (such as *IDATE* accepts) specifying when the iterative statement should stop looping.

usingTimer *TIMER* [I.S. Operator]

If **usingTimer** is given, *TIMER* is reused as the timer for **forDuration** or **untilDate**, rather than creating a new timer. This can reduce allocation if one of these i.s.oprs is used within another loop.

resourceName *RESOURCE* [I.S. Operator]

RESOURCE specifies a resource name to be used as the timer storage (see the File Package Types section of Chapter 17). If *RESOURCE* = *T*, it will be converted to an internal name.

Some examples:

```
(during 6MONTHS timerUnits 'SECONDS
  until (TENANT-VACATED? HouseHolder)
  do (DISMISS <for-about-a-day>
    (HARRASS HouseHolder)
  finally (if (NOT (TENANT-VACATED? HouseHolder))
    then (EVICT-TENANT HouseHolder)))
```

This example shows that how is is possible to have two termination condition: when the time interval of 6MONTHS has elapsed, or when the predicate (TENANT-VACATED? HouseHolder) becomes true. Note that the “finally” clause is executed regardless of which termination condition caused it.

MISCELLANEOUS

Also note that since the millisecond clock will “roll over” about every two weeks, “6MONTHS” wouldn’t be an appropriate interval if the timer units were the default case, namely `MILLISECONDS`.

```
(do (forDuration (CONSTANT (ITIMES 10 24 60 60 1000))
    do (CARRY.ON.AS.USUAL)
    finally (PROMPTPRINT "Have you had your 10-day check-up?")))
```

This infinite loop breaks out with a warning message every 10 days. One could question whether the millisecond clock, which is used by default, is appropriate for this loop, since it rolls-over about every two weeks.

```
(SETQ \RandomTimer (SETUPTIMER 0))
(untilDate "31-DEC-83 23:59:59" usingTimer \RandomTimer
 when (WINNING?) do (RETURN)
 finally (ERROR "You’ve been losing this whole year!"))
```

Here is a usage of an explicit date for the time interval; also, some storage has been squirreled away (as the value of `\RandomTimer`) for use by the call to `SETUPTIMER` in this loop.

```
(forDuration SOMEINTERVAL
 resourceName \INNERLOOPBOX
 timerunits 'TICKS
 do (CRITICAL.INNER.LOOP))
```

For this loop, you don’t want any `CONS`ing to take place, so `\INNERLOOPBOX` is defined as a resource which “caches” a timer cell (if it isn’t already so defined), and wraps the entire statement in a `WITH-RESOURCES` call. Furthermore, a time unit of `TICKS` is specified, for lower overhead in this critical inner loop. In fact specifying a resourceName of `T` is the same as specifying it to be `\ForDurationOfBox`; this is just a simpler way to specify that a resource is wanted, without having to think up a name.

Resources

Medley is based on the use of a storage-management system which allocates memory space for new data objects, and automatically reclaims the space when no longer in use. More generally, Medley manages shared “resources”, such as files, semaphors for processes, etc. The protocols for allocating and freeing such resources resemble those of ordinary storage management.

Sometimes you need to explicitly manage the allocation of resources. You may want the efficiency of explicit reclamation of certain temporary data; or it may be expensive to initialize a complex data object; or there may be an application that must not allocate new cells during some critical section of code.

The file manager type `RESOURCES` is available to help with the definition and usage of such classes of data; the definition of a `RESOURCE` specifies prototype code to do the basic management operations. The file manager command `RESOURCES` is used to save such definitions on files, and `INITRESOURCES` (see the Miscellaneous File Manager Commands section of Chapter 17) causes the initialization code to be output.

The basic needs of resource management are:

1. Obtaining a data item from the Lisp memory management system and configuring it to be a totally new instance of the resource in question
2. Freeing up an instance which is no longer needed

MEDLEY REFERENCE MANUAL

3. Getting an instance of the resource for temporary usage (whether “fresh” or a formerly freed-up instance)
4. Setting up any prerequisite global data structures and variables

A resources definition consists of four “methods”: `INIT`, `NEW`, `GET`, and `FREE`; each “method” is a form that will specialize the definition for four corresponding user-level macros `INITRESOURCE`, `NEWRESOURCE`, `GETRESOURCE`, and `FREERESOURCE`. `PUTDEF` is used to make a resources definition, and the four components are specified in a proplist:

```
(PUTDEF
  'RESOURCENAME
  'RESOURCES
  ' (NEW  NEW-INSTANCE-GENERATION-CODE
      FREE FREEING-UP-CODE
      GET  GET-INSTANCE-CODE
      INIT INITIALIZATION-CODE) )
```

Each of the `xxx-CODE` forms is a form that will appear as if it were the body of a substitution macro definition for the corresponding macro (see the discussion on the macros below).

A Simple Example

Suppose one has several pieces of code which use a 256-character string as a scratch string. One could simply generate a new string each time, but that would be inefficient if done repeatedly. If you can guarantee that there are no re-entrant uses of the scratch string, then it could simply be stored in a global variable. However, if the code might be re-entrant on occasion, the program has to take precautions that two programs do not use the same scratch string at the same time. (This consideration becomes very important in a multi-process environment. It is hard to guarantee that two processes won't be running the same code at the same time, without using elaborate locks.) A typical tactic would be to store the scratch string in a global variable, and set the variable to `NIL` whenever the string is in use (so that re-entrant usages would know to get a “new” instance). For example, assuming the global variable `TEMPSTRINGBUFFER` is initialized to `NIL`:

```
[DEFINEQ (WITHSTRING NIL
  (PROG ((BUF (OR (PROG1 TEMPSTRINGBUFFER
    (SETQ TEMPSTRINGBUFFER NIL) )
    (ALLOCSTRING 256) ) ) )
```

... use the scratch string in the variable `BUF` ...

```
(SETQ TEMPSTRINGBUFFER BUF)
(RETURN]
```

Here, the basic elements of a “resource” usage may be seen:

1. A call `(ALLOCSTRING 256)` allocates fresh instances of “buffer”
2. After usage is completed the instance is returned to the “free” state, by putting it back in the global variable `TEMPSTRINGBUFFER` where a subsequent call will find it
3. The prog-binding of `BUF` will get an existing instance of a string buffer if there is one -- otherwise it will get a new instance which will later be available for reuse
4. Some initialization is performed before usage of the resource (in this case, it is the setting of the global variable `TEMPSTRINGBUFFER`).

Given the following resources definition:

```
(PUTDEF
  'STRINGBUFFER
  'RESOURCES
  '(NEW (ALLOCSTRING 256)
    FREE (SETQ TEMPSTRINGBUFFER (PROG1 . ARGS))
    GET (OR (PROG1 TEMPSTRINGBUFFER
      (SETQ TEMPSTRINGBUFFER NIL))
      (NEWRESOURCE TEMPSTRINGBUFFER)))
    INIT (SETQ TEMPSTRINGBUFFER NIL)))
```

we could then redo the example above as

```
(DEFINEQ (WITHSTRING NIL
  (PROG ((BUF (GETRESOURCE STRINGBUFFER)))
```

... use the string in the variable BUF ...

```
(FREERESOURCE STRINGBUFFER BUF)
(RETURN]
```

The advantage of doing the coding this way is that the resource management part of WITHSTRING is fully contained in the expansions of GETRESOURCE and FREERESOURCE, and thus there is no confusion between what is WITHSTRING code and what is resource management code. This particular advantage will be multiplied if there are other functions which need a “temporary” string buffer; and of course, the resultant modularity makes it much easier to contemplate minor variations on, as well as multiple clients of, the STRINGBUFFER resource.

In fact, the scenario just shown above in the WITHSTRING example is so commonly useful that an abbreviation has been added; if a resources definition is made with **only** a NEW method, then appropriate FREE, GET, and INIT methods will be inferred, along with a coordinated globalvar, to be parallel to the above definition. So the above definition could be more simply written

```
(PUTDEF 'STRINGBUFFER
  'RESOURCES
  '(NEW (ALLOCSTRING 256)))
```

and everything would work the same.

The macro WITH-RESOURCES simplifies the common scoping case, where at the beginning of some piece of code, there are one or more GETRESOURCE calls the results of which are each bound to a lambda variable; and at the ending of that code a FREERESOURCE call is done on each instance. Since the resources are locally bound to variables with the same name as the resource itself, the definition for WITHSTRING then simplifies to

```
(DEFINEQ (WITHSTRING NIL
  (WITH-RESOURCES (STRINGBUFFER)
```

... use the string in the variable STRINGBUFFER ...]

Trade-offs in More Complicated Cases

This simple example presumes that the various functions which use the resource are generally not re-entrant. While an occasional re-entrant use will be handled correctly (another example of the resource will simply be created), if this were to happen too often, then many of the resource requests will create and throw away new objects, which defeats one of the major purposes of using resources. A slightly more complex GET and FREE method can be of much benefit in maintaining a pool of available

MEDLEY REFERENCE MANUAL

resources; if the resource were defined to maintain a list of “free” instances, then the GET method could simply take one off the list and the FREE method could just push it back onto the list. In this simple example, the SETQ in the FREE method defined above would just become a “push”, and the first clause of the GET method would just be (pop TEMPSTRINGBUFFER)

A word of caution: if the datatype of the resource is something very small that Medley is “good” at allocating and reclaiming, then explicit user storage management will probably not do much better than the combination of cons/createcell and the garbage collector. This would especially be so if more complicated GET and FREE methods were to be used, since their overhead would be closer to that of the built-in system facilities. Finally, it must be considered whether retaining multiple instances of the resource is a net gain; if the re-entrant case is truly rare, it may be more worthwhile to retain at most one instance, and simply let the instances created by the rarely-used case be reclaimed in the normal course of garbage collection.

Macros for Accessing Resources

Four user-level macros are defined for accessing resources:

(NEWRESOURCE <i>RESOURCE</i> NAME . <i>ARGS</i>)	[Macro]
(FREERESOURCE <i>RESOURCE</i> NAME . <i>ARGS</i>)	[Macro]
(GETRESOURCE <i>RESOURCE</i> NAME . <i>ARGS</i>)	[Macro]
(INITRESOURCE <i>RESOURCE</i> NAME . <i>ARGS</i>)	[Macro]

Each of these macros behave as if they were defined as a substitution macro of the form

((*RESOURCE*NAME . *ARGS*) *MACROBODY*)

where the expression *MACROBODY* is selected by using the “code” supplied by the corresponding method from the *RESOURCE*NAME definition.

Note that it is possible to pass “arguments” to your resource allocation macros. For example, if the GET method for the resource FOO is (GETFOO . *ARGS*), then (GETRESOURCE FOO *X Y*) is transformed into (GETFOO *X Y*). This form was used in the FREE method of the STRINGBUFFER resource described above, to pass the old STRINGBUFFER object to be freed.

(WITH-RESOURCES (<i>RESOURCE RESOURCE ...</i>) <i>FORM FORM ...</i>)	[Macro]

The WITH-RESOURCES macro binds lambda variables of the same name as the resources (for each of the resources *RESOURCE* , *RESOURCE* , etc.) to the result of the GETRESOURCE macro; then executes the forms *FORM* , *FORM* , etc., does a FREERESOURCE on each instance, and returns the value of the last form (evaluated and saved before the FREERESOURCES).

Note: (WITH-RESOURCES *RESOURCE ...*) is interpreted the same as (WITH-RESOURCES (*RESOURCE*) *...*). Also, the singular name WITH-RESOURCE is accepted as a synonym for WITH-RESOURCES.

Saving Resources in a File

Resources definitions may be saved on files using the RESOURCES file package command (see the Miscellaneous File Package Commands section of Chapter 17). Typically, one only needs the full definition available when compiling or interpreting the code, so it is appropriate to put the file package command in a (DECLARE: EVAL@COMPILE DONTCOPY ...) declaration, just as one might

MISCELLANEOUS

do for a `RECORDS` declaration. But just as certain record declarations need *some* initialization in the run-time environment, so do most resources. This initialization is specified by the resource's `INIT` method, which is executed automatically when the resource is defined by the `PUTDEF` output by the `RESOURCES` command. However, if the `RESOURCES` command is in a `DONTCOPY` expression and thus is not included in the compiled file, then it is necessary to include a separate `INITRESOURCES` command (see the Miscellaneous File Manager Commands section of Chapter 17) in the filecoms to insure that the resource is properly initialized.

MEDLEY REFERENCE MANUAL

[This page intentionally left blank]

MISCELLANEOUS

13. MEDLEY EXECUTIVES

In most Common Lisp implementations, there is a “top-level read-eval-print loop,” which reads an expression, evaluates it, and prints the results. In Medley, the Exec acts as the top-level loop, but does much more.

The Exec traps all `THROWS`, and recovers gracefully. It prints all values resulting from evaluation, on separate lines. (When zero values are returned, nothing is printed).

The Exec keeps track of your previous inputs, in the history list. Each entry you type creates a history event, which stores the input and its values.

It's easy to use the results of earlier events, redo an event, or recall an earlier input, edit it, and run it. This makes it much easier to get your work done.

Multiple Execs and the Exec's Type

Sometimes you need more than one Exec open at a time. It's easy to open as many as you need by using the right button background menu and selecting the kind of Exec you need. The Execs are differentiated from one another by their “names” in their title bars and by their prompts. For example, the second Exec you open may have a prompt like `2/50>` if it's the second Common Lisp Exec you've opened. Events in each Exec are placed on the global history list with their Exec number so the system can tell them apart.

Several variables are very important to an Exec since they control the format of reading and printing. Together these variables describe a type of exec, or its mode. Some standard bindings for the variables have been named to make mode setting easy. The names provide you with an Exec of the Common Lisp (`LISP`), Interlisp or Old Interlisp (`IL`), or Medley (`XCL`) type. An Exec's type is displayed in the title bar of its window:



```
Exec 2 (XCL)
2/53> *package*
#<Package XCL-USER>
2/56> *readtable*
#<ReadTable XCL/74,151670>
2/57> A
```

A Brief Example of Exec Interactions

The following dialogue contains examples and gives the flavor of the use of an Exec. The commands are described in greater detail in the following sections. For now, be sure to type these examples to an Exec whose `*PACKAGE*` is set to the `XCL-USER` package. The Exec that Medley starts up with is set to the `XCL-USER` package. Each prompt consists of an Exec number, an event number and a prompt character (“>” for Common Lisp and “←” for Interlisp).

MEDLEY REFERENCE MANUAL

```
Exec 2 (LISP)
2/1188> (SETQ FOO 5)
5
2/1189> (SETQ FOO 10)
10
2/1190> UNDO SETQ
SETQ undone.
2/1191> FOO
5
2/1192>
```

You have instructed the Exec to UNDO the previous event.

```
Exec 2 (LISP)
2/1192> SET(LST1 (A B C))
(A B C)
2/1195> (SETQ LST2 '(D E F))
(D E F)
2/1196> (MAPC #'(LAMBDA (X) (SETF (GET
X 'MYPROP) T)) LST1)
(A B C)
2/1197>
```

The Exec accepts input both in APPLY format (the SET) and EVAL format (the SETQ). In event 1196, you added a property MYPROP to the symbols A, B, and C.

```
Exec 2 (LISP)
2/1192> SET(LST1 (A B C))
(A B C)
2/1195> (SETQ LST2 '(D E F))
(D E F)
2/1196> (MAPC #'(LAMBDA (X) (SETF (GET
X 'MYPROP) T)) LST1)
(A B C)
2/1197> USE LST2 FOR LST1 IN 1196
(D E F)
2/1198>
```

You told the Exec to go back to event 1196, substitute LST2 for LST1, and then re-execute the expression.

```
Exec 2 (XCL)
NIL
2/48> (setf myhash (make-hash-table))
#<Hash-Table @ 361,117340>
2/49> (setf (gethash 'foo myhash)(string
'foo))
"F00"
2/50>
```

If STRING were computationally expensive (it isn't), you might be caching its value for later use.

```
Exec 2 (XCL)
2/48> (setf myhash (make-hash-table))
#<Hash-Table @ 361,117340>
2/49> (setf (gethash 'foo myhash)(string
'foo))
"F00"
2/50> use fie for foo in string
"FIE"
2/51>
```

You now decide you would like to redo the SETF with a different value. You can specify the event using any symbol in the expression.

```

Exec 2 (XCL)
2/68> ?? setf
                USE FIE FOR FOO IN STRING
                (SETF (GETHASH (QUOTE FIE) MYH
ASH) (STRING (QUOTE FIE)))
                "FIE"
2/68>_

```

Here you ask the Exec (using the ?? command) what it has on its history list for the last input. Since the event corresponds to a command, the Exec displays both the original command and the generated input.

You'll usually deal with the Exec at top level or in the debugger, where you type in expressions for evaluation, and see the values printed out. An Exec acts much like a standard Lisp top-level loop, but before it evaluates an input, it first adds it to the history list. If the operation is aborted or causes an error, the input is still available for you to modify or re-execute.

After updating the history list, the Exec executes the computation (i.e., evaluates the form or applies the function to its arguments), saves the value in the history-list entry for that input, and prints the result. Finally the Exec displays a prompt to show it's again ready for input.

Input Formats

The Exec accepts three forms of input: an expression to be evaluated (EVAL-format), a function-name and arguments to apply it to (APPLY-format), and Exec commands, as follows:

EVAL-format input If you type a single expression, either followed by a carriage-return, or, in the case of a list, terminated with balanced parenthesis, the expression is evaluated and the value is returned. For example, if the value of FOO is the list (A B C):

```

Exec 3 (XCL)
3/133> foo
(A B C)
3/134>

```

Similarly, if you type a Lisp expression, beginning with a left parenthesis and terminated by a matching right parenthesis, the form is simply passed to EVAL for evaluation. Notice that it is not necessary to type a carriage return at the end of such a form; the reader will supply one automatically. If a carriage-return is typed before the final matching right parenthesis or bracket, it is treated the same as a space, and input continues. The following examples are interpreted identically:

```

Exec 3 (XCL)
3/38> (+ 1 (* 2 3))
7
3/40> (+ 1 (*
2 3))
7
3/41>

```

APPLY-format input Often, you call functions with constant argument values, which would have to be quoted if you typed them in EVAL-format. For convenience, if you type a symbol immediately followed by a list, the symbol is APPLIED to the elements within the list, unevaluated. The input is terminated by the matching right parenthesis. For example, typing LOAD (FOO) is equivalent to typing (LOAD 'FOO), and GET (X

MEDLEY REFERENCE MANUAL

COLOR) is equivalent to (GET 'X 'COLOR). As a simple special case, a single right parenthesis is treated as a balanced set of parentheses, e.g. UNBREAK) is equivalent to UNBREAK()

The reader will only supply the “carriage return” automatically if no space appears between the initial symbol and the list that follows; if there is a space after the initial symbol on the line and the list that follows, the input is not terminated until you type a carriage return.

The Exec will not consider unparenthesized input with more than one argument to be in apply format, e.g.:

LIST (1) is apply format (executes after closing parenthesis is typed)

LIST (1) is apply format (second argument is a list, no trailing arguments given)

LIST ' (1) 2 3 is NOT apply format, arguments are evaluated

LIST 1 2 3 is NOT apply format, arguments are evaluated

LIST 1 not legal input: second argument is not a list

Note that APPLY-format input cannot be used for macros or special forms.

Exec commands The Exec recognizes a number of commands, which usually refer to past events on the history list. These commands are treated specially; for example, they may not be put on the history list. The format of a command is always a line beginning with the command name. (The Exec looks up the command name independent of package.) The remainder of the line, if any, is treated as “arguments” to the command. For example,

```
128> UNDO
mapc undone
129> UNDO (FOO --)
foo undone
```

are both valid command inputs.

Event Specification

Exec commands, like UNDO, frequently refer to previous events in the session’s history. All Exec commands use the same conventions and syntax for indicating which event(s) the command refers to. This section shows you the syntax used to specify previous events.

An event address identifies one event on the history list. For example, the event address 42 refers to the event with event number 42, and -2 refers to two events back in the current Exec. Usually, an event address will contain only one or two commands.

Event addresses can be concatenated. For example, if FOO refers to event N, FOO FIE will refer to the first event before event N which contains FIE.

The symbols used in event addresses (such as AND, F, etc.) are compared with STRING-EQUAL, so that it does not matter what the current package is when you type an event address symbol to an Exec.

MEDLEY EXECUTIVES

Specifications used below of the form *EventAddress* refer to event addresses, as described above. Since an event address may contain multiple words, the event address is parsed by searching for the words which delimit it. For example, in *EventAddress AND EventAddress*, the notation *EventAddress* corresponds to all words up to the AND in the event specification, and *EventAddress* to all words after the AND in the event specification.

Event addresses are interpreted as follows:

N (an integer) If N is positive, it refers to the event with event number N (no matter which Exec the event occurred in.) If N is negative, it always refers to the event -N events backwards, counting only events belonging to the current Exec.

F Specifies that the next object in the event address is to be searched for, regardless of what it is. For example, F -2 looks for an event containing -2.

FROM *EventAddress*

All events since *EventAddress*, inclusive. For example, if there is a single Exec and the current event is number 53, then FROM 49 specifies events 49, 50, 51, and 52. FROM includes events from *all* Execs.

ALL *EventAddress*

Specifies all events satisfying *EventAddress*. For example, ALL LOAD, ALL SUCHTHAT FOO-P.

empty If nothing is specified, it is the same as specifying -1, i.e., the last event in the current Exec.

EventSpec AND EventSpec AND ... AND EventSpec

Each of the is an event specification. The lists of events are concatenated. For example, REDO ALL MAPC AND ALL STRING AND 32 redoes all events containing MAPC, all containing STRING, and also event 32. Duplicate events are removed.

Exec Commands

You enter an Exec commands by typing the name of the command at the prompt. The name of an Exec command is not a symbol and therefore is not sensitive to the setting of the current package (the value of *PACKAGE*).

EventSpec is used to denote an event specification which in most cases will be either a specific event address (e.g., 42) or a relative one (e.g., -3). Unless specified otherwise, omitting *EventSpec* is the same as specifying *EventSpec* = -1. For example, REDO and REDO -1 are the same.

REDO *EventSpec*

[Exec command]

Redoes the event or events specified by *EventSpec*. For example, REDO 123 redoes the event numbered 123.

MEDLEY REFERENCE MANUAL

RETRY *EventSpec* [Exec command]

Like REDO but sets the debugger parameters so that any errors that occur while executing *EventSpec* will cause breaks.

USE *NEW* [FOR *OLD*] [IN *EventSpec*] [Exec command]

Substitutes *NEW* for *OLD* in the events specified by *EventSpec*, and redoes the result. *NEW* and *OLD* can include lists or symbols, etc.

For example, USE SIN (- X) FOR COS X IN -2 AND -1 will substitute SIN for every occurrence of COS in the previous two events, and substitute (- X) for every occurrence of X, and reexecute them. (The substitutions do not change the previous information saved about these events on the history list.)

If IN *EventSpec* is omitted, the first member of *OLD* is used to search for the appropriate event. For example, USE DEFAULTFONT FOR DEFLATFONT is equivalent to USE DEFAULTFONT FOR DEFLATFONT IN F DEFLATFONT. The F is inserted to handle the case where the first member of *OLD* could be interpreted as an event address command.

If *OLD* is omitted, substitution is for the “operator” in that command. For example FBOUNDPF (FF) followed by USE CALLS is equivalent to USE CALLS FOR FBOUNDPF IN -1.

If *OLD* is not found, USE will print a question mark, several spaces and the pattern that was not found. For example, if you specified USE Y FOR X IN 104 and X was not found, “X ?” is printed to the Exec.

You can also specify more than one substitution simultaneously as follows:

USE *NEW* FOR *OLD* AND ... AND *NEW* FOR *OLD* [IN *EventSpec*] [Exec command]

[The USE command is parsed by a small finite state parser to distinguish the expressions and arguments. For example, USE FOR FOR AND AND AND FOR FOR will be parsed correctly.]

Every USE command involves three pieces of information: the expressions to be substituted, the arguments to be substituted for, and an event specification that defines the input expression in which the substitution takes place. If the USE command has the same number of expressions as arguments, the substitution procedure is straightforward. For example, USE X Y FOR U V means substitute X for U and Y for V, and is equivalent to USE X FOR U AND Y FOR V.

However, the USE command also permits distributive substitutions for substituting several expressions for the same argument. For example, USE A B C FOR X means first substitute A for X then substitute B for X (in a new copy of the expression), then substitute C for X. The effect is the same as three separate USE commands.

Similarly, USE A B C FOR D AND X Y Z FOR W is equivalent to USE A FOR D AND X FOR W, followed by USE B FOR D AND Y FOR W, followed by USE C FOR D AND Z FOR W. USE A B C FOR D AND X FOR Y also corresponds to three substitutions, the first with A for D and X for Y, the second with B for D, and X for Y, and the third with C

MEDLEY EXECUTIVES

for D, and again X for Y. However, `USE A B C FOR D AND X Y FOR Z` is ambiguous and will cause an error.

Essentially, the `USE` command operates by proceeding from left to right handling each `AND` separately. Whenever the number of expressions exceeds the available expressions, multiple `USE` expressions are generated. Thus `USE A B C D FOR E F` means substitute A for E at the same time substituting B for F, then in another copy of the indicated expression, substitute C for E and D for F. This is also equivalent to `USE A C FOR E AND B D FOR F`.

The `USE` command correctly handles the situation where one of the old expressions is the same as one of the new ones, `USE X Y FOR Y X`, or `USE X FOR Y AND Y FOR X`.

? *NAME* [Exec command]

If *NAME* is not provided describes all available Exec commands by printing the name, argument list, and description of each. With *NAME*, only that command is described.

?? *EventSpec* [Exec command]

Prints the most recent event matching the given *EventSpec*. Without *EventSpec*, lists all entries on the history list from all execs, not necessarily in the order in which they occurred (since the list is in allocation order). If you haven't completed typing a command it will be listed as "<in progress>".

Note: Event numbers are allocated at the time the prompt is printed, except in the Old Interlisp exec where they are assigned at the end of type-in. This means that if activity occurs in another exec, the number printed next to the command is not necessarily the number associated with the event.

CONN *DIRECTORY* [Exec command]

Changes default pathname to *DIRECTORY*.

DA [Exec command]

Returns current date and time.

DIR *PATHNAME KEYWORDS* [Exec command]

Shows a directory listing for *PATHNAME* or the connected directory. If provided, *KEYWORDS* indicate information to be displayed for each file. Some keywords are: *AUTHOR*, *AU*, *CREATIONDATE*, *DA*, etc.

DO-EVENTS *INPUTS ENV* [Exec command]

DO-EVENTS is intended as a way of putting together several different events, which can include commands. It executes the multiple *INPUTS* as a single event. The values returned by the *DO-EVENTS* event are the concatenation of the values of the inputs. An input is not an *EventSpec*, but a call to a function or command. If *ENV* is provided it is a lexical environment in which all evaluations (functions and commands) will take place. Event specification in the *INPUTS* should be explicit, not relative, since referring to the last event will reinvoked the executing *DO-EVENTS* command.

MEDLEY REFERENCE MANUAL

FIX <i>EventSpec</i>	[Exec command]
Edits the specified event prior to re-executing it. If the number of characters in the fixed line is less than the variable <code>TTYINFIXLIMIT</code> then it will be edited using <code>TTYIN</code> , otherwise the Lisp editor is called via <code>EDITE</code> .	
FORGET <i>EventSpec</i>	[Exec command]
Erases UNDO information for the specified events.	
NAME <i>COMMAND-NAME ARGUMENTS EVENT-SPEC</i>	[Exec command]
Defines a new command, <i>COMMAND-NAME</i> , and its <i>ARGUMENTS</i> , containing the events in <i>EVENT-SPEC</i> .	
NDIR <i>PATHNAME KEYWORDS</i>	[Exec command]
Shows a directory listing for <i>PATHNAME</i> or the connected directory in abbreviated format. If provided, <i>KEYWORDS</i> indicate information to be displayed for each file. Some keywords are: <code>AUTHOR</code> , <code>AU</code> , <code>CREATIONDATE</code> , <code>DA</code> , etc.	
PL <i>SYMBOL</i>	[Exec command]
Prints the property list of <i>SYMBOL</i> in an easy to read format.	
REMEMBER <i>&REST EVENT-SPEC</i>	[Exec command]
Tells File Manager to remember type-in from specified event(s), <i>EVENT-SPEC</i> , as expressions to save.	
SHH <i>LINE</i>	[Exec command]
Executes <i>LINE</i> without history list processing.	
UNDO <i>EventSpec</i>	[Exec command]
Undoes the side effects of the specified event (see below under “Undoing”).	
PP <i>NAME TYPES</i>	[Exec command]
Shows (prettyprinted) the definitions for <i>NAME</i> specified by <i>TYPES</i> .	
SEE <i>FILES</i>	[Exec command]
Prints the contents of <i>FILES</i> in the Exec window, hiding comments.	
SEE* <i>FILES</i>	[Exec command]
Prints the contents of <i>FILES</i> in the Exec window, showing comments.	
TIME <i>FORM &KEY REPEAT &ENVIRONMENT ENV</i>	[Exec command]
Times the evaluation of <i>FORM</i> in the lexical environment <i>ENV</i> , repeating <i>REPEAT</i> number of times. Information is displayed in the Exec window.	
TY <i>FILES</i>	[Exec command]
Exactly like the <i>TYPE</i> Exec command.	

MEDLEY EXECUTIVES

TYPE *FILES*

[Exec command]

Prints the contents of *FILES* in the Exec window, hiding comments.

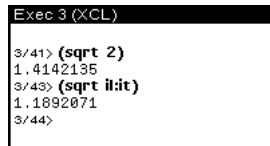
Variables

A number of variables are provided for convenience in the Exec.

IL:IT

[Variable]

Whenever an event is completed, the global value of the variable **IT** is reset to the event's value. For example,



```
Exec 3 (XCL)
3/41> (sqrt 2)
1.4142135
3/43> (sqrt il:it)
1.1892071
3/44>
```

Following a **??** command, **IL:IT** is set to the value of the last event printed. The inspector has an option for setting the variable **IL:IT** to the current selection or inspected object, as well. The variable **IL:IT** is global, and is shared among all Execs. **IL:IT** is a convenient mechanism for passing values from one process to another.

Note: **IT** is in the Interlisp package and these examples are intended for an Exec whose ***PACKAGE*** is set to **XCL-USER**. Thus, **IT** must be package qualified (the **IL:**).

The following variables are maintained independently by each Exec. (When a new Exec is started, the initial values are **NIL**, or, for a nested Exec, the value for the “parent” Exec. However, events executed under a nested Exec will not affect the parent values.)

CL:-

[Variable]

CL:+

[Variable]

CL:++

[Variable]

CL:+++

[Variable]

While a form is being evaluated by the Exec, the variable **CL:-** is bound to the form, **CL:+** is bound to the previous form, **CL:++** the one before, etc. If the input is in apply-format rather than eval-format, the value of the respective variable is just the function name.

CL:*

[Variable]

CL:**

[Variable]

CL:***

[Variable]

While a form is being evaluated by the Exec, the variable **CL:*** is bound to the (first) value returned by the last event, **CL:**** to the event before that, etc. The variable **CL:*** differs from **IT** in that **IT** is global while each separate Exec maintains its own copy of **CL:***, **CL:**** and **CL:*****. In addition, the history commands change **IT**, but only inputs that are retained on the history list can change **CL:***.

MEDLEY REFERENCE MANUAL

CL:/	[Variable]
CL://	[Variable]
CL:///	[Variable]

While a form is being evaluated by an Exec, the variable CL:/ is bound to a list of the results of the last event in that Exec, CL:// to the values of the event before that, etc.

Fonts in the Exec

The Exec can use different fonts for displaying the prompt, user's input, intermediate printout, and the values returned by evaluation. The following variables control the Exec's font use:

PROMPTFONT	[Variable]
Font used for printing the event prompt.	
INPUTFONT	[Variable]
Font used for echoing your type-in.	
PRINTOUTFONT	[Variable]
Font used for any intermediate printing caused by execution of a command or evaluation of a form. Initially the same as DEFAULTFONT.	
VALUEFONT	[Variable]
Font used to print the values returned by evaluation of a form. Initially the same as DEFAULTFONT.	

Modifying an Exec

(CHANGESLICE <i>N HISTORY</i> —)	[Function]

Changes the maximum number of events saved on the history list *HISTORY* to *N*. If NIL, *HISTORY* defaults to the top level history LISPXHISTORY.

The effect of *increasing* the time-slice is gradual: the history list is simply allowed to grow to the corresponding length before any events are forgotten. *Decreasing* the time-slice will immediately remove a sufficient number of the older events to bring the history list down to the proper size. However, CHANGESLICE is undoable, so that these events are (temporarily) recoverable. Therefore, if you want to recover the storage associated with these events without waiting *N* more events until the CHANGESLICE event drops off the history list, you must perform a FORGET command.

Defining New Commands

You can define new Exec commands using the XCL:DEFCOMMAND macro.

(XCL:DEFCOMMAND <i>NAME ARGUMENT-LIST &REST BODY</i>)	[Macro]

XCL:DEFCOMMAND is like XCL:DEFMACRO, but defines new Exec commands. The *ARGUMENT-LIST* can have keywords, and use all of the features of macro argument lists. When *NAME* is subsequently typed to the Exec, the rest of the line is processed like the arguments to a macro, and the *BODY* is executed. XCL:DEFCOMMAND is a definer; the

MEDLEY EXECUTIVES

File Manager will remember typed-in definitions and allow them to be saved, edited with EDITDEF, etc.

There are three kinds of commands that can be defined, :EVAL, :QUIET, and :INPUT. Commands can also be marked as only for the debugger, in which case they are labelled as :DEBUGGER. The command type is noted by supplying a list for the *NAME* argument to XCL:DEFCOMMAND, where the first element of the list is the command name, and the other elements are keyword(s) for the command type and, optionally :DEBUGGER.

The documentation string in user defined Exec commands is automatically added to the documentation descriptions by the CL:DOCUMENTATION function under the COMMANDS type and can be shown using the ? Exec command.

:EVAL This is the default. The body of the command just gets executed, and its value is the value of the event. For example (in an XCL Exec),

```
Exec 3 (XCL)
3/47> (DEFCOMMAND (LS :EVAL)
      (&OPTIONAL (NAMESTRING
*DEFAULT-PATHNAME-DEFAULTS*)
      &REST DIRECTORY-KEYWORDS)
      (MAPC #'(LAMBDA (PATHNAME)
        (FORMAT T "~&~A" (NAMESTRING PATHNAME)))
      (APPLY #'DIRECTORY NAMESTRING
        (VALUES)))
      (VALUES))
LS
3/48>
```

would define the LS command to print out all file names that match the input NAMESTRING. The (VALUES) means that no value will be printed by the event, only the intermediate output from the FORMAT.

:QUIET These commands are evaluated, but neither your input nor the results of the command are stored on the history list. For example, the ?? and SHH commands are quiet.

:INPUT These commands work more like macros, in that the result of evaluating the command is treated as a new line of input. The FIX command is an input command. The result is treated as a line; a single expression in EVAL-format should be returned as a list of the expression to EVAL.

Undoing

Note: This discussion only applies to undoing under the Exec or Debugger, and within the UNDOABLY macro; text and structure editors handle undoing differently.

The UNDO facility allows recording of destructive changes such that they can be played back to restore a previous state. There are two kinds of UNDOing: one is done by the Exec, the other is available for use in your code. Both methods share information about what kind of operations can be undone and where the changes are recorded.

Undoing in the Exec

UNDO *EventSpec*

[Exec command]

The Exec's UNDO command is implemented by watching the evaluation of forms and requiring undoable operations in that evaluation to save enough information on the history list to reverse their side effects. The Exec simply executes operations, and any undoable changes that occur are automatically saved on the history list by the responsible functions. The UNDO command works on itself the same way: it recovers the saved information and performs the corresponding inverses. Thus, UNDO is effective on itself, so that you can UNDO an UNDO, and UNDO that, etc.

Only when you attempt to undo an operation does the Exec check to see whether any information has been saved. If none has been saved, and you have specifically named the event you want undone, the Exec types `nothing saved`. (When you just type UNDO, the Exec only tries to undo the last operation.)

UNDO watches evaluation using CL:EVALHOOK (thus, calling CL:EVALHOOK cannot be undone). Each form given to EVAL is examined against the list LISPXFNS to see if it has a corresponding undoable version. If an undoable version of a call is found, it is called with the same arguments instead of the original. Therefore, before evaluating all subforms of your input, the Exec substitutes the corresponding undoable call for any destructive operation. For example, if you type `(DEFUN FOO . . .)`, undoable versions of the forms that set the definition into the symbol function cell are evaluated. FOO's function definition itself is not made undoable.

Undoing in Programs

There are two ways to make a program undoable. The simplest method is to wrap the program's form in the UNDOABLY macro. The other is to call undoable versions of destructive operations directly.

(XCL:UNDOABLY &REST FORMS)

[Macro]

Executes the forms in *FORMS* using undoable versions of all destructive operations. This is done by "walking" (see WALKFORM) all of the *FORMS* and rewriting them to use the undoable versions of destructive operations (LISPXFNS makes the association).

(STOP-UNDOABLY &REST FORMS)

[Macro]

Normally executes as PROG; however, within an UNDOABLY form, explicitly causes *FORMS* not to be done undoably. Turns off rewriting of the *FORMS* to be undoable inside an UNDOABLY macro.

Undoable Versions of Common Functions

When efficiency is a serious concern, you may need more control over the saving of undo information than that provided by the UNDOABLY macro.

To make a function undoable, you can simply substitute the corresponding undoable function in your program. When the undoable function is called, it will save the undo information in the current event on the history list.

MEDLEY EXECUTIVES

Various operations, most notably SETF, have undoable versions. The following undoable macros are initially available:

UNDOABLY-POP	UNDOABLY-SET-SYMBOL
UNDOABLY-PUSH	UNDOABLY-MAKUNBOUND
UNDOABLY-PUSHNEW	UNDOABLY-FMAKUNBOUND
UNDOABLY-REMF	UNDOABLY-SETQ
UNDOABLY-ROTATEF	XCL:UNDOABLY-SETF
UNDOABLY-SHIFTF	UNDOABLY-PSETF
UNDOABLY-DECF	UNDOABLY-SETF-SYMBOL-FUNCTION
UNDOABLY-INCF	UNDOABLY-SETF-MACRO-FUNCTION

Note: Many destructive Common Lisp functions do not have undoable versions, e.g., CL:NREVERSE, CL:SORT, etc. You can see the current list of undoable functions on the association list LISPXFNS.

Modifying the UNDO Facility

You may want to extend the UNDO facility after creating a form whose side effects might be undoable, for instance a file renaming function.

You need to write an undoable version of the function. You can do this by explicitly saving previous state information, or by renaming calls in the function to their undoable equivalent. Undo information should be saved on the history list using IL:UNDOSAVE.

You must then hook the undoable version of the function into the undo facility. You do this by either using the IL:LISPXFNS association list, or in the case of a SETF modifier, on the IL:UNDOABLE-SETF-INVERSE property of the SETF function.

LISPXFNS

[Variable]

Contains an association list that maps from destructive operations to their undoable form. Initially this list contains:

```
((CL:POP . UNDOABLY-POP)
 (CL:PSETF . UNDOABLY-PSETF)
 (CL:PUSH . UNDOABLY-PUSH)
 (CL:PUSHNEW . UNDOABLY-PUSHNEW)
 ((CL:REMF) . UNDOABLY-REMF)
 (CL:ROTATEF . UNDOABLY-ROTATEF)
 (CL:SHIFTF . UNDOABLY-SHIFTF)
 (CL:DECF . UNDOABLY-DECF)
 (CL:INCF . UNDOABLY-INCF)
 (CL:SET . UNDOABLY-SET-SYMBOL)
 (CL:MAKUNBOUND . UNDOABLY-MAKUNBOUND)
 (CL:FMAKUNBOUND . UNDOABLY-FMAKUNBOUND)
 ... plus the original Interlisp undo associations)
```

(XCL:UNDOABLY-SETF PLACE VALUE ...)

[Macro]

Like CL:SETF but saves information so it may be undone. UNDOABLY-SETF uses undoable versions of the SETF function located on the UNDOABLE-SETF-INVERSE property of the function being SETFed. Initially these SETF names have such a property:

```
CL:SYMBOL-FUNCTION - UNDOABLY-SETF-SYMBOL-FUNCTION
CL:MACRO-FUNCTION - UNDOABLY-SETF-MACRO-FUNCTION
```

MEDLEY REFERENCE MANUAL

(**UNDOABLY-SETQ** &REST FORMS) [Function]

Typed-in SETQs (and SETFs on symbols) are made undoable by substituting a call to UNDOABLY-SETQ. UNDOABLY-SETQ operates like SETQ on lexical variables or those with dynamic bindings; it only saves information on the history list for changes to global, “top-level” values.

(**UNDOSAVE** UNDOFORM HISTENTRY) [Function]

Adds the undo information UNDOFORM to the SIDE property of the history event HISTENTRY. If there is no SIDE property, one is created. If the value of the SIDE property is NOSAVE, the information is not saved. HISTENTRY specifies an event. If HISTENTRY=NIL, the value of LISPXHIST is used. If both HISTENTRY and LISPXHIST are NIL, UNDOSAVE is a no-op.

The form of UNDOFORM is (FN . ARGS). Undoing is done by performing (APPLY (CAR UNDOFORM) (CDR UNDOFORM)).

\#UNDOSAVES [Variable]

The maximum number of UNDOFORMs to be saved for a single event. When the count of UNDOFORMs reaches this number, UNDOSAVE prints the message CONTINUE SAVING?, asking if you want to continue saving. If you answer NO or default, UNDOSAVE discards the previously saved information for this event, and makes NOSAVE be the value of the property SIDE, which disables any further saving for this event. If you answer YES, UNDOSAVE changes the count to -1, which is then never incremented, and continues saving. The purpose of this feature is to avoid tying up large quantities of storage for operations that will never need to be undone.

If \#UNDOSAVES is negative, then when the count reaches (ABS \#UNDOSAVES), UNDOSAVE simply stops saving without printing any messages or other interactions. \#UNDOSAVES = NIL is equivalent to \#UNDOSAVES = infinity. \#UNDOSAVES is initially NIL.

The configuration described here is very satisfactory. You pay a very small price for the ability to undo what you type in, since the interpreted evaluation is simply watched for destructive operations, or if you wish to protect yourself from malfunctioning in your own programs, you can explicitly call, or rewrite your program to explicitly call, undoable functions.

Undoing Out of Order

UNDOABLY-SETF operates undoably by saving (on the history list) the cell that is to be changed and its original contents. Undoing an UNDOABLY-SETF restores the saved contents.

This implementation can produce unexpected results when multiple modifications are made to the same piece of storage and then undone out of order. For example, if you type (SETF (CAR FOO) 1), followed by (SETF (CAR FOO) 2), then undo both events by undoing the most recent event first, then undoing the older event, FOO will be restored to its state before either event operated. However if you undo the first event, then the second event, (CAR FOO) will be 1, since this is what was in CAR of FOO before (UNDOABLY-SETF (CAR FOO) 2) was executed. Similarly, if you type

(NCONC FOO ' (1)), followed by (NCONC FOO ' (2)), undoing just (NCONC FOO ' (1)) will remove both 1 and 2 from FOO. The problem in both cases is that the two operations are not independent.

In general, operations are always independent if they affect different lists or different sublists of the same list. Undoing in reverse order of execution, or undoing independent operations, is always guaranteed to do the right thing. However, undoing dependent operations out of order may not always have the predicted effect.

Format and Use of the History List

LISPXHISTORY

[Variable]

The Exec currently uses one primary history list, LISPXHISTORY for the storing events.

The history list is in the form (*EVENTS* *EVENT#* *SIZE* *MOD*), where *EVENTS* is a list of events with the most recent event first, *EVENT#* is the event number for the most recent event on *EVENTS*, *SIZE* is the the maximum length *EVENTS* is allowed to grow. *MOD* is is the maximum event number to use, after which event numbers roll over. *LISPXHISTORY* is initialized to (NIL 0 100 1000).

The history list has a maximum length, called its time-slice. As new events occur, existing events are aged, and the oldest events are forgotten. The time-slice can be changed with the function *CHANGESLICE*. Larger time-slices enable longer memory spans, but tie up correspondingly greater amounts of storage. Since you seldom need really ancient history, a relatively small time-slice such as 30 events is usually adequate, although some users prefer to set the time-slice as large as 200 events.

Each individual event on *EVENTS* is a list of the form (*INPUT* *ID* *VALUE* . *PROPS*). For Exec events, *ID* is a list (*EVENT-NUMBER* *EXEC-ID*). The *EVENT-NUMBER* is the number of the event, while the *EXEC-ID* is a string that uniquely identifies the Exec. (The *EXEC-ID* is used to identify which events belong to the "same" Exec.) *VALUE* is the (first) value of the event. *PROPS* is a property list used to associate other information with the event (described below).

INPUT is the input sequence for the event. Normally, this is just the input that you type in. For an *APPLY*-format input this is a list consisting of two expressions; for an *EVAL*-format input, this is a list of just one expression; for an input entered as list of atoms, *INPUT* is simply that list. For example,

User Input

INPUT is:

LIST(1 2)	(LIST (1 2))
(LIST 1 1)	((LIST 1 1))
DIR "{DSK}<LISPFILES>"cr	(DIR "{DSK}<LISPFILES>")

If you type in an Exec command that executes other events (REDO, USE, etc.), several events might result. When there is more than one input, they are wrapped together into one invocation of the *DO-EVENTS* command.

The same convention is used for representing multiple inputs when a *USE* command involves sequential substitutions. For example, if you type *FBOUNDP* (FOO) and then *USE*

MEDLEY REFERENCE MANUAL

FIE FUM FOR FOO, the input sequence that will be constructed is DO-EVENTS (EVENT FBOUNDP (FIE)) (EVENT FBOUNDP (FUM)), which is the result of substituting FIE for FOO in (FBOUNDP (FOO)) concatenated with the result of substituting FUM for FOO in (FBOUNDP (FOO)).

PROPS is a property list of the form (PROPERTY VALUE PROPERTY VALUE ...), that can be used to associate arbitrary information with a particular event. Currently, the following properties are used by the Exec:

SIDE

A list of the side effects of the event. See UNDOSAVE.

LISPXPRT

Used to record calls to EXEC-FORMAT, and printed by the ?? command.

Making or Changing an Exec

(XCL:ADD-EXEC &KEY PROFILE REGION TTY ID) [Function]

Creates a new process and window with an Exec running in it. *PROFILE* is the type of the Exec to be created (see below under XCL:SET-EXEC-TYPE). *REGION* optionally gives the shape and location of the window to be used. If not provided you will be prompted. *TTY* is a flag, which, if true, causes the tty to be given to the new Exec process. *ID* is a string identifier to use for events generated in this exec. *ID* defaults to the number given to the Exec process created.

(XCL:EXEC &KEY WINDOW PROMPT COMMAND-TABLES ENVIRONMENT PROFILE TOP-LEVEL-P TITLE FUNCTION ID) [Function]

This is the main entry to the Exec. The arguments are:

WINDOW defaults to the current TTY display stream, or can be provided a window in which the Exec will run.

PROMPT is the prompt to print.

COMMAND-TABLES is a list of hash-tables for looking up commands (e.g., *EXEC-COMMAND-TABLE* or *DEBUGGER-COMMAND-TABLE*).

ENVIRONMENT is a lexical environment used to evaluate things in.

READTABLE is the default readtable to use (defaults to the “Common Lisp” readtable).

PROFILE is a way to set the Exec’s type (see above, “Multiple Execs and the Exec’s Type”).

TOP-LEVEL-P is a boolean, which should be true if this Exec is at the top level (it’s NIL for debugger windows, etc).

TITLE is an identifying title for the window title of the Exec.

FUNCTION is a function used to actually evaluate events, default is EVAL-INPUT.

MEDLEY EXECUTIVES

ID is a string identifier to use for events generated in this Exec. *ID* defaults to the number given to the Exec process.

XCL:*PER-EXEC-VARIABLES* [Variable]

A list of pairs of the form (*VAR INIT*). Each time an Exec is entered, the variables in **PER-EXEC-VARIABLES** are rebound to the value returned by evaluating *INIT*. The initial value of **PER-EXEC-VARIABLES** is:

```
( (*PACKAGE* *PACKAGE*)
  (* *)
  (** **)
  (***) ***)
  (+ +)
  (++ ++)
  (+++ +++)
  (- -)
  (/ /)
  (/// ///)
  (HELPFLAG T)
  (*EVALHOOK* NIL)
  (*APPLYHOOK* nil)
  (*ERROR-OUTPUT* *TERMINAL-IO*)
  (*READTABLE* *READTABLE*)
  (*package* *package*)
  (*eval-function* *eval-function*)
  (*exec-prompt* *exec-prompt*)
  (*debugger-prompt* *debugger-prompt*))
```

Most of these cause the values to be (re)bound to their current value in any inferior Exec, or to NIL, their value at the “top level”.

XCL:*EVAL-FUNCTION* [Variable]

Bound to the function used by the Exec to evaluate input. Typically in an Interlisp Exec this is *IL:EVAL*, and in a Common Lisp Exec, *CL:EVAL*.

XCL:*EXEC-PROMPT* [Variable]

Bound to the string printed by the Exec as a prompt for input. Typically in an Interlisp Exec this is “←”, and in a Common Lisp Exec, “>”.

XCL:*DEBUGGER-PROMPT* [Variable]

Bound to the string printed by the debugger Exec as a prompt for input. Typically in an Interlisp Exec this is “←:”, and in a Common Lisp Exec, “:”.

(XCL:EXEC-EVAL FORM &OPTIONAL ENVIRONMENT) [Function]

Evaluates *FORM* (using *EVAL*) in the lexical environment *ENVIRONMENT* the same as though it were typed in to *EXEC*, i.e., the event is recorded, and the evaluation is made undoable by substituting the UNDOABLE-functions for the corresponding destructive functions. *XCL:EXEC-EVAL* returns the value(s) of the form, but does not print it, and does not reset the variables ***, ****, *****, etc.

MEDLEY REFERENCE MANUAL

(**XCL:EXEC-FORMAT** *CONTROL-STRING &REST ARGUMENTS*) [Function]

In addition to saving inputs and values, the Exec saves many system messages on the history list. For example, FILE CREATED . . . , FN redefined, VAR reset, output of TIME, BREAKDOWN, ROOM, save their output on the history list, so that when ?? prints the event, the output is also printed. The function XCL:EXEC-FORMAT can be used in your code similarly. XCL:EXEC-FORMAT performs (APPLY #'CL:FORMAT *TERMINAL-IO* *CONTROL-STRING ARGUMENTS*) and also saves the format string and arguments on the history list associated with the current event.

(**XCL:SET-EXEC-TYPE** *NAME*) [Function]

Sets the type of the current Exec to that indicated by *NAME*. This can be used to set up the Exec to your liking. *NAME* may be an atom or string. Possible names are:

```
INTERLISP, IL *READTABLE* INTERLISP
              *PACKAGE* INTERLISP
              XCL:*DEBUGGER-PROMPT* "←: "
              XCL:*EXEC-PROMPT* "←"
              XCL:*EVAL-FUNCTION* IL:EVAL

XEROX-COMMON-LISP, XCL *READTABLE* XCL
                      *PACKAGE* XCL-USER
                      XCL:*DEBUGGER-PROMPT* ": "
                      XCL:*EXEC-PROMPT* "> "
                      XCL:*EVAL-FUNCTION* CL:EVAL

COMMON-LISP, CL *READTABLE* LISP
                *PACKAGE* USER
                XCL:*DEBUGGER-PROMPT* ": "
                XCL:*EXEC-PROMPT* "> "
                XCL:*EVAL-FUNCTION* CL:EVAL

OLD-INTERLISP-T *READTABLE* OLD-INTERLISP-T
                *PACKAGE* INTERLISP
                XCL:*DEBUGGER-PROMPT* "←: "
                XCL:*EXEC-PROMPT* ": "
                XCL:*EVAL-FUNCTION* IL:EVAL
```

(**XCL:SET-DEFAULT-EXEC-TYPE** *NAME*) [Function]

Like XCL:SET-EXEC-TYPE, but sets the type of Execs created by default, as from the background menu. Initially XCL. This can be used in your greet file to set default Execs to your liking.

Editing Exec Input

The Exec features an input editor which provides completion, spelling correction, help facility, and character-level editing. The implementation is borrowed from the Interlisp module TTYIN. This section describes the use of the TTYIN editor from the perspective of the Exec.

Editing Your Input

Some editing operations can be performed using any of several characters; characters that are interrupts will, of course, not be read, so several alternatives are given. The following characters may be used to edit your input:

CONTROL-A

MEDLEY EXECUTIVES

BACKSPACE Deletes a character. At the start of the second or subsequent lines of your input, deletes the last character of the previous line.

CONTROL-W Deletes a “word”. Generally this means back to the last space or parenthesis.

CONTROL-Q Deletes the current line, or if the current line is blank, deletes the previous line.

CONTROL-R Refreshes the current line. Two in a row refreshes the whole buffer (when doing multiline input).

ESCAPE Tries to complete the current word from the spelling list **USERWORDS**. In the case of ambiguity, completes as far as is uniquely determined, or beeps.

UNDO key Retrieves characters from the previous non-empty buffer when it is able to; e.g., when typed at the beginning of the line this command restores the previous line you typed; when typed in the middle of a line fills in the remaining text from the old line; when typed following **CONTROL-Q** or **CONTROL-W** restores what those commands erased.

CONTROL-X Goes to the end of your input (or end of expression if there is an excess right parenthesis) and returns if parentheses are balanced.

If you are already at the end of the input and the expression is balanced except for lacking one or more right parentheses, **CONTROL-X** adds the required right parentheses to balance and returns.

During most kinds of input, lines are broken, if possible, so that no word straddles the end of the line. The pseudo-carriage return ending the line is still read as a space, however; i.e., the program keeps track of whether a line ends in a carriage return or is merely broken at some convenient point. You will not get carriage returns in your strings unless you explicitly type them.

Using the Mouse

Editing with the mouse during **TTYIN** input is slightly different than with other modules. The mouse buttons are interpreted as follows during **TTYIN** input:

LEFT Moves the caret to where the cursor is pointing. As you hold down **LEFT**, the caret moves around with the cursor; after you let up, any type-in will be inserted at the new position.

MIDDLE

LEFT+RIGHT Like **LEFT**, but moves only to word boundaries.

RIGHT Deletes text from the caret to the cursor, either forward or backward. While you hold down **RIGHT**, the text to be deleted is inverted; when you let up, the text goes away. If you let up outside the scope of the text, nothing is deleted (this is how to cancel this operation).

If you hold down **MOVE**, **COPY**, **SHIFT** or **CTRL** while pressing the mouse buttons, you instead get secondary selection, move selection or delete selection. The selection is made by holding the appropriate key down while pressing the mouse buttons **LEFT** (to select a character) or **MIDDLE** (to select a word), and optionally extend the selection either left or right using **RIGHT**. While you are doing this, the caret does not move, but the selected text is highlighted in a manner indicating what is about to happen. When the selection is complete, release the mouse buttons and then lift up on **MOVE**/**COPY**/**CTRL**/**SHIFT** and the appropriate action will occur:

MEDLEY REFERENCE MANUAL

COPY

SHIFT The selected text is inserted as if it were typed. The text is highlighted with a broken underline during selection.

CTRL The selected text is deleted. The text is complemented during selection.

MOVE

CTRL+SHIFT Combines copy and delete. The selected text is moved to the caret.

You can cancel a selection in progress by pressing **LEFT** or **MIDDLE** as if to select, and moving outside the range of the text.

The most recent text deleted by mouse command can be inserted at the caret by typing the **UNDO** key. This is the same key that retrieves the previous buffer when issued at the end of a line.

Editing Commands

A number of characters have special effects while typing to the Exec. Some of them merely move the caret inside the input stream. While caret positioning can often be done more conveniently with the mouse, some of the commands, such as the case changing commands, can be useful for modifying the input.

In the descriptions below, current word means the word the cursor is under, or if under a space, the previous word. Currently, parentheses are treated as spaces, which is usually what you want, but can occasionally cause confusion in the word deletion commands.

Most commands can be preceded by a numeric argument. A numeric argument can be a number or an escape. You enter the numeric argument by holding down the meta key and entering a number. You only need to hold down the meta key for the first digit of the argument. Entering escape as a numeric argument means infinity.

Some commands also accept negative arguments, but some only look at the magnitude of the argument. Most of these commands are confined to work within one line of text unless otherwise noted.

Cursor Movement Commands

Meta-BACKSPACE Backs up one (or n) characters.

Meta-SPACE Moves forward one (or n) characters.

Meta-^ Moves up one (or n) lines.

Meta-LINEFEED Moves down one (or n) lines.

Meta-(Moves back one (or n) words.

Meta-) Moves ahead one (or n) words.

Meta-tab Moves to end of line; with an argument moves to nth end of line; **Meta-Control-tab** goes to end of buffer.

Meta-Control-L Moves to start of line (or nth previous, or start of buffer).

Meta-{ Goes to start of buffer.

Meta-} Goes to end of buffer.

MEDLEY EXECUTIVES

Meta-[Moves to beginning of the current list, where cursor is currently under an element of that list or its closing paren. (See also the auto-parenthesis-matching feature below under “Assorted Flags”).

Meta-] Moves to end of current list.

Meta-Sx Skips ahead to next (or nth) occurrence of character x, or rings the bell.

Meta-Bx Backward search.

Buffer Modification Commands

Meta-Zx Zaps characters from cursor to next (or nth) occurrence of x. There is no unzap command.

Meta-A

Meta-R Repeats the last S, B, or Z command, regardless of any intervening input.

Meta-K Kills the character under the cursor, or n chars starting at the cursor.

Meta-CR When the buffer is empty is the same as undo i.e. restores buffer's previous contents. Otherwise is just like a <cr> (except that it also terminates an insert). Thus, **Meta-CR Meta-CR** will repeat the previous input (as will undo<cr> without the meta key).

Meta-O Does “Open line”, inserting a crlf after the cursor, i.e., it breaks the line but leaves the cursor where it is.

Meta-T Transposes the characters before and after the cursor. When typed at the end of a line, transposes the previous two characters. Refuses to handle odd cases, such as tabs.

Meta-G Grabs the contents of the previous line from the cursor position onward. **Meta-n Meta-G** grabs the nth previous line.

Meta-L Puts the current word, or n words on line, in lower case. **Meta-<escape> Meta-L** puts the rest of the line in lower case; or if given at the end of line puts the entire line in lower case.

Meta-U Analogous to **Meta-L**, for putting word, line, or portion of line in upper case.

Meta-C Capitalizes. If you give it an argument, only the first word is capitalized; the rest are just lowercased.

Meta-Control-Q Deletes the current line. **Meta-<escape> Meta-Control-Q** deletes from the current cursor position to the end of the buffer. No other arguments are handled.

Meta-Control-W Deletes the current word, or the previous word if sitting on a space.

Miscellaneous Commands

Meta-P Prettyprints buffer. Clears the buffer and reprints it using prettyprint. If there are not enough right parentheses, it will supply more; if there are too many, any excess remains unprettyprinted at the end of the buffer. May refuse to do anything if there is an unclosed string or other error trying to read the buffer.

MEDLEY REFERENCE MANUAL

Meta-N Refreshes line. Same as **Control-R**. **Meta-`<escape>` Meta-N** refreshes the whole buffer; **Meta-n Meta-N** refreshes n lines. Cursor movement in TTYIN depends on TTYIN being the only source of output to the window; in some circumstances, you may need to refresh the line for best results.

Meta-Control-Y Gets an Interlisp Exec. **Meta-`<escape>` Meta-Control-Y** Gets an Interlisp Exec, but first unread the contents of the buffer from the cursor onward. Thus if you typed at TTYIN something destined for Interlisp, you can do **Meta-Control-L Meta-`<escape>` Meta-Control-Y** and give it to Lisp.

Meta-`_` Adds the current word to the spelling list USERWORDS. With zero argument, removes word. See TTYINCOMPLETEFLG.

Useful Macros

If the event is considered short enough, the Exec command `FIX` will load the buffer with the event's input, rather than calling the structure editor. If you really wanted the Lisp editor for your fix, you can say `FIX EVENT - |TTY:|`.

?= Handler

Typing the characters `?=<cr>` displays the arguments to the function currently in progress. Since TTYIN wants you to be able to continue editing the buffer after a `?=`, it prints the arguments below your type-in and then puts the cursor back where it was when `?=` was typed.

Assorted Flags

These flags control aspects of TTYIN's behavior. Some have already been mentioned. All are initially set to T.

?ACTIVATEFLG [Variable]

If true, enables the feature whereby `? lists alternative completions from the current spelling list.`

SHOWPARENFLG [Variable]

If true, then whenever you are typing Lisp input and type a right parenthesis, TTYIN will briefly move the cursor to the matching parenthesis, assuming it is still on the screen. The cursor stays there for about 1 second, or until you type another character (i.e., if you type fast you will never notice it).

USERWORDS [Variable]

USERWORDS contains words you mentioned recently: functions you have defined or edited, variables you have set or evaluated at the executive level, etc. This happens to be a very convenient list for context-free escape completion; if you have recently edited a function, chances are good you may want to edit it again (typing `"ED (xx$)"`) or type a call to it. If there is no completion for the current word from USERWORDS, or there is more than one possible completion, TTYIN beeps. If typed when not inside a word, Escape completes to the value of LASTWORD, i.e., the last thing you typed that the Exec noticed,

MEDLEY EXECUTIVES

except that Escape at the beginning of the line is left alone (it is an Old Interlisp Exec command).

If you really wanted to enter an escape, you can, of course, just quote it with a CONTROL-V, like you can other control characters.

You may explicitly add words to USERWORDS yourself that would not get there otherwise. To make this convenient online the edit command [←] means “add the current atom to USERWORDS” (you might think of the command as pointing out this atom). For example, you might be entering a function definition and want to point to one or more of its arguments or prog variables. Giving an argument of zero to this command will instead remove the indicated atom from USERWORDS.

Note that this feature loses some of its value if the spelling list is too long, if there are too many alternative completions for you to get by with typing a few characters followed by escape. Lisp’s maintenance of the spelling list USERWORDS keeps the temporary section (which is where everything goes initially unless you say otherwise) limited to \#USERWORDS atoms, initially 100. Words fall off the end if they haven’t been used (they are used if FIXSPELL corrects to one, or you use <escape> to complete one).

Old Interlisp T compatibility

The Old Interlisp exec contains a few extra Exec commands not listed above. They are explained here.

In addition to the normal Event addresses you can also specify the following Event addresses:

= Specifies that the next object is to be searched for in the values of events, instead of the inputs

SUCHTHAT *PRED* Specifies an event for which the function *PRED* returns true. *PRED* should be a function of two arguments, the input portion of the event, and the event itself.

PAT Any other event address command specifies an event whose input contains an expression that matches *PAT*. When multiple Execs are active, all events are searched, no matter which Exec they belong to. The pattern can be a simple symbol, or a more complex search pattern.

Significant Changes in MEDLEY Release [REDACTED]ase

There are two major differences between the Medley release and older versions of the system:

- SETQ does not interact with the File Manager. In older releases (Koto, etc.), when you typed in (SETQ FOO *some-new-value*) the executive responded with (FOO reset) and the file manager was told that FOO’s value had changed. Files containing FOO were marked for cleanup, if none existed you were prompted for one when you typed (FILES?).

This is still the case in the Old Interlisp executive but not in any of the others. If you are setting a variable that is significant to a program and you want to save it on a file, you should use the Common Lisp macro CL:DEFPARAMETER instead of SETQ. This will give the symbol a definition of type VARIABLES (instead of VARS), and it will be noticed by the File Manager. Subsequent

MEDLEY REFERENCE MANUAL

changes to the variable must be done by another call to `CL:DEFPARAMETER` or by editing it using `ED` (not `DV`).

- The following functions and variables are only available in the Old Interlisp Exec: `LISPX`, `USEREXEC`, `LISPXEVAL`, `READBUF`, `(READLINE)`, `(LISPXREAD)`, `(LISPXREADP)`, `(LISPXUNREAD)`, `(PROMPTCHAR)`, `(HISTORYSAVE)`, `(LISPXSTOREVALUE)`, `(LISPXFIND)`, `(HISTORYFIND)`, `(HISROTYMATCH)`, `(ENTRY)`, `(UNDOSAVE)`, `#UNDOSAVES`, `(NEW/FN)`, `(LISPX/)`, `(UNDOLISPX)`, `(UNDOLISPX1)`, and `(PRINTHISTORY)`.

The function `USEREXEC` invokes an old-style executive, but uses the package and readtable of its caller. Callers of `LISPXEVAL` should use `EXEC-EVAL` instead.

MEDLEY EXECUTIVES

[This page intentionally left blank]

MEDLEY REFERENCE MANUAL

14. ERRORS AND DEBUGGING

Occasionally, while a program is running, an error occurs which stops the computation. Errors can be caused in different ways. A coding mistake may have caused the wrong arguments to be passed to a function, or caused the function to attempt something illegal. For example, `PLUS` will cause an error if its arguments are not numbers. It is also possible to interrupt a computation by typing one of the “interrupt characters,” such as Control-D or Control-E (Medley interrupt characters are listed in Chapter 30). Finally, you can specify that certain functions automatically cause an error whenever they are entered (see Chapter 15). This facilitates debugging by allowing you to examine the context within the computation.

When an error occurs, the system can either reset and unwind the stack, or go into a “break”, and attempt to debug the program. You can modify the mechanism that decides whether to unwind the stack or break, and is described in the Controlling When to Break section in this chapter. Within a break, Medley offers an extensive set of “break commands”.

This chapter explains what happens when errors occur. It also tells you how to handle program errors using breaks and break commands. The debugging capabilities of the break window facility are described, as well as the variables that control its operation. Finally, advanced facilities for modifying and extending the error mechanism are presented.

Breaks

One of the most useful debugging facilities in Medley is the ability to put the system into a “break”, stopping a computation at any point, allowing you to interrogate the state of the world and affect the course of the computation. When a break occurs, a “break window” (see the Break Windows section below) is brought up near the TTY window of the broken process. The break window looks like a top-level executive window, except that the prompt character is “:” instead of “←” as in the top-level executive. A break saves the environment where the break occurred, so that you may evaluate variables and expressions in the broken environment. In addition, the break program recognizes a number of useful “break commands”, providing an easy way to interrogate the state of the broken computation.

Breaks may be entered in several ways. Some interrupt characters (Chapter 30) automatically cause a break whenever you type them. Function errors may also cause a break, depending on the depth of the computation (see Controlling When to Break below). Finally, Medley provides facilities which make it easy to “break” suspect functions so that they always cause a break whenever they are entered.

Within a break you have access to all of the power of Medley; you can do anything you can do at the top-level executive. For example, you can evaluate an expression, call the editor, change the function, and evaluate the expression again, all without leaving the break. You can also type in commands like `REDO`, and `UNDO` (Chapter 13), to redo or undo previously executed events, including break commands.

Similarly, you can prettyprint functions, define new functions or redefine old ones, load a file, compile functions, time a computation, etc. In addition, you can examine the stack (see Chapter 11), and even force a return back to some higher function via the functions `RETFROM` or `RETEVAL`.

MEDLEY REFERENCE MANUAL

Once a break occurs, *you* are in complete control of the flow of the computation, and the computation will not proceed without specific instruction from you. If you type in an expression whose evaluation causes an error, the break is maintained. Similarly if you abort a computation initiated from within the break (by typing Control-E), the break is maintained. Only if you give one of the commands that exits from the break, or evaluates a form which does a RETFROM or RETEVAL out of BREAK1, will the computation continue. Also, BREAK1 does not “turn off” Control-D, so a Control-D will force an immediate return to the top level.

Break Windows

When a break occurs, a break window is brought up near the TTY window of the broken process and the terminal stream switched to it. The title of the break window is changed to the name of the broken function and the reason for the break. If a break occurs under a previous break, a new break window is created.

If a break is caused by a storage full error, the display break package will not try to open a new break window, since this would cause an infinite loop.

While in a break window, clicking the middle button brings up a menu of break commands: EVAL, EDIT, revert, ↑, OK, BT, BT!, and ?=. Clicking on these commands is equivalent to typing the corresponding break command, except BT and BT! which behave differently from the typed-in commands (see Break Commands below).

The BT and BT! menu commands bring up a backtrace menu beside the break window showing the frames on the stack. BT shows frames for which REALFRAMEP is T; BT! shows all frames. When one of the frames is selected from the backtrace menu, it is grayed and the function name and the variables bound in that frame (including local variables and PROG variables) are printed in the “backtrace frame window.” If the left button is used for the selection, only named variables are printed. If the middle button is used, all variables are printed (variables without names appear as *var* N). The “backtrace frame” window is an inspect window (see Chapter 26). In this window, the left button is used to select the name of the function, the names of the variables or the values of the variables. For example, below is a picture of a break window with a backtrace menu created by BT. The OPENSTREAM stack frame has been selected, so its variables are shown in an inspect window on top of the break window:

OPENSTREAM Frame	
OPENSTREAM	
FILE	{DSK}FOO
ACCESS	INPUT
RECOG	OLD
PARAMETERS	NIL
OBSOLETE	NIL
*var*6	OLD
*var*7	NIL
*var*8	NIL
ERRORSET {DSK}FOO - FILE NOT FOUND break: 1	
BREAK1	
EVALA	FILE NOT FOUND
OPENSTREAM	{DSK}FOO
EVAL	
LISPX	
ERRORSET	(OPENSTREAM broken)
EVALQT	46;
ERRORSET	A
T	

After selecting an item, the middle button brings up a menu of commands that apply to the selected item. If the function name is selected, you are given a choice of editing the function or seeing the

ERRORS AND DEBUGGING

compiled code with `INSPECTCODE` (Chapter 26). If you edit the function in this way, the editor is called in the broken process, so variables evaluated in the editor are in the broken process.

If a variable name is selected, the command `SET` is offered. Selecting `SET` will `READ` a value and set the selected to the value read.

Note: The inspector will only allow the setting of named variables. Even with this restriction it is still possible to crash the system by setting variables inside system frames. Exercise caution in setting variables in other than your own code.

If a value is selected, the inspector is called on the selected value.

The internal break variable `LASTPOS` (see the section below) is set to the selected backtrace menu frame so that the normal break commands `EDIT`, `revert`, and `?=` work on the currently selected frame. The commands `EVAL`, `revert`, `↑`, `OK`, and `?=` in the break menu cause the corresponding commands to be “typed in.” This means that these break commands will not have the intended effect if characters have already been typed in. The typed-in break commands `BT`, `BTV`, etc. use the value of `LASTPOS` to determine where to start listing the stack, so selecting a stack frame name in the backtrace menu affects these commands.

Break Commands

The basic function of the break package is `BREAK1`. `BREAK1` is just another Interlisp function, not a special system feature like the interpreter or the garbage collector. It has arguments, and returns a value, the same as any other function. For more information on the function `BREAK1`, see *Creating Breaks with `BREAK1`* below.

The value returned by `BREAK1` is called “the value of the break.” You can specify this value explicitly by using the `RETURN` break command (see below). But in most cases, the value of a break is given implicitly, via a `GO` or `OK` command, and is the result of evaluating “the break expression.” The break expression, stored in the variable `BRKEXP`, is an expression equivalent to the computation that would have taken place had no break occurred. For example, if you break on the function `FOO`, the break expression is the body of the definition of `FOO`. When you type `OK` or `GO`, the body of `FOO` is evaluated, and its value returned as the value of the break, i.e., to whatever function called `FOO`. `BRKEXP` is set up by the function that created the call to `BREAK1`. For functions broken with `BREAK` or `TRACE`, `BRKEXP` is equivalent to the body of the definition of the broken function (see Chapter 15). For functions broken with `BREAKIN`, using `BEFORE` or `AFTER`, `BRKEXP` is `NIL`. For `BREAKIN AROUND`, `BRKEXP` is the indicated expression (see Chapter 15).

`BREAK1` recognizes a large set of break commands. These are typed in *without* parentheses. In order to facilitate debugging of programs that perform input operations, the carriage return that is typed to complete the `GO`, `OK`, `EVAL`, etc. commands is discarded by `BREAK1`, so that it will not be part of the input stream after the break.

GO

[Break Command]

Evaluates `BRKEXP`, prints its value, and returns it as the value of the break. Releases the break and allows the computation to proceed.

MEDLEY REFERENCE MANUAL

OK [Break Command]

Same as GO except that the value of BRKEXP is not printed.

EVAL [Break Command]

Same as OK except that the break is maintained after the evaluation. The value of EVAL is bound to the local variable !VALUE, which you can interrogate. Typing GO or OK following EVAL will not cause BRKEXP to be reevaluated, but simply returns the value of !VALUE as the value of the break. Typing another EVAL will cause reevaluation. EVAL is useful when you are not sure whether the break will produce the correct value and want to examine it before continuing with the computation.

RETURN FORM [Break Command]

FORM is evaluated, and returned as the value of the break. For example, one could use the EVAL command and follow this with RETURN (REVERSE !VALUE).

↑ [Break Command]

Calls ERROR! and aborts the break, making it “go away” without returning a value. This is a useful way to unwind to a higher level break. All other errors, including those encountered while executing the GO, OK, EVAL, and RETURN commands, maintain the break.

The following four commands refer to “the broken function”, whose name is stored in the BREAK1 argument BRKFN.

!GO [Break Command]

The broken function is unbroken, the break expression is evaluated, the function is rebroken, and then the break is exited with the value printed.

!OK [Break Command]

The broken function is unbroken, the break expression is evaluated, the function is rebroken, and then the break is exited without the value printed.

UB [Break Command]

Unbreaks the broken function.

@ [Break Command]

Resets the variable LASTPOS, which establishes a context for the commands ?=, ARGS, BT, BTV, BTV*, EDIT, and IN? described below. LASTPOS is the position of a function call on the stack. It is initialized to the function just before the call to BREAK1, i.e., (STKNTH -1 'BREAK1).

When control passes from BREAK1, e.g. as a result of an EVAL, OK, GO, REVERT, ↑ command, or via a RETFROM or RETEVAL you type in, (RELSTK LASTPOS) is executed to release this stack pointer.

ERRORS AND DEBUGGING

@ treats the rest of the teletype line as its argument(s). It first resets LASTPOS to (STKNTH -1 'BREAK1) and then for each atom on the line, @ searches down the stack for a call to that atom. The following atoms are treated specially:

- @ Do not reset LASTPOS to (STKNTH -1 'BREAK1) but leave it as it was, and continue searching from that point.
- a number *N* If negative, move LASTPOS down the stack *N* frames. If positive, move LASTPOS up the stack *N* frames.
- / The next atom on the line (which should be a number) specifies that the *previous* atom should be searched for that many times. For example, "@ FOO / 3" is equivalent to "@ FOO FOO FOO".
- = Resets LASTPOS to the *value* of the next expression, e.g., if the value of FOO is a stack pointer, "@ = FOO FIE" will search for FIE in the environment specified by (the value of) FOO.

For example, if the push-down stack looks like:

```
[9]  BREAK1
[8]  FOO
[7]  COND
[6]  FIE
[5]  COND
[4]  FIE
[3]  COND
[2]  FIE
[1]  FUM
```

then "@ FIE COND" will set LASTPOS to the position corresponding to [5]; "@ @ COND" will then set LASTPOS to [3]; and "@ FIE / 3 - 1" to [1].

If @ cannot successfully complete a search for function *FN*, it searches the stack again from that point looking for a call to a function whose name is a possible misspelling of *FN* (see spelling correction in Chapter 20). If the search is still unsuccessful, @ types (*FN* NOT FOUND), and then aborts.

When @ finishes, it types the name of the function at LASTPOS, i.e., (STKNAME LASTPOS).

@ can be used on BRKCOMS (see Creating Breaks with BREAK1 below). In this case, the *next* command on BRKCOMS is treated the same as the rest of the teletype line.

?=

[Break Command]

This is a multi-purpose command. Its most common use is to interrogate the value(s) of the arguments of the broken function. For example, if FOO has three arguments (X Y Z), then typing ?= to a break on FOO will produce:

```
: ?=
X = value of X
```

MEDLEY REFERENCE MANUAL

```
Y = value of Y
Z = value of Z
:
```

?= operates on the rest of the teletype line as its arguments. If the line is empty, as in the above case, it operates on all of the arguments of the broken function. If the you type ?= X (CAR Y), you will see the value of X, and the value of (CAR Y). The difference between using ?= and typing X and (CAR Y) directly to BREAK1 is that ?= evaluates its inputs as of the stack frame LASTPOS, i.e., it uses STKEVAL. This provides a way of examining variables or performing computations *as of a particular point on the stack*. For example, @ FOO / 2 followed by ?= X will allow you to examine the value of X in the previous call to FOO, etc.

?= also recognizes numbers as referring to the correspondingly numbered argument, i.e., it uses STKARG in this case. Thus

```
:@ FIE
FIE
:?= 2
```

will print the name and value of the second argument of FIE.

?= can also be used on BRKCOMS (see Creating Breaks with BREAK1 below), in which case the next command on BRKCOMS is treated as the rest of the teletype line. For example, if BRKCOMS is (EVAL ?= (X Y) GO), BRKEXP is evaluated, the values of X and Y printed, and then the function exited with its value being printed.

?= prints variable values using the function SHOWPRINT (see Chapter 25), so that if SYSPRETTYFLG = T, the value is prettyprinted.

?= is a universal mnemonic for displaying argument names and their corresponding values. In addition to being a break command, ?= is an edit macro that prints the argument names and values for the current expression (see Chapter 16), and a read macro (actually ? is the read macro character) which does the same for the current level list being read.

PB

[Break Command]

Prints the bindings of a given variable. Similar to ?=, except ascends the stack starting from LASTPOS, and, for each frame in which the given variable is bound, prints the frame name and value of the variable (with PRINTLEVEL reset to (2 . 3)), e.g.

```
:PB FOO
@ FN1: 3
@ FN2: 10
@ TOP: NOBIND
```

PB is also a programmer's assistant command (see Chapter 13) that can be used when not in a break. PB is implemented via the function PRINTBINDINGS.

BT

[Break Command]

Prints a backtrace of function names starting at LASTPOS. The value of LASTPOS is changed by selecting an item from the backtrace menu (see the Break Window Variables

ERRORS AND DEBUGGING

section below) or by the @ command. The several nested calls in system packages such as break, edit, and the top level executive appear as the single entries ****BREAK****, ****EDITOR****, and ****TOP**** respectively.

BT [Break Command]

Prints a backtrace of function names *with* variables beginning at LASTPOS.

The value of each variable is printed with the function SHOWPRINT (see Chapter 25), so that if SYSPRETTYFLG = T, the value is prettyprinted.

BT+ [Break Command]

Same as BT except also prints local variables and arguments to SUBRs.

BT* [Break Command]

Same as BT except prints arguments to local variables.

BT! [Break Command]

Same as BT except prints *everything* on the stack.

BT, BT, BT+, BT*, and BT! all take optional functional arguments. Use these arguments to choose functions to be *skipped* on the backtrace. As the backtrace scans down the stack, the name of each stack frame is passed to each of the arguments of the backtrace command. If any of these functions returns a non-NIL value, then that frame is skipped, and not shown in the backtrace. For example, BT EXPRP will skip all functions defined by expr definitions, BT (LAMBDA (X) (NOT (MEMB X FOOFNS))) will skip all but those functions on FOOFNS. If used on BRKCOMS (see Creating Breaks with BREAK1 below) the functional argument is no longer optional, i.e., the next element on BRKCOMS must either be a list of functional arguments, or NIL if no functional argument is to be applied.

For BT, BT, BT+, BT*, and BT!, if Control-P is used to change a printlevel during the backtrace, the printlevel is restored after the backtrace is completed.

The value of BREAKDELIMITER, initially the carriage return character, is printed to delimit the output of ?= and backtrace commands. You can reset it (e.g. to a comma) for more linear output.

ARGS [Break Command]

Prints the names of the variables bound at LASTPOS, i.e., (VARIABLES LASTPOS) (see Chapter 11). For most cases, these are the arguments to the function entered at that position, i.e., (ARGLIST (STKNAME LASTPOS)).

REVERT [Break Command]

Goes back to position LASTPOS on stack and reenters the function called at that point with the arguments found on the stack. If the function is not already broken, REVERT first breaks it, and then unbreaks it after it is reentered.

REVERT can be given the position using the conventions described for @, e.g., REVERT FOO -1 is equivalent to @ FOO -1 followed by REVERT.

MEDLEY REFERENCE MANUAL

REVERT is useful for restarting a computation in the situation where a bug is discovered at some point *below* where the problem actually occurred. REVERT essentially says “go back there and start over in a break.” REVERT will work correctly if the names or arguments to the function, or even its function type, have been changed.

ORIGINAL

[Break Command]

For use in conjunction with BREAKMACROS (see Creating Breaks with BREAK1 below). Form is (ORIGINAL . COMS). COMS are executed without regard for BREAKMACROS. Useful for redefining a break command in terms of itself.

EDIT

[Break Command]

Designed for use in conjunction with breaks caused by errors. Facilitates editing the expression causing the break:

```
NON-NUMERIC ARG
NIL
(IPLUS BROKEN)
:EDIT
IN FOO...
(IPLUS X Z)
EDIT
*(3 Y)
*OK
FOO
:
```

and you can continue by typing OK, EVAL, etc.

This command is very simple conceptually, but its implementation is complicated by all of the exceptional cases involving interactions with compiled functions, breaks on user functions, error breaks, breaks within breaks, et al. Therefore, we shall give the following simplified explanation which will account for 90% of the situations arising in actual usage. For those others, EDIT will print an appropriate failure message and return to the break.

EDIT begins by searching up the stack beginning at LASTPOS (set by @ command, initially position of the break) looking for a form, i.e., an internal call to EVAL. Then EDIT continues from that point looking for a call to an interpreted function, or to EVAL. It then calls the editor on either the EXPR or the argument to EVAL in such a way as to look for an expression EQ to the form that it first found. It then prints the form, and permits interactive editing to begin. You can then type successive 0's to the editor to see the chain of superforms for this computation.

If you exit from the edit with an OK, the break expression is reset, if possible, so that you can continue with the computation by simply typing OK. (Evaluating the new BRKEXP will involve reevaluating the form that causes the break, so that if (PUTD (QUOTE (FOO)) BIG-COMPUTATION) were handled by EDIT, BIG-COMPUTATION would be reevaluated.) However, in some situations, the break expression cannot be reset. For example, if a compiled function FOO incorrectly called PUTD and caused the error Arg not atom followed by a break on PUTD, EDIT might be able to find the form headed by FOO, and also find *that* form in some higher interpreted function. But after you corrected the problem in the FOO-form, if any, you would still not have informed EDIT what to do

ERRORS AND DEBUGGING

about the immediate problem, i.e., the incorrect call to PUTD. However, if FOO were *interpreted*, EDIT would find the PUTD form itself, so that when you corrected that form, EDIT could use the new corrected form to reset the break expression.

IN?

[Break Command]

Similar to EDIT, but just prints parent form, and superform, but does not call the editor, e.g.,

```
ATTEMPT TO RPLAC NIL
T
(RPLACD BROKEN)
:IN?
FOO: (RPLACD X Z)
```

Although EDIT and IN? were designed for error breaks, they can also be useful for user breaks. For example, if upon reaching a break on his function FOO, you determine that there is a problem in the *call* to FOO, you can edit the calling form and reset the break expression with one operation by using EDIT.

Controlling When to Break

When an error occurs, the system has to decide whether to reset and unwind the stack, or go into a break. In the middle of a complex computation, it is usually helpful to go into a break, so that you may examine the state of the computation. However, if the computation has only proceeded a little when the error occurs, such as when you mistype a function name, you would normally just terminate a break, and it would be more convenient for the system to simply cause an error and unwind the stack in this situation. The decision over whether or not to induce a break depends on the depth of computation, and the amount of time invested in the computation. The actual algorithm is described in detail below; suffice it to say that the parameters affecting this decision have been adjusted empirically so that trivial type-in errors do not cause breaks, but deep errors do.

(BREAKCHECK ERRORPOS ERXN)

[Function]

BREAKCHECK is called by the error routine to decide whether or not to induce a break when a error occurs. *ERRORPOS* is the stack position at which the error occurred; *ERXN* is the error number. Returns T if a break should occur; NIL otherwise.

BREAKCHECK returns T (and a break occurs) if the “computation depth” is greater than or equal to *HELPDEPTH*. *HELPDEPTH* is initially set to 7, arrived at empirically by taking into account the overhead due to *LISPX* or *BREAK*.

If the depth of the computation is less than *HELPDEPTH*, BREAKCHECK next calculates the length of time spent in the computation. If this time is greater than *HELPTIME* milliseconds, initially set to 1000, then BREAKCHECK returns T (and a break occurs), otherwise NIL.

BREAKCHECK determines the “computation depth” by searching back up the stack looking for an *ERRORSET* frame (*ERRORSET*s indicate how far back unwinding is to take place when an error occurs, see the Catching Errors section below). At the same time, it counts the number of internal calls to *EVAL*. As soon as the number of calls to *EVAL* exceeds *HELPDEPTH*, BREAKCHECK immediately stops searching for an *ERRORSET* and returns T.

MEDLEY REFERENCE MANUAL

Otherwise, `BREAKCHECK` continues searching until either an `ERRORSET` is found or the top of the stack is reached. (If the second argument to `ERRORSET` is `INTERNAL`, the `ERRORSET` is ignored by `BREAKCHECK` during this search.) `BREAKCHECK` then counts the number of function calls between the error and the last `ERRORSET`, or the top of the stack. The number of function calls plus the number of calls to `EVAL` (already counted) is used as the “computation depth”.

`BREAKCHECK` determines the computation time by subtracting the value of the variable `HELPCLOCK` from the value of `(CLOCK 2)`, the number of milliseconds of compute time (see Chapter 12). `HELPCLOCK` is rebound to the current value of `(CLOCK 2)` for each computation typed in to `LISPX` or to a break. The time criterion for breaking can be suppressed by setting `HELPTIME` to `NIL` (or a very big number), or by setting `HELPCLOCK` to `NIL`. Setting `HELPCLOCK` to `NIL` will not have any effect beyond the current computation, because `HELPCLOCK` is rebound for each computation typed in to `LISPX` and `BREAK`.

You can suppress all error breaks by setting the top level binding of the variable `HELPFLAG` to `NIL` using `SETTOPVAL` (`HELPFLAG` is bound as a local variable in `LISPX`, and reset to the global value of `HELPFLAG` on every `LISPX` line, so just `SETQing` it will not work.) If `HELPFLAG` = `T` (the initial value), the decision whether to cause an error or break is decided based on the computation time and the computation depth, as described above. Finally, if `HELPFLAG` = `BREAK!`, a break will always occur following an error.

Break Window Variables

The appearance and use of break windows is controlled by the following variables:

(WBREAK ONFLG) [Function]

If `ONFLG` is non-`NIL`, break windows and trace windows are enabled. If `ONFLG` is `NIL`, break windows are disabled (break windows do not appear, but the executive prompt is changed to “:” to indicate that the system is in a break). `WBREAK` returns `T` if break windows are currently enabled; `NIL` otherwise.

MaxBkMenuWidth [Variable]
MaxBkMenuHeight [Variable]

The variables `MaxBkMenuWidth` (default 125) and `MaxBkMenuHeight` (default 300) control the maximum size of the backtrace menu. If this menu is too small to contain all of the frames in the backtrace, it is made scrollable in both vertical and horizontal directions.

AUTOBACKTRACEFLG [Variable]

This variable controls when and what kind of backtrace menu is automatically brought up. The value of `AUTOBACKTRACEFLG` can be one of the following:

`NIL` The backtrace menu is not automatically brought up (the default).

`T` On error breaks the BT menu is brought up.

`BT!` On error breaks the BT! menu is brought up.

ERRORS AND DEBUGGING

ALWAYS The BT menu is brought up on both error breaks and user breaks (calls to functions broken by **BREAK**).

ALWAYS! On both error breaks and user breaks the **BT!** menu is brought up.

BACKTRACEFONT [Variable]

The backtrace menu is printed in the font **BACKTRACEFONT**.

CLOSEBREAKWINDOWFLG [Variable]

The system normally closes break windows after the break is exited. If **CLOSEBREAKWINDOWFLG** is **NIL**, break windows will not be closed on exit. In this case, you must close all break windows.

BREAKREGIONSPEC [Variable]

Break windows are positioned near the TTY window of the broken process, as determined by the variable **BREAKREGIONSPEC**. The value of this variable is a region (see Chapter 27) whose **LEFT** and **BOTTOM** fields are an offset from the **LEFT** and **BOTTOM** of the TTY window. The **WIDTH** and **HEIGHT** fields of **BREAKREGIONSPEC** determine the size of the break window.

TRACEWINDOW [Variable]

The trace window, **TRACEWINDOW**, is used for tracing functions. It is brought up when the first tracing occurs and stays up until you close it. **TRACEWINDOW** can be set to a particular window to cause the tracing formation to print there.

TRACEREGION [Variable]

The trace window is first created in the region **TRACEREGION**.

Creating Breaks with **BREAK1**

The basic function of the break package is **BREAK1**, which creates a break. A break appears to be a regular executive, with the prompt “:”, but **BREAK1** also detects and interpretes break commands (see the Break Commands section above).

(**BREAK1** *BRKEXP BRKWHEN BRKFN BRKCOMS BRKTYPE ERRORN*) [NLambda Function]

If *BRKWHEN* (evaluated) is non-**NIL**, a break occurs and commands are then taken from *BRKCOMS* or the terminal and interpreted. All inputs not recognized by **BREAK1** are simply passed on to the programmer's assistant.

If *BRKWHEN* is **NIL**, *BRKEXP* is evaluated and returned as the value of **BREAK1**, without causing a break.

When a break occurs, if *ERRORN* is a list whose **CAR** is a number, **ERRORMESS** (see the Signalling Errors section below) is called to print an identifying message. If *ERRORN* is a list whose **CAR** is not a number, **ERRORMESS1** (see the Signalling Errors section below) is called. Otherwise, no preliminary message is printed. Following this, the message (*BRKFN* broken) is printed.

MEDLEY REFERENCE MANUAL

Since `BREAK1` itself calls functions, when one of these is broken, an infinite loop would occur. `BREAK1` detects this situation, and prints `Break within a break on FN`, and then simply calls the function without going into a break.

The commands `GO`, `!GO`, `OK`, `!OK`, `RETURN` and `↑` are the only ways to leave `BREAK1`. The command `EVAL` causes `BRKEXP` to be evaluated, and saves the value on the variable `!VALUE`. Other commands can be defined for `BREAK1` via `BREAKMACROS` (below).

`BRKTYPE` is `NIL` for user breaks, `INTERRUPT` for Control-H breaks, and `ERRORX` for error breaks. For breaks when `BRKTYPE` is not `NIL`, `BREAK1` will clear and save the input buffer. If the break returns a value (i.e., is not aborted via `↑` or Control-D) the input buffer is restored.

The fourth argument to `BREAK1` is `BRKCOMS`, a list of break commands that `BREAK1` interprets and executes as though they were keyboard input. One can think of `BRKCOMS` as another input file which always has priority over the keyboard. Whenever `BRKCOMS` = `NIL`, `BREAK1` reads its next command from the keyboard. Whenever `BRKCOMS` is non-`NIL`, `BREAK1` takes `(CAR BRKCOMS)` as its next command and sets `BRKCOMS` to `(CDR BRKCOMS)`. For example, suppose you wished to see the value of the variable `X` after a function was evaluated. You could set up a break with `BRKCOMS` = `(EVAL (PRINT X) OK)`, which would have the desired effect. If `BRKCOMS` is non-`NIL`, the value of a break command is not printed. If you desire to see a value, you must print it yourself, as in the above example. The function `TRACE` (see Chapter 15) uses `BRKCOMS`: it sets up a break with two commands; the first one prints the arguments of the function, or whatever you specify, and the second is the command `GO`, which causes the function to be evaluated and its value printed.

Note: If an error occurs while interpreting the `BRKCOMS` commands, `BRKCOMS` is set to `NIL`, and a full interactive break occurs.

The break package has a facility for redirecting output to a file. All output resulting from `BRKCOMS` is output to the value of the variable `BRKFILE`, which should be the name of an open file. Output due to user type-in is not affected, and will always go to the terminal. `BRKFILE` is initially `T`.

BREAKMACROS

[Variable]

`BREAKMACROS` is a list of the form `((NAME COM1 ... COM1) (NAME COM2 ... COM2) ...)`. Whenever an atomic command is given to `BREAK1`, it first searches the list `BREAKMACROS` for the command. If the command is equal to `NAME`, `BREAK1` simply appends the corresponding commands to the front of `BRKCOMS`, and goes on. If the command is not found on `BREAKMACROS`, `BREAK1` then checks to see if it is one of the built in commands, and finally, treats it as a function or variable as before.

If the command is not the name of a defined function, bound variable, or `LISPX` command, `BREAK1` will attempt spelling correction using `BREAKCOMSLST` as a spelling list. If spelling correction is unsuccessful, `BREAK1` will go ahead and call `LISPX` anyway, since the atom may also be a misspelled history command.

ERRORS AND DEBUGGING

For example, the command `ARGS` could be defined by including on `BREAKMACROS` the form:

```
(ARGS (PRINT (VARIABLES LASTPOS T)))
```

(BREAKREAD TYPE)

[Function]

Useful within `BREAKMACROS` for reading arguments. If `BRKCOMS` is non-NIL (the command in which the call to `BREAKREAD` appears was not typed in), returns the next break command from `BRKCOMS`, and sets `BRKCOMS` to `(CDR BRKCOMS)`.

If `BRKCOMS` is NIL (the command was typed in), then `BREAKREAD` returns either the rest of the commands on the line as a list (if `TYPE = LINE`) or just the next command on the line (if `TYPE` is not `LINE`).

For example, the `BT` command is defined as `(BAKTRACE LASTPOS NIL (BREAKREAD 'LINE) 0 T)`. Thus, if you type `BT`, the third argument to `BAKTRACE` is NIL. If you type `BT SUBRP`, the third argument is `(SUBRP)`.

BREAKRESETFORMS

[Variable]

If you are developing programs that change the way a user and Medley normally interact (e.g., change or disable the interrupt or line-editing characters, turn off echoing, etc.), debugging them by breaking or tracing may be difficult, because Medley might be in a “funny” state at the time of the break. `BREAKRESETFORMS` is designed to solve this problem. You put in `BREAKRESETFORMS` expressions suitable for use in conjunction with `RESETFORM` or `RESETSAVE` (see Changing and Restoring System State below). When a break occurs, `BREAK1` evaluates each expression on `BREAKRESETFORMS` *before* any interaction with the terminal, and saves the values. When the break expression is evaluated via an `EVAL`, `OK`, or `GO`, `BREAK1` first restores the state of the system with respect to the various expressions on `BREAKRESETFORMS`. When control returns to `BREAK1`, the expressions on `BREAKRESETFORMS` are *again* evaluated, and their values saved. When the break is exited with an `OK`, `GO`, `RETURN`, or `↑` command, by typing `Control-D`, or by a `RETFROM` or `RETEVAL` you type in, `BREAK1` again restores state. Thus the net effect is to make the break invisible with respect to your programs, but nevertheless allow you to interact in the break in the normal fashion.

All user type-in is scanned to make the operations undoable, as described in Chapter 13. At this point, `RETFROMs` and `RETEVALs` are also noticed. However, if you type in an expression which calls a function that then does a `RETFROM`, this `RETFROM` will not be noticed, and the effects of `BREAKRESETFORMS` will *not* be reversed.

As mentioned earlier, `BREAK1` detects “Break within a break” situations, and avoids infinite loops. If the loop occurs because of an error, `BREAK1` simply rebinds `BREAKRESETFORMS` to NIL, and calls `HELP`. This situation most frequently occurs when there is a bug in a function called by `BREAKRESETFORMS`.

`SETQ` expressions can also be included on `BREAKRESETFORMS` for saving and restoring system parameters, e.g. `(SETQ LISPXHISTORY NIL)`, `(SETQ DWIMFLG NIL)`, etc. These are handled specially by `BREAK1` in that the current value of the variable is saved before the `SETQ` is executed, and upon restoration, the variable is set back to this value.

Signalling Errors

With the Medley release, Interlisp errors use the Xerox Common Lisp (XCL) error system. Most of the functions still exist for compatibility with previous releases, but the underlying machinery has changed. There are some incompatible differences, especially with respect to error numbers. All errors are now handled by signalling an object of type `XCL:CONDITION`. This means the error numbers generated are different from the old Interlisp method of registered numbers for well-known errors and error messages for all other errors. The mapping from Interlisp errors to Lisp error conditions is listed in the Error List sections below. The obsolete error numbers still generate error messages, but they are useless.

(**ERRORX** *ERXM*) [Function]

Calls `CL:ERROR` after first converting *ERXM* into a condition. If *ERXM* is `NIL` the value of `*LAST-CONDITION*` is used. If *ERXM* is an Interlisp error descriptor, it is first converted to a condition. If *ERXM* is already a condition, it is passed along unchanged. **ERRORX** also sets a proceed case for `XCL:PROCEED`, which will attempt to re-evaluate the caller of **ERRORX**, much as `OK` did in older versions of the break package.

(**ERROR** *MESS MESS NOBREAK*) [Function]

Prints *MESS* (using `PRIN1`), followed by a space if *MESS* is an atom, otherwise a carriage return. Then *MESS* is printed (using `PRIN1` if *MESS* is a string; otherwise `PRINT`). For example, `(ERROR "NON-NUMERIC ARG" T)` prints

```
NON-NUMERIC ARG
T
```

and `(ERROR 'FOO "NOT A FUNCTION")` prints `FOO NOT A FUNCTION`. If both *MESS* and *MESS* are `NIL`, the message printed is simply `ERROR`.

If *NOBREAK* = `T`, **ERROR** prints its message and then calls **ERROR!** (below). Otherwise it calls `(ERRORX ' (17 (MESS . MESS)) )`, i.e., generates error number 17, in which case the decision as to whether to break, and whether to print a message, is handled as any other error.

If the value of `HELPFLAG` (see the Controlling When to Break section above) is `BREAK!`, a break will always occur, irregardless of the value of *NOBREAK*.

If **ERROR** causes a break, the “break expression” is `(ERROR MESS MESS NOBREAK)`. Using the `GO`, `OK`, `,` or `EVAL` break commands (see the Break Commands section above) will simply call **ERROR** again. It is sometimes helpful to design programs that call **ERROR** such that if the call to **ERROR** returns (as the result of using the `RETURN` break command), the operation is tried again. This lets you fix any problems within the break environment, and try to continue the operation.

(**HELP** *MESS MESS BRKTYPE*) [Function]

Prints *MESS* and *MESS* similar to **ERROR**, and then calls `BREAK1` passing *BRKTYPE* as the *BRKTYPE* argument. If both *MESS* and *MESS* are `NIL`, `Help!` is used for the message. **HELP** is a convenient way to program a default condition, or to terminate some portion of a program which the computation is theoretically never supposed to reach.

ERRORS AND DEBUGGING

(SHOULDNT MESS)	[Function]
Useful in situations when a program detects a condition that should never occur. Calls HELP with the message arguments <i>MESS</i> and "Shouldn't happen!" and a BRKTYPE argument of 'ERRORX.	
(ERROR!)	[Function]
Equivalent to XCL:ABORT, except that if no ERRORSET or XCL:CATCH-ABORT is found, it unwinds to the top of the process.	
(RESET)	[Function]
Programmable Control-D; immediately returns to the top level.	
LAST-CONDITION	[Variable]
Value is the condition object most recently signaled.	
(SETERRORN NUM MESS)	[Function]
Converts its arguments into a condition, then sets the value of *LAST-CONDITION* to the result.	
(ERRORMESS U)	[Function]
Prints message corresponding to its first argument. For example, (ERRORMESS ' (17 T)) would print: T is not a LIST	
(ERRORMESS1 MESS MESS MESS)	[Function]
Prints the message corresponding to a HELP or ERROR break.	
(ERRORSTRING X)	[Function]
Returns as a new string the message corresponding to error number X, e.g., (ERRORSTRING 10) = "NON-NUMERIC ARG".	

Catching Errors

All error conditions are not caused by program bugs. For some programs, it is reasonable for some errors to occur (such as file not found errors) and it is possible for the program to handle the error itself. There are a number of functions that allow a program to "catch" errors, rather than abort the computation or cause a break.

(ERRORSET FORM FLAG)	[Function]
Performs (EVAL FORM). If no error occurs in the evaluation of FORM, the value of ERRORSET is a list containing one element, the value of (EVAL FORM). If an error did occur, the value of ERRORSET is NIL.	
ERRORSET is a lambda function, so its arguments are evaluated <i>before</i> it is entered, i.e., (ERRORSET X) means EVAL is called with the <i>value</i> of X. In most cases, ERSETQ and NLSETQ (below) are more useful.	

MEDLEY REFERENCE MANUAL

Note: Beginning with the Medley release, there are no longer frames named `ERRORSET` on the stack and any programs that explicitly look for them must be changed.

Performance Note: When a call to `ERSETQ` or `NLSETQ` is compiled, the form to be evaluated is compiled as a separate function. However, compiling a call to `ERRORSET` does not compile *FORM*. Therefore, if *FORM* performs a lengthy computation, using `ERSETQ` or `NLSETQ` can be much more efficient than using `ERRORSET`.

The argument *FLAG* controls the printing of error messages if an error occurs. If a *break* occurs below an `ERRORSET`, the message is printed regardless of the value of *FLAG*.

If *FLAG* = `T`, the error message is printed; if *FLAG* = `NIL`, the error message is not printed (unless `NLSETQGAG` is `NIL`, see below).

If *FLAG* = `INTERNAL`, this `ERRORSET` is ignored for the purpose of deciding whether or not to break or print a message (see the Controlling When to Break section above). However, the `ERRORSET` is in effect for the purpose of flow of control, i.e., if an error occurs, this `ERRORSET` returns `NIL`.

If *FLAG* = `NOBREAK`, no break will occur, even if the time criterion for breaking is met (the Controlling When to Break section above). *FLAG* = `NOBREAK` will *not* prevent a break from occurring if the error occurs more than `HELPDEPTH` function calls below the errorset, since `BREAKCHECK` will stop searching before it reaches the `ERRORSET`. To guarantee that no break occurs, you would also either have to reset `HELPDEPTH` or `HELPFLAG`.

(**ERSETQ** *FORM*) [NLambda Function]

Evaluates *FORM*, letting a break happen if an error occurs, but 9[^] brings you back to the `ERSETQ`. Performs (`ERRORSET 'FORM T`), printing error messages.

(**NLSETQ** *FORM*) [NLambda Function]

Evaluates *FORM*, without breaking, returning `NIL` if an error occurs or a list containing *FORM* if no error occurs. Performs (`ERRORSET 'FORM NIL`), without printing error messages.

NLSETQGAG [Variable]

If `NLSETQGAG` is `NIL`, error messages will print, regardless of the *FLAG* argument of `ERRORSET`. `NLSETQGAG` effectively changes all `NLSETQs` to `ERSETQs`. `NLSETQGAG` is initially `T`.

Changing and Restoring System State

In Medley, a computation can be interrupted / aborted at any point due to an error, or more forcefully, because a Control-D was typed, causing return to the top level. This situation creates problems for programs that need to perform a computation with the system in a “different state”, e.g., different radix, input file, readtable, etc. but want to be able to restore the state when the computation has completed. While program errors and Control-E are “caught” by `ERRORSETs`, Control-D is not. The program could redefine Control-D as a user interrupt (see Chapter 30), check for it, reenable it, and

ERRORS AND DEBUGGING

call RESET or something similar. Thus the system may be left in its changed state as a result of the computation being aborted. The following functions address this problem.

These functions cannot handle the situation where their environment is exited via anything other than a normal return, an error, or a reset. Therefore, a RETEVAL, RETFROM, RESUME, etc., will never be seen.

(**RESETLST** *FORM* . . . *FORM*)

[NLambda NoSpread Function]

RESETLST evaluates its arguments in order, after setting up an ERRORSET so that any reset operations performed by RESETSAVE (see below) are restored when the forms have been evaluated (or an error occurs, or a Control-D is typed). If no error occurs, the value of RESETLST is the value of *FORM*, otherwise RESETLST generates an error (after performing the necessary restorations).

RESETLST compiles open.

(**RESETSAVE** *X Y*)

[NLambda NoSpread Function]

RESETSAVE is used within a call to RESETLST to change the system state by calling a function or setting a variable, while specifying how to restore the original system state when the RESETLST is exited (normally, or with an error or Control-D).

If *X* is atomic, resets the top level value of *X* to the value of *Y*. For example, (RESETSAVE LISPXHISTORY EDITHISTORY) resets the value of LISPXHISTORY to the value of EDITHISTORY, and provides for the original value of LISPXHISTORY to be restored when the RESETLST completes operation, (or an error occurs, or a Control-D is typed).

Note: If the variable is simply rebound, the RESETSAVE will not affect the most recent binding but will change only the top level value, and therefore probably not have the intended effect.

If *X* is not atomic, it is a form that is evaluated. If *Y* is NIL, *X* must return as its value its “former state”, so that the effect of evaluating the form can be reversed, and the system state can be restored, by applying CAR of *X* to the value of *X*. For example, (RESETSAVE (RADIX 8)) performs (RADIX 8), and provides for RADIX to be reset to its original value when the RESETLST completes by applying RADIX to the value returned by (RADIX 8).

In the special case that CAR of *X* is SETQ, the SETQ is transparent for the purposes of RESETSAVE, i.e. you could also have written (RESETSAVE (SETQ X (RADIX 8))), and restoration would be performed by applying RADIX, not SETQ, to the previous value of RADIX.

If *Y* is not NIL, it is evaluated (before *X*), and its *value* is used as the restoring expression. This is useful for functions which do not return their “previous setting”. For example,

[RESETSAVE (SETBRK . . .) (LIST 'SETBRK (GETBRK]

will restore the break characters by applying SETBRK to the value returned by (GETBRK), which was computed before the (SETBRK . . .) expression was evaluated. The restoration expression is “evaluated” by *applying* its CAR to its CDR. This insures that the “arguments” in the CDR are not evaluated again.

MEDLEY REFERENCE MANUAL

If *X* is *NIL*, *Y* is still treated as a restoration expression. Therefore,

```
(RESETSAVE NIL (LIST 'CLOSEF FILE))
```

will cause *FILE* to be closed when the *RESETLST* that the *RESETSAVE* is under completes (or an error occurs or a Control-D is typed).

RESETSAVE can be called when *not* under a *RESETLST*. In this case, the restoration is performed at the next *RESET*, i.e., Control-D or call to *RESET*. In other words, there is an “implicit” *RESETLST* at the top-level executive.

RESETSAVE compiles open. Its value is not a “useful” quantity.

(RESETVAR VAR NEWVALUE FORM)

[NLambda Function]

Simplified form of *RESETLST* and *RESETSAVE* for resetting and restoring global variables. Equivalent to *(RESETLST (RESETSAVE VAR NEWVALUE) FORM)*. For example, *(RESETVAR LISPXHISTORY EDITHISTORY (FOO))* resets *LISPXHISTORY* to the value of *EDITHISTORY* while evaluating *(FOO)*. *RESETVAR* compiles open. If no error occurs, its value is the value of *FORM*.

(RESETVARS VARSLST E ... E)

[NLambda NoSpread Function]

Similar to *PROG*, except that the variables in *VARSLST* are global variables. In a deep bound system (like Medley), each variable is “rebound” using *RESETSAVE*.

In a shallow bound system (like Interlisp-10) *RESETVARS* and *PROG* are identical, except that the compiler insures that variables bound in a *RESETVARS* are declared as *SPECVARS* (see Chapter 18).

RESETVARS, like *GETATOMVAL* and *SETATOMVAL* (see Chapter 2), is provided to permit compatibility (i.e. transportability) between a shallow bound and deep bound system with respect to conceptually global variables.

Note: Like *PROG*, *RESETVARS* returns *NIL* unless a *RETURN* statement is executed.

(RESETFORM RESETFORM FORM FORM ... FORM)

[NLambda NoSpread Function]

Simplified form of *RESETLST* and *RESETSAVE* for resetting a system state when the corresponding function returns as its value the “previous setting.” Equivalent to *(RESETLST (RESETSAVE RESETFORM) FORM FORM ... FORM)*. For example, *(RESETFORM (RADIX 8) (FOO))*. *RESETFORM* compiles open. If no error occurs, it returns the value returned by *FORM*.

For some applications, the restoration operation must be different depending on whether the computation completed successfully or was aborted somehow (e.g., by an error or by typing Control-D). To facilitate this, while the restoration operation is being performed, the value of *RESETSTATE* is bound to *NIL*, *ERROR*, *RESET*, or *HARDRESET* depending on whether the exit was normal, due to an error, due to a reset (i.e., Control-D), or due to call to *HARDRESET* (see Chapter 23). As an example of the use of *RESETSTATE*,

```
(RESETLST
  (RESETSAVE (INFILE X)
    (LIST ' [LAMBDA (FL)
```

ERRORS AND DEBUGGING

```
(COND ((EQ RESETSTATE 'RESET)
      (CLOSEF FL)
      (DELFIL FL)
      (X))
      FORMS)
```

will cause *X* to be closed and deleted only if a Control-D was typed during the execution of *FORMS*.

When specifying complicated restoring expressions, it is often necessary to use the old value of the saving expression. For example, the following expression will set the primary input file (to *FL*) and execute some forms, but reset the primary input file only if an error or Control-D occurs.

```
(RESETLST
  (SETQ TEM (INPUT FL))
  (RESETSAVE NIL
    (LIST '(LAMBDA (X) (AND RESETSTATE (INPUT X)))
          TEM))
  FORMS)
```

So that you will not have to explicitly save the old value, the variable *OLDVALUE* is bound at the time the restoring operation is performed to the value of the saving expression. Using this, the previous example could be recoded as:

```
(RESETLST
  (RESETSAVE (INPUT FL)
    '(AND RESETSTATE (INPUT OLDVALUE)))
  FORMS)
```

As mentioned earlier, restoring is performed by applying *CAR* of the restoring expression to the *CDR*, so *RESETSTATE* and *(INPUT OLDVALUE)* will not be evaluated by the *APPLY*. This particular example works because *AND* is an *nlambda* function that explicitly evaluates its arguments, so *APPLY*ing *AND* to *(RESETSTATE (INPUT OLDVALUE))* is the same as *EVAL*ing *(AND RESETSTATE (INPUT OLDVALUE))*. *PROGN* also has this property, so you can use a lambda function as a restoring form by enclosing it within a *PROGN*.

The function *RESETUNDO* (see Chapter 13) can be used in conjunction with *RESETLST* and *RESETSAVE* to provide a way of specifying that the system be restored to its prior state by *undoing* the side effects of the computations performed under the *RESETLST*.

Error List

There are currently fifty-plus types of errors in Medley. Some of these errors are implementation dependent, i.e., appear in Medley but may not appear in other Interlisp systems. The error number is set internally by the code that detects the error before it calls the error handling functions, and is used by *ERRORMESS* for printing error messages.

Most errors will print the offending expression as part of the error message. Error number 18 (Control-B) always causes a break (unless *HELPFLAG* is *NIL*). All other errors cause breaks if *BREAKCHECK* returns *T* (see Controlling When to Break above).

The following error messages are arranged numerically with the printed message next to the error number. *X* is the offending expression in each error message. The obsolete error numbers still generate error messages, but they aren't particularly useful. For information on how to use the Common Lisp error conditions in your own programs, see *Common Lisp: the Language* by Steele.

MEDLEY REFERENCE MANUAL

- 0 **Obsolete.**
- 1 **Obsolete.**
- 2 **Stack Overflow**
Occurs when computation is too deep, either with respect to number of function calls, or number of variable bindings. Usually because of a non-terminating recursive computation, i.e., a bug. Condition type: `STACK-OVERFLOW`.
- 3 **RETURN to nonexistent block: X**
Call to RETURN when not inside of an interpreted PROG. Condition type: `ILLEGAL-RETURN`.
- 4 **X is not a LIST**
RPLACA called on a non-list. Condition type: `XCL:SIMPLE-TYPE-ERROR` *culprit* :EXPECTED-TYPE 'LIST
- 5 **Device error: X**
An error with the local disk drive. Condition type: `XCL:SIMPLE-DEVICE-ERROR` *message*
- 6 **Serious condition XCL:ATTEMPT-TO-CHANGE-CONSTANT occurred.**
Via SET or SETQ. Condition type: `XCL:ATTEMPT-TO-CHANGE-CONSTANT`
- 7 **Attempt to rplac NIL with X**
Attempt either to RPLACA or to RPLACD NIL with something other than NIL. Condition type: `XCL:ATTEMPT-TO-RPLAC-NIL` *message*
- 8 **GO to a nonexistent tag: X.**
GO when not inside of a PROG, or GO to nonexistent label. Condition type: `ILLEGAL-GO` *tag*
- 9 **File won't open: X**
From OPENSTREAM (see Chapter 24). Condition type: `XCL:FILE-WONT-OPEN` *pathname*
- 10 **X is not a NUMBER**
A numeric function e.g., PLUS, TIMES, GREATERP, expected a number and didn't get one. Condition type: `XCL:SIMPLE-TYPE-ERROR` *culprit* :EXPECTED TYPE 'CL:NUMBER
- 11 **Symbol name too long**
Attempted to create a symbol (via PACK, or typing one in, or reading from a file) with too many characters. In Medley, the maximum number of characters in a symbol is 255. Condition type: `XCL:SYMBOL-NAME-TOO-LONG`
- 12 **Symbol hash table full**
No room for any more (new) atoms. Condition type: `XCL:SYMBOL-HT-FULL`
- 13 **Stream not open: X**
From an I/O function, e.g., READ, PRINT, CLOSEP. Condition type: `XCL:STREAM-NOT-OPEN` *stream*
- 14 **X is not a SYMBOL.**
SETQ, PUTPROP, GETTOPVAL, etc., given a non-atomic argument. Condition type: `XCL:SIMPLE-TYPE-ERROR` *culprit* :EXPECTED-TYPE 'CL:SYMBOL

ERRORS AND DEBUGGING

- 15 **Obsolete**
- 16 **End of file X**

From an input function, e.g., READ, READC, RATOM. After the error occurs, the file will still be left open. Condition type: END-OF-FILE *stream*
- 17 **X varying messages.**

Call to ERROR (see Signalling Errors above). Condition type: INTERLISP-ERROR MESSAGE
- 18 **Obsolete**
- 19 **Illegal stack arg: X**

A stack function expected a stack position and was given something else. This might occur if the arguments to a stack function are reversed. Also occurs if you specified a stack position with a function name, and that function was not found on the stack (see Chapter 11). Condition type: ILLEGAL-STACK-ARG *arg*.
- 20 **Obsolete**
- 21 **Array space full**

System will first initiate a garbage collection of array space, and if no array space is reclaimed, will then generate this error. Condition type: XCL:ARRAY-SPACE-FULL.
- 22 **File system resources exceeded: X**

Includes no more disk space, disk quota exceeded, directory full, etc. Condition type: XCL:FS-RESOURCE-EXCEEDED
- 23 **File not found**

File name does not correspond to a file in the corresponding directory. Can also occur if file name is ambiguous. Condition type: XCL:FILE-NOT-FOUND *pathname*
- 24 **Obsolete**
- 25 **Invalid argument: X**

A form ends in a non-list other than NIL, e.g., (CONS T . 3). Condition type: INVALID-ARGUMENT-LIST *argument*
- 26 **Hash table full: X**

See hash array functions, Chapter 6. Condition type: XCL:HASH-TABLE-FULL *table*
- 27 **Invalid argument: X**

Catch-all error. Currently used by PUTD, EVALA, ARG, FUNARG, etc. Condition type: INVALID-ARGUMENT-LIST *argument*
- 28 **X is not a ARRAYP.**

ELT or SETA given an argument that is not a legal array (see Chapter 5). Condition type: XCL:SIMPLE-TYPE-ERROR *culprit* :EXPECTED-TYPE 'ARRAYP
- 29 **Obsolete**
- 30 **Stack ptr ahs been released NOBIND**

A released stack pointer was supplied as a stack descriptor for a purpose other than as a stack pointer to be re-used (see Chapter 11). Condition type: STACK-POINTER-RELEASED *name*

MEDLEY REFERENCE MANUAL

- 31 **Serious condition XCL:STORAGE-EXHAUSTED occurred.**
Following a garbage collection, if not enough words have been collected, and there is no un-allocated space left in the system, this error is generated. Condition type: XCL:STORAGE-EXHAUSTED
- 32 Obsolete
- 33 Obsolete
- 34 **No more data types available**
All available user data types have been allocated (see Chapter 8). Condition type: XCL:DATA-TYPES-EXHAUSTED
- 35 **Serious condition XCL:ATTEMPT-TO-CHANGE-CONSTANT occurred.**
In a PROG or LAMBDA expression. Condition type: XCL:ATTEMPT-TO-CHANGE-CONSTANT
- 36 Obsolete
- 37 Obsolete
- 38 **X is not a READTABLEP.**
The argument was expected to be a valid read table (see Chapter 25). Condition type: XCL:SIMPLE-TYPE-ERROR *culprit* :EXPECTED-TYPE 'READTABLEP
- 39 **X is not a TERMTABLEP.**
The argument was expected to be a valid terminal table (see Chapter 30). Condition type: XCL:SIMPLE-TYPE-ERROR *culprit* :EXPECTED-TYPE 'TERMTABLEP
- 40 Obsolete
- 41 **Protection violation: X**
Attempt to open a file that you do not have access to. Also reference to unassigned device. Condition type: XCL:FS-PROTECTION-VIOLATION
- 42 **Invalid pathname: X**
Illegal character in file specification, illegal syntax, e.g. two ;'s etc. Condition type: XCL:INVALID-PATHNAME *pathname*
- 43 Obsolete
- 44 **X is an unbound variable**
This occurs when a variable (symbol) was used which had neither a stack binding (wasn't an argument to a function nor a PROG variable) nor a top level value. The "culprit" ((CADR ERRORMESS)) is the symbol. If DWIM corrects the error, no error occurs and the error number is not set. However, if an error is going to occur, whether or not it will cause a break, the error number will be set. Condition type: UNBOUND-VARIABLE *name*
- 45 **Serious condition UNDEFINED-CAR-OF-FORM occurred.**
Undefined function error. This occurs when a form is evaluated whose function position (CAR) does not have a definition as a function. Condition type: UNDEFINE-CAR-OF FORM *function*

ERRORS AND DEBUGGING

46 ***X* varying messages.**

This error is generated if APPLY is given an undefined function. Culprit is *(LIST FN ARGS)*
Condition type: UNDEFINED-FUNCTION-IN-APPLY

47 **CONTROL E**

Control-E was typed. Condition type: XCL:CONTROL-E-INTERRUPT

48 **Floating point underflow.**

Underflow during floating-point operation. Condition type: XCL:FLOATING-UNDERFLOW

49 **Floating point overflow.**

Overflow during floating-point operation. Condition type: XCL:OVERFLOW

50 **Obsolete**

51 ***X* is not a HASH-TABLE**

Hash array operations given an argument that is not a hash array. Condition type:
XCL:SIMPLE-TYPE-ERROR *culprit* :EXPECTED-TYPE 'CL:HASH-TABLE

52 **Too many arguments to *X***

Too many arguments given to a lambda-spread, lambda-nospread, or nlambda-spread function.

Medley does not cause an error if more arguments are passed to a function than it is defined with. This argument occurs when more individual arguments are passed to a function than Medley can store on the stack at once. The limit is currently 80 arguments.

In addition, many system functions, e.g., DEFINE, ARGLIST, ADVISE, LOG, EXPT, etc, also generate errors with appropriate messages by calling ERROR (see Signalling Errors above) which causes error number 17. Condition type: TOO-MANY-ARGUMENTS *callee* :MAXIMUM
CL:CALL-ARGUMENTS-LIMIT

MEDLEY REFERENCE MANUAL

[This page intentionally left blank]

15. BREAKING, TRACING, AND ADVISING

Medley provides several different facilities for modifying the behavior of a function without actually editing its definition. By “breaking” a function, you can cause breaks to occur at various times in the running of an incomplete program, so that the program state can be inspected. “Tracing” a function causes information to be printed every time the function is entered or exited.

“Advising” is a facility for specifying longer-term function modifications. Even system functions can be changed through advising.

Breaking Functions and Debugging

Debugging a collection of Lisp functions involves isolating problems within particular functions and/or determining when and where incorrect data are being generated and transmitted. In the Medley, there are three facilities which allow you to (temporarily) modify selected function definitions so that you can follow the flow of control in your programs, and obtain this debugging information. All three redefine functions in terms of a system function, `BREAK1` (see Chapter 14).

`BREAK` modifies the definition of a function *FN*, so that whenever *FN* is called and a break condition (user-defined) is satisfied, a function break occurs. You can then interrogate the state of the machine, perform any computation, and continue or return from the call.

`TRACE` modifies a definition of a function *FN* so that whenever *FN* is called, its arguments (or some other user-specified values) are printed. When the value of *FN* is computed it is printed also. `TRACE` is a special case of `BREAK`.

`BREAKIN` allows you to insert a breakpoint inside an expression defining a function. When the breakpoint is reached and if a break condition (defined by you) is satisfied, a temporary halt occurs and you can again investigate the state of the computation.

The following two examples illustrate these facilities. In the first example, the function `FACTORIAL` is traced. `TRACE` redefines `FACTORIAL` so that it print its arguments and value, and then goes on with the computation. When an error occurs on the fifth recursion, a full interactive break occurs. The situation is then the same as though `(BREAK FACTORIAL)` had been performed instead of `(TRACE FACTORIAL)`, now you can evaluate various Interlisp forms and direct the course of the computation. In this case, the variable *N* is examined, and `BREAK1` is instructed to return 1 as the value of this cell to `FACTORIAL`. The rest of the tracing proceeds without incident. Presumably, `FACTORIAL` would be edited to change *L* to 1.

```
←PP FACTORIAL
(FACTORIAL
  [LAMBDA (N)
    (COND
      ((ZEROP N)
        L)
      (T (ITIMES N (FACTORIAL (SUB1 N))
        FACTORIAL
        ← (TRACE FACTORIAL)
        (FACTORIAL)
        ← (FACTORIAL 4)
```

MEDLEY REFERENCE MANUAL

```

FACTORIAL:
N = 4
  FACTORIAL:
  N = 3
    FACTORIAL:
    N = 2
      FACTORIAL:
      N = 1
        FACTORIAL:
        N = 0
UNBOUND ATOM
L
(FACTORIAL BROKEN)
:N
0
:RETURN 1
      FACTORIAL = 1
    FACTORIAL = 1
  FACTORIAL = 2
  FACTORIAL = 6
FACTORIAL = 24
24
←

```

In the second example, a non-recursive definition of FACTORIAL has been constructed. BREAKIN is used to insert a call to BREAK1 just after the PROG label LOOP. This break is to occur only on the last two iterations, when N is less than 2. When the break occurs, in trying to look at the value of N, NN is mistakenly typed. The break is maintained, however, and no damage is done. After examining N and M the computation is allowed to continue by typing OK. A second break occurs after the next iteration, this time with N = 0. When this break is released, the function FACTORIAL returns its value of 120.

```

←PP FACTORIAL
(FACTORIAL
  [LAMBDA (N)
    (PROG ((M 1))
      LOOP (COND
        ((ZEROP N)
          (RETURN M))
        (SETQ M (ITIMES M N))
        (SETQ N (SUB1 N))
        (GO LOOP])
      FACTORIAL
      ←(BREAKIN FACTORIAL (AFTER LOOP) (ILESSP N 2])
      SEARCHING...
      FACTORIAL
      ←((FACTORIAL 5)
        ((FACTORIAL) BROKEN)
        :NN
        U.B.A.
        NN
        (FACTORIAL BROKEN AFTER LOOP)
        :N
        1
        :M
        120

```

BREAKING, TRACING, AND ADVISING

```
:OK
(FACTORIAL)

((FACTORIAL) BROKEN)
:N
0
:OK
(FACTORIAL)
120
←
```

Note: BREAK and TRACE can also be used on CLISP words which appear as CAR of form, e.g. FETCH, REPLACE, IF, FOR, DO, etc., even though these are not implemented as functions. For conditional breaking, you can refer to the entire expression via the variable EXP, e.g. (BREAK (FOR (MEMB 'UNTIL EXP))).

(**BREAK0** *FN WHEN COMS* — —) [Function]

Sets up a break on the function *FN*; returns *FN*. If *FN* is not defined, returns (*FN* NOT DEFINED).

The value of *WHEN*, if non-NIL, should be an expression that is evaluated whenever *FN* is entered. If the value of the expression is non-NIL, a break is entered, otherwise the function simply called and returns without causing a break. This provides the means of conditionally breaking a function.

The value of *COMS*, if non-NIL, should be a list of break commands, that are interpreted and executed if a break occurs. (See the *BRKCOMS* argument to BREAK1, Chapter 14.)

BREAK0 sets up a break by doing the following:

- Redefines *FN* as a call to BREAK1 (Chapter 14), passing an equivalent definition of *FN*, *WHEN*, *FN*, and *COMS* as the *BRKEXP*, *BRKWHEN*, *BRKFN*, and *BRKCOMS* arguments to BREAK1

- Defines a GENSYM (Chapter 2) with the original definition of *FN*, and puts it on the property list of *FN* under the property BROKEN

- Puts the form (BREAK0 *WHEN COMS*) on the property list of *FN* under the property BRKINFO (for use in conjunction with REBREAK)

- Adds *FN* to the front of the list BROKENFNS.

If *FN* is non-atomic and of the form (*FN* IN *FN*), BREAK0 breaks every call to *FN* from within *FN*. This is useful for breaking on a function that is called from many places, but where one is only interested in the call from a specific function, e.g., (RPLACA IN FOO), (PRINT IN FIE), etc. It is similar to BREAKIN described below, but can be performed even when *FN* is compiled or blockcompiled, whereas BREAKIN only works on interpreted functions. If *FN* is not found in *FN*, BREAK0 returns the value (*FN* NOT FOUND IN *FN*).

BREAK0 breaks one function *inside* another by first calling a function which changes the name of *FN* wherever it appears inside of *FN* to that of a new function, *FN1-IN-FN2*,

MEDLEY REFERENCE MANUAL

which is initially given the same function definition as *FN*. Then `BREAK0` proceeds to break on *FN1-IN-FN2* exactly as described above. In addition to breaking *FN1-IN-FN2* and adding *FN1-IN-FN2* to the list `BROKENFNS`, `BREAK0` adds *FN1* to the property value for the property `NAMESCHANGED` on the property list of *FN* and puts *(FN . FN)* on the property list of *FN1-IN-FN2* under the property `ALIAS`. This will enable `UNBREAK` to recognize what changes have been made and restore the function *FN* to its original state.

If *FN* is nonatomic and not of the above form, `BREAK0` is called for each member of *FN* using the same values for *WHEN*, *COMS*, and *FILE*. This distributivity permits you to specify complicated break conditions on several functions. For example,

```
(BREAK0 ' (FOO1 ((PRINT PRIN1) IN (FOO2 FOO3)))
      ' (NEQ X T)
      ' (EVAL ?= (Y Z) OK) )
```

will break on `FOO1`, `PRINT-IN-FOO2`, `PRINT-IN-FOO3`, `PRIN1-IN-FOO2` and `PRIN1-IN-FOO3`.

If *FN* is non-atomic, the value of `BREAK0` is a list of the functions broken.

(BREAK X) [NLambda NoSpread Function]

For each atomic argument, it performs `(BREAK0 ATOM T)`. For each list, it performs `(APPLY 'BREAK0 LIST)`. For example, `(BREAK FOO1 (FOO2 (GREATERP N 5) (EVAL)))` is equivalent to `(BREAK0 'FOO1 T)` and `(BREAK0 'FOO2 ' (GREATERP N 5) ' (EVAL))`.

(TRACE X) [NLambda NoSpread Function]

For each atomic argument, it performs `(BREAK0 ATOM T ' (TRACE ?= NIL GO))`. The flag `TRACE` is checked for in `BREAK1` and causes the message "*FUNCTION* :" to be printed instead of `(FUNCTIONBROKEN)`.

For each list argument, `CAR` is the function to be traced, and `CDR` the forms to be viewed, i.e., `TRACE` performs:

```
(BREAK0 (CAR LIST) T (LIST 'TRACE ' ?= (CDR LIST) 'GO))
```

For example, `(TRACE FOO1 (FOO2 Y))` causes both `FOO1` and `FOO2` to be traced. All the arguments of `FOO1` are printed; only the value of `Y` is printed for `FOO2`. In the special case when you want to see *only* the value, you can perform `(TRACE (FUNCTION))`. This sets up a break with commands `(TRACE ?= (NIL) GO)`.

Note: You can always call `BREAK0` to obtain combination of options of `BREAK1` not directly available with `BREAK` and `TRACE`. These two functions merely provide convenient ways of calling `BREAK0`, and will serve for most uses.

Note: `BREAK0`, `BREAK`, and `TRACE` print a warning if you try to modify a function on the list `UNSAFE.TO.MODIFY.FNS` (Chapter 10).

(BREAKIN FN WHERE WHEN COMS) [NLambda Function]

`BREAKIN` enables you to insert a break, i.e., a call to `BREAK1` (Chapter 14), at a specified location in the interpreted function *FN*. `BREAKIN` can be used to insert breaks before or

BREAKING, TRACING, AND ADVISING

after PROG labels, particular SETQ expressions, or even the evaluation of a variable. This is because BREAKIN operates by calling the editor and actually inserting a call to BREAK1 at a specified point *inside* of the function. If *FN* is a compiled function, BREAKIN returns (*FN* UNBREAKABLE) as its value.

WHEN should be an expression that is evaluated whenever the break is entered. If the value of the expression is non-NIL, a break is entered, otherwise the function simply called and returns without causing a break. This provides the means of creating a conditional break. For BREAKIN, unlike BREAK0, if *WHEN* is NIL, it defaults to T.

COMS, if non-NIL, should be a list of break commands, that are interpreted and executed if a break occurs. (See the *BRKCONMS* argument to BREAK1, Chapter 14.)

WHERE specifies where in the definition of *FN* the call to BREAK1 is to be inserted. *WHERE* should be a list of the form (BEFORE ...), (AFTER ...), or (AROUND ...). You specify where the break is to be inserted by a sequence of editor commands, preceded by one of the symbols BEFORE, AFTER, or AROUND, which BREAKIN uses to determine what to do once the editor has found the specified point, i.e., put the call to BREAK1 BEFORE that point, AFTER that point, or AROUND that point. For example, (BEFORE COND) will insert a break before the first occurrence of COND, (AFTER COND 2 1) will insert a break after the predicate in the first COND clause, (AFTER BF (SETQ X &)) after the *last* place X is set. Note that (BEFORE TTY:) or (AFTER TTY:) permit you to type in commands to the editor, locate the correct point, and verify it, and exit from the editor with OK. BREAKIN then inserts the break BEFORE, AFTER, or AROUND that point.

Note: A STOP command typed to TTY: produces the same effect as an unsuccessful edit command in the original specification, e.g., (BEFORE CONDD). In both cases, the editor aborts, and BREAKIN types (NOT FOUND).

If *WHERE* is (BEFORE ...) or (AFTER ...), the break expression is NIL, since the value of the break is irrelevant. For (AROUND ...), the break expression will be the indicated form. In this case, you can use the EVAL command to evaluate that form, and examine its value, before allowing the computation to proceed. For example, if you inserted a break after a COND predicate, e.g., (AFTER (EQUAL X Y)), you would be powerless to alter the flow of computation if the predicate were not true, since the break would not be reached. However, by breaking (AROUND (EQUAL X Y)), you can evaluate the break expression, i.e., (EQUAL X Y), look at its value, and return something else if desired.

If *FN* is interpreted, BREAKIN types SEARCHING... while it calls the editor. If the location specified by *WHERE* is not found, BREAKIN types (NOT FOUND) and exits. If it is found, BREAKIN puts T under the property BROKEN-IN and (*WHERE WHEN COMS*) under the the property BRKINFO on the property list of *FN*, and adds *FN* to the front of the list BROKENFNs.

Multiple break points, can be inserted with a single call to BREAKIN by using a list of the form ((BEFORE ...) ... (AROUND ...)) for *WHERE*. It is also possible to call BREAK or TRACE on a function which has been modified by BREAKIN, and conversely to BREAKIN a function which has been redefined by a call to BREAK or TRACE.

MEDLEY REFERENCE MANUAL

The message typed for a `BREAKIN` break is `((FN) BROKEN)`, where *FN* is the name of the function inside of which the break was inserted. Any error, or typing control-E, will cause the full identifying message to be printed, e.g., `(FOO BROKEN AFTER COND 2 1)`.

A special check is made to avoid inserting a break inside of an expression headed by any member of the list `NOBREAKS`, initialized to `(GO QUOTE *)`, since this break would never be activated. For example, if `(GO L)` appears before the label `L`, `BREAKIN (AFTER L)` will not insert the break inside of the `GO` expression, but skip this occurrence of `L` and go on to the next `L`, in this case the label `L`. Similarly, for `BEFORE` or `AFTER` breaks, `BREAKIN` checks to make sure that the break is being inserted at a “safe” place. For example, if you request a break `(AFTER X)` in `(PROG ... (SETQ X &) ...)`, the break will actually be inserted after `(SETQ X &)`, and a message printed to this effect, e.g., `BREAK INSERTED AFTER (SETQ X &)`.

(UNBREAK X) [NLambda NoSpread Function]

`UNBREAK` takes an indefinite number of functions modified by `BREAK`, `TRACE`, or `BREAKIN` and restores them to their original state by calling `UNBREAK0`. Returns list of values of `UNBREAK0`.

`(UNBREAK)` will unbreak all functions on `BROKENFNs`, in reverse order. It first sets `BRKINFOLST` to `NIL`.

`(UNBREAK T)` unbreaks just the first function on `BROKENFNs`, i.e., the most recently broken function.

(UNBREAK0 FN —) [Function]

Restores *FN* to its original state. If *FN* was not broken, value is `(NOT BROKEN)` and no changes are made. If *FN* was modified by `BREAKIN`, `UNBREAKIN` is called to edit it back to its original state. If *FN* was created from `(FN IN FN)`, (i.e., if it has a property `ALIAS`), the function in which *FN* appears is restored to its original state. All dummy functions that were created by the break are eliminated. Adds property value of `BRKINFO` to the front of `BRKINFOLST`.

Note: `(UNBREAK0 '(FN IN FN))` is allowed: `UNBREAK0` will operate on `(FN -IN-FN)` instead.

(UNBREAKIN FN) [Function]

Performs the appropriate editing operations to eliminate all changes made by `BREAKIN`. *FN* may be either the name or definition of a function. Value is *FN*.

`UNBREAKIN` is automatically called by `UNBREAK` if *FN* has property `BROKEN-IN` with value `T` on its property list.

(REBREAK X) [NLambda NoSpread Function]

Nlambda nospread function for rebreaking functions that were previously broken without having to respecify the break information. For each function on *X*, `REBREAK` searches `BRKINFOLST` for break(s) and performs the corresponding operation. Value is a

BREAKING, TRACING, AND ADVISING

list of values corresponding to calls to `BREAK0` or `BREAKIN`. If no information is found for a particular function, returns `(FN - NO BREAK INFORMATION SAVED)`.

`(REBREAK)` rebreaks everything on `BRKINFOLST`, so `(REBREAK)` is the inverse of `(UNBREAK)`.

`(REBREAK T)` rebreaks just the first break on `BRKINFOLST`, i.e., the function most recently unbroken.

(CHANGENAME *FN FROM TO*)

[Function]

Replaces all occurrences of *FROM* by *TO* in the definition of *FN*. If *FN* is defined by an expr definition, `CHANGENAME` performs `(ESUBST TO FROM (GETD FN))` (see Chapter 16). If *FN* is compiled, `CHANGENAME` searches the literals of *FN* (and all of its compiler generated subfunctions), replacing each occurrence of *FROM* with *TO*.

Note that *FROM* and *TO* do not have to be functions, e.g., they can be names of variables, or any other literals.

`CHANGENAME` returns *FN* if at least one instance of *FROM* was found, otherwise `NIL`.

(VIRGINFN *FN FLG*)

[Function]

The function that knows how to restore functions to their original state regardless of any amount of breaks, breakins, advising, compiling and saving exprs, etc. It is used by `PRETTYPRINT`, `DEFINE`, and the compiler.

If *FLG* = `NIL`, as for `PRETTYPRINT`, it does not modify the definition of *FN* in the process of producing a “clean” version of the definition; it works on a copy.

If *FLG* = `T`, as for the compiler and `DEFINE`, it physically restores the function to its original state, and prints the changes it is making, e.g., `FOO UNBROKEN`, `FOO UNADVISED`, `FOO NAMES RESTORED`, etc.

Returns the virgin function definition.

Advising

The operation of advising gives you a way of modifying a function without necessarily knowing how the function works or even what it does. Advising consists of modifying the *interface* between functions as opposed to modifying the function definition itself, as in editing. `BREAK`, `TRACE`, and `BREAKDOWN`, are examples of the use of this technique: they each modify user functions by placing relevant computations *between* the function and the rest of the programming environment.

The principal advantage of advising, aside from its convenience, is that it allows you to treat anyone’s functions as “black boxes,” and to modify them without concern for their contents or details of operations. For example, you could modify `SYSOUT` to set `SYSDATE` to the time and date of creation by `(ADVISE 'SYSOUT' (SETQ SYSDATE (DATE)))`.

As with `BREAK`, advising works equally well on compiled and interpreted functions. Similarly, it is possible to make a change which only operates when a function is called from some other specified function. For example, you can modify the interface between two particular functions, instead of the

MEDLEY REFERENCE MANUAL

interface between one function and the rest of the world. This latter feature is especially useful for changing the *internal* workings of a system function.

For example, suppose you wanted TIME (Chapter 22) to print the results of your measurements to the file FOO instead of the terminal. You can accomplish this by (ADVISE ' ((PRIN1 PRINT SPACES) IN TIME) 'BEFORE ' (SETQQ U FOO)).

Advising PRIN1, PRINT, or SPACES directly would have affected all calls to these frequently used functions, whereas advising ((PRIN1 PRINT SPACES) IN TIME) affects just those calls to PRIN1, PRINT, and SPACES from TIME.

Advice can also be specified to operate after a function has been evaluated. The value of the body of the original function can be obtained from the variable !VALUE, as with BREAK1.

Implementation of Advising

After a function has been modified several times by ADVISE, it will look like:

```
(LAMBDA arguments
  (PROG (!VALUE)
    (SETQ !VALUE
      (PROG NIL
        advice1
          .
          .      advice before
          .
        advicem
        (RETURN BODY)))
    advice1
      .
      .      advice after
      .
    advicem
    (RETURN !VALUE)))
```

where *BODY* is equivalent to the original definition. If *FN* was originally an expr definition, *BODY* is the body of the definition, otherwise a form using a GENSYM which is defined with the original definition.

The structure of a function modified by ADVISE allows a piece of advice to bypass the original definition by using the function RETURN. For example, if (COND ((ATOM X) (RETURN Y))) were one of the pieces of advice *before* a function, and this function was entered with X atomic, Y would be returned as the value of the inner PROG, !VALUE would be set to Y, and control passed to the advice, if any, to be executed *AFTER* the function. If this same piece of advice appeared *after* the function, Y would be returned as the value of the entire advised function.

The advice (COND ((ATOM X) (SETQ !VALUE Y))) *after* the function would have a similar effect, but the rest of the advice *after* the function would still be executed.

Note: Actually, ADVISE uses its own versions of PROG, SETQ, and RETURN, (called ADV-PROG, ADV-SETQ, and ADV-RETURN) to enable advising these functions.

BREAKING, TRACING, AND ADVISING

Advise Functions

ADVISE is a function of four arguments: *FN*, *WHEN*, *WHERE*, and *WHAT*. *FN* is the function to be modified by advising, *WHAT* is the modification, or piece of advice. *WHEN* is either BEFORE, AFTER, or AROUND, and indicates whether the advice is to operate BEFORE, AFTER, or AROUND the body of the function definition. *WHERE* specifies exactly where in the list of advice the new advice is to be placed, e.g., FIRST, or (BEFORE PRINT) meaning before the advice containing PRINT, or (AFTER 3) meaning after the third piece of advice, or even (: TTY:). If *WHERE* is specified, ADVISE first checks to see if it is one of LAST, BOTTOM, END, FIRST, or TOP, and operates accordingly. Otherwise, it constructs an appropriate edit command and calls the editor to insert the advice at the corresponding location.

Both *WHEN* and *WHERE* are optional arguments, in the sense that they can be omitted in the call to ADVISE. In other words, ADVISE can be thought of as a function of two arguments (ADVISE *FN* *WHAT*), or a function of three arguments: (ADVISE *FN* *WHEN* *WHAT*), or a function of four arguments: (ADVISE *FN* *WHEN* *WHERE* *WHAT*). Note that the advice is always the *last* argument. If *WHEN* = NIL, BEFORE is used. If *WHERE* = NIL, LAST is used.

(ADVISE *FN* *WHEN* *WHERE* *WHAT*)

[Function]

FN is the function to be advised, *WHEN* = BEFORE, AFTER, or AROUND, *WHERE* specifies where in the advice list the advice is to be inserted, and *WHAT* is the piece of advice.

If *FN* is of the form (*FN* IN *FN*), *FN* is changed to *FN1*-IN-*FN2* throughout *FN*, as with break, and then *FN1*-IN-*FN2* is used in place of *FN*. If *FN* and/or *FN* are lists, they are distributed as with BREAK0.

If *FN* is broken, it is unbroken before advising.

If *FN* is not defined, an error is generated, NOT A FUNCTION.

If *FN* is being advised for the first time, i.e., if (GETP *FN* 'ADVISED) = NIL, a GENSYM is generated and stored on the property list of *FN* under the property ADVISED, and the GENSYM is defined with the original definition of *FN*. An appropriate expr definition is then created for *FN*, using private versions of PROG, SETQ, and RETURN, so that these functions can also be advised. Finally, *FN* is added to the (front of) ADVISEDFNs, so that (UNADVISE T) always unadvise the last function advised.

If *FN* has been advised before, it is moved to the front of ADVISEDFNs.

If *WHEN* = BEFORE or AFTER, the advice is inserted in *FN*'s definition either BEFORE or AFTER the original body of the function. Within that context, its position is determined by *WHERE*. If *WHERE* = LAST, BOTTOM, END, or NIL, the advice is added following all other advice, if any. If *WHERE* = FIRST or TOP, the advice is inserted as the first piece of advice. Otherwise, *WHERE* is treated as a command for the editor, similar to BREAKIN, e.g., (BEFORE 3), (AFTER PRINT).

If *WHEN* = AROUND, the body is substituted for * in the advice, and the result becomes the new body, e.g., (ADVISE 'FOO 'AROUND '(RESETFORM (OUTPUT T) *)). Note that if several pieces of AROUND advice are specified, earlier ones will be embedded inside later ones. The value of *WHERE* is ignored.

MEDLEY REFERENCE MANUAL

Finally (LIST WHEN WHERE WHAT) is added (by ADDPROP) to the value of property ADVISE on the property list of *FN*, so that a record of all the changes is available for subsequent use in readvising. Note that this property value is a list of the advice in order of calls to ADVISE, not necessarily in order of appearance of the advice in the definition of *FN*.

The value of ADVISE is *FN*.

If *FN* is non-atomic, every function in *FN* is advised with the same values (but copies) for WHEN, WHERE, and WHAT. In this case, ADVISE returns a list of individual functions.

Note: Advised functions can be broken. However if a function is broken at the time it is advised, it is first unbroken. Similarly, advised functions can be edited, including their advice. UNADVISE will still restore the function to its unadvised state, but any changes to the body of the definition will survive. Since the advice stored on the property list is the same structure as the advice inserted in the function, editing of advice can be performed on either the function's definition or its property list.

(UNADVISE *X*)

[NLambda NoSpread Function]

An nlambda nospread like UNBREAK. It takes an indefinite number of functions and restores them to their original unadvised state, including removing the properties added by ADVISE. UNADVISE saves on the list ADVINFOLST enough information to allow restoring a function to its advised state using READVICE. ADVINFOLST and READVICE thus correspond to BRKINFOLST and REBREAK. If a function contains the property READVICE, UNADVISE moves the current value of the property ADVISE to READVICE.

(UNADVISE) unadvise all functions on ADVISEDFNS in reverse order, so that the most recently advised function is unadvised last. It first sets ADVINFOLST to NIL.

(UNADVISE *T*) unadvise the first function of ADVISEDFNS, i.e., the most recently advised function.

(READVICE *X*)

[NLambda NoSpread Function]

An nlambda nospread like REBREAK for restoring a function to its advised state without having to specify all the advise information. For each function on *X*, READVICE retrieves the advise information either from the property READVICE for that function, or from ADVINFOLST, and performs the corresponding advise operation(s). It also stores this information on the property READVICE if not already there. If no information is found for a particular function, value is (*FN*-NO ADVISE SAVED).

(READVICE) readvises everything on ADVINFOLST.

(READVICE *T*) readvises the first function on ADVINFOLST, i.e., the function most recently unadvised.

A difference between ADVISE, UNADVISE, and READVICE versus BREAK, UNBREAK, and REBREAK, is that if a function is not rebroken between successive (UNBREAK)s, its break information is forgotten. However, once READVICE is called on a function, that function's advice is permanently saved on its property list (under READVICE); subsequent calls to

BREAKING, TRACING, AND ADVISING

UNADVISE will not remove it. In fact, calls to UNADVISE update the property READVICE with the current value of the property ADVICE, so that the sequence READVICE, ADVISE, UNADVISE causes the augmented advice to become permanent. The sequence READVICE, ADVISE, READVICE removes the “intermediate advice” by restoring the function to its earlier state.

(**ADVISEDUMP** *X FLG*)

[Function]

Used by PRETTYDEF when given a command of the form (ADVISE ...) or (ADVICE ...). If *FLG* = T, ADVISEDUMP writes both a DEFLIST and a READVICE; this corresponds to (ADVISE ...). If *FLG* = NIL, only the DEFLIST is written; this corresponds to (ADVICE ...). In either case, ADVISEDUMP copies the advise information to the property READVICE, thereby making it “permanent” as described above.

SEdit - The EDITOR

16. SEdit - The Structure Editor

Medley's code editors are "structure" editors—they know how to take advantage of Lisp code being represented as lists. One is a display editor named SEdit and the other is a TTY-based editor.

Starting the Editor

The editor is normally called using the following functions:

(DF *FN*) [NLambda NoSpread Function]

Edit the definition of the function *FN*. *DF* handles exceptional cases (the function is broken or advised, the definition is on the property list, the function needs to be loaded from a file, etc.) the same as *EDITF* (see below).

If you call *DF* with a name that has no function definition, you are prompted with a choice of definers to use.

(DV *VAR*) [NLambda NoSpread Function]

Edit the value of the variable *VAR*.

(DP *NAME PROP*) [NLambda NoSpread Function]

Edit property *PROP* of the symbol *NAME*. If *PROP* is not given, the whole property list of *NAME* is edited.

(DC *FILE*) [NLambda NoSpread Function]

Edit the file package commands (or "filecoms," see Chapter 17) for the file *FILE*.

(ED *NAME OPTIONS*) [Function]

This function finds out what kind of definition *NAME* has and lets you edit it. If *NAME* has more than one definition (e.g., it's both a function and a macro), you will be prompted for the right one. If *NAME* has no definition, you'll be asked what kind of definition to create.

Choosing Your Editor

The default editor may be set with *EDITMODE*:

INTERLISP-D REFERENCE MANUAL

(EDITMODE *NEWMODE*)

[Function]

If *NEWMODE* is *DISPLAY*, sets the default editor to be SEdit; or the teletype editor (if *NEWMODE* is *TELETYPE*). Returns the previous setting. If *NEWMODE* is *NIL*, returns the previous setting without setting a new editor.

SEdit - The Structure Editor

SEdit is a structure editor. You use a structure editor when you want to edit objects instead of text. SEdit is a part of the environment and operates directly on objects in the system you are running. SEdit behaves differently depending on the type of objects you are editing.

Common Lisp definitions: SEdit always edits a copy of a Common Lisp definition. The changes made while you edit a function will not be installed until the edit session is complete.

For example, when you edit a Common Lisp function, you edit the definition of the function and not the executable version of the function. When you end the session the comments will be stripped of the definition and the definition will be installed as the executable version of the function.

Interlisp functions and macros: SEdit edits the actual structure that will be run, except editing the source for a compiled function. In this case, changes are made and the function is unsaved when you complete the edit session.

All other structures: Variables, property lists and other structures are edited directly in place, i.e. SEdit installs all changes as they are made.

If you make a severe editing error, you can abort the edit session with an Abort command (see Command Keys, below). This command undoes all changes from the beginning of the edit session and exits from SEdit without changing your environment.

If you change the definition of an object that is being edited in an SEdit window, Medley will ask you if you want to throw away the changes made there.

SEdit supports the standard Copy-Select mechanism in Medley.

An SEdit Session

Whenever you call SEdit, a new SEdit window is created. This SEdit window has its own process. You can make edits in the window, shrink it while you do something else, expand it and edit some more, and finally close the window when you are done.

Throughout an edit session, SEdit remembers everything that you do in a change history. You can undo and redo edits sequentially. When you end the edit session, SEdit forgets this information and installs the changes in the system.

You signal the end of the session in the following ways:

- Close the window.
- Shrink the window. If you expand the window again, you can continue editing.
- Issue a Completion Command, see below.

SEdit Carets

There are two carets in SEdit, the edit caret and the structure caret. The edit caret appears when characters are edited within a single symbol, string, or comment. Anything you type will appear at the edit caret as part of the item it's in. The edit caret looks like this:

(a  b)

The structure caret appears when the edit point is between symbols (or strings or comments), so that anything you type will go into a new one. It looks like this:

(a  b)

SEdit changes the caret frequently, depending on where the caret is positioned. The left mouse button positions the edit caret. The middle mouse button positions the structure caret.

The Mouse

The left mouse button selects parts of Lisp structures. The middle mouse button selects whole Lisp structures.

For example; select the Q in LEQ below by pressing the left mouse button when the pointer is over the Q.

(LEQ  n 1)

INTERLISP-D REFERENCE MANUAL

Any characters you type in now will be appended to the symbol LEQ.

Selecting the same letter with the middle mouse button selects the whole symbol (this matches TEdit's character / word selection convention), and sets a structure caret between the LEQ and the n:

(LEQ▲ n 1)

Any characters you type in now will form a new symbol between the LEQ and the n.

Larger structures can be selected in two ways. Use the middle mouse button to position the mouse cursor on the parenthesis of the list you want to edit. Press the mouse button multiple times, without moving the mouse, extends the selection. In the previous example, if the middle button was pressed twice, the list (LEQ . . .) would be selected:

(LEQ n 1)

Press the button a third time and you will select the list containing the (LEQ n 1) to be selected.

The right mouse button positions the mouse cursor for selecting sequences of structures or substructures. Extended selections are indicated by a box enclosing the structures selected. The selection extends in the same mode as the original selection. That is, if the original selection was a character selection, the right button will be used to select more characters in the same atom. Extended selections also have the property of being marked for pending deletion. That is, the selection takes the place of the caret, and anything typed in is inserted in place of the selection.

For example, selecting the E by pressing the left mouse button and selecting the Q by pressing the right mouse button will produce:

(LEQ n 1)

Similarly, pressing the middle mouse button and then selecting with the right mouse button extends the selection by whole structures. In our example, pressing the middle mouse button to select LEQ and pressing the right mouse button to select the 1 will produce:

(LEQ n 1)

This is not the same as selecting the entire list, as above. Instead, the elements in the list are collectively selected, but the list itself is not.

Gaps

SEdit requires that everything edited must have an underlying Lisp structure at all times. Some characters, such as single quote “`'`” have no meaning by themselves, but must be followed by something more. When you type such a character, SEdit puts a “gap” where the rest of the input should go. When you type, the gap is automatically replaced.

A gap looks like: `-x-`

After you type a quote, the gap looks like this: `'` with the gap marked for pending deletion.

Broken Atoms

When you type an atom (a symbol or a number), SEdit saves the characters you type until you are finished. Typing any character that cannot belong to an atom, like a space or open parenthesis, ends the atom. SEdit then tries to create an atom with the characters you just typed, just as if they were read by the Lisp reader. The atom then becomes part of the structure you're editing.

If an error occurs when SEdit reads the atom, SEdit creates a structure called a Broken-Atom. A Broken-Atom looks and behaves just like a normal atom, but is printed in italics to tell you that something is wrong.

SEdit creates a Broken-Atom when the characters typed don't make a legal atom. For example, the characters "DECLARE:" can't be a symbol because the colon is a package specifier, but the form is not correct for a package-qualified symbol. Similarly, the characters "#b123" cannot represent an integer in base two, because 2 and 3 aren't legal digits in base two, so SEdit would make a Broken-Atom that looks like *#b123*.

You can edit Broken-Atoms just like real atoms. Whenever you finish editing a Broken-Atom, SEdit again tries to create an atom from the characters. If SEdit succeeds, it reprints the atom in SEdit's default font, rather than in italics. Be sure to correct any Broken-Atoms you create before exiting SEdit, since Broken-Atoms do not behave in any useful way outside SEdit.

Special Characters

Some characters have special meanings in Lisp, and are therefore treated specially by SEdit. SEdit must always have a complete structure to work on at any level of the edit. This means that SEdit needs a special way to type in structures such as lists, strings, and quoted objects. In most instances these structures can be typed in just as they would be to a regular Exec, but in the following cases this is not possible.

INTERLISP-D REFERENCE MANUAL

Lists: ()	Lists begin with an open parenthesis character "(". Typing an open parenthesis gives a balanced list. SEdit inserts both an open and a close parenthesis. The structure caret is placed between the two parentheses. List elements can be typed in at the structure caret. When a close parenthesis, ")" is typed, the caret will be moved outside the list, effectively finishing the list. Square bracket characters, "[" and "]", have no special meaning in SEdit, as they have no special meaning in Common Lisp.
Single Quote: '	
Backquote: `	
Comma: ,	
At Sign: ,@	
Dot: ,.	
Hash Quote: #'	All these characters are special macro characters in Common Lisp. When you type one, SEdit will echo the character followed by a gap, which you should then fill in.
Dotted Lists: (.)	Use period to enter dotted pairs. After you type a dot, SEdit prints a dot and a gap to fill in for the tail of the list. To dot an existing list, point the cursor between the last and second to last elements, and type a dot. To undot a list, select the tail of the list before the dot while holding down the SHIFT key.
Single escape: \ or %	Use the single escape characters to make symbols with special characters. The single escape character for Interlisp is "%". The single escape character for Common Lisp is "\". When you want to create a symbol with a special character in it you have to type a single escape character before you type the character itself. SEdit does not echo the single escape character until you type the following character. For example; create the Common Lisp symbol APAREN- (. Since SEdit normally will treat the "(" as the start of a new list you have to tell SEdit to treat it as an ordinary character. You do this by typing a "\" before you type the "(".
Multiple Escape:	Use the multiple escape character when you enter symbols with many special characters. SEdit always balances multiple escape characters. When you type one, SEdit adds another, with the caret between them. If you type a second vertical bar, the caret moves after it, but is still in the same symbol, so you can add more unescaped characters.

Comment: `;` A semicolon starts a comment. When you type a semicolon, an empty comment is inserted with the caret in position to type the comment. Comments can be edited like strings.

There are three levels of comments supported by SEdit: single-, double-, and triple-semicolon. Single-semicolon comments are formatted at the comment column, about three-quarters of the way across the window. Double-semicolon comments are formatted at the current indentation of the code they are in. Triple semicolon comments are formatted against the left margin. The level of a comment can be increased or decreased by pointing after the semicolon, and either typing another semicolon, or backspacing over the preceding semicolon. Comments can be placed anywhere in your Common Lisp code. However, in Interlisp code, they must follow the placement rules for Interlisp comments.

String: `"` Enter strings in SEdit by typing a double quote. SEdit balances the double quotes: When one is typed, SEdit produces a second, with the caret between the two. If you type a double-quote in the middle of a string, SEdit breaks the string in two, leaving the caret between them.

SEdit Commands

Enter SEdit commands either from the keyboard or from the SEdit menu. When possible, SEdit uses a named key on the keyboard, e.g., the DELETE key. Other commands are combinations of Meta, Control, and alphabetic keys. For the alphabetic command keys, either uppercase or lowercase will work.

There are two menus available, as an alternative means of invoking commands. They are the middle button popup menu, and the attached command menu. These menus are described in more detail below.

Meta-A Abort the session. Throw away the changes made to the form.

Meta-B Change the Print Base. Prompts for entry of the desired Print Base, in decimal. SEdit redisplay fixed point numbers in this new base.

Control-C Tell SEdit that this session is complete and compiles the definition being edited. The variable *COMPILE-FN* determines which function to use as compiler. See the Options section below.

Control-Meta-C Signals the system that this edit is complete, compiles the definition being editing, and closes the window.

DELETE Deletes the current selection.

INTERLISP-D REFERENCE MANUAL

Meta-E Evaluate the current selection. If the result is a structure, the inspector is called on it, allowing the user to choose how to look at the result. Otherwise, the result is printed in the SEdit prompt window. The evaluation is done in the process from which the edit session was started. Thus, while editing a function from a break window, evaluations are done in the context of the break.

FIND

Meta-F Find a specified structure, or sequence of structures. If there is a current selection, SEdit looks for the next occurrence of the selected structure. If there is no selection, SEdit prompts for the structure to find, and searches forward from the position of the caret. The found structure will be selected, so the Find command can be used to easily find the same structure again.

If a sequence of structures are selected, SEdit will look for the next occurrence of the same sequence. Similarly, when SEdit prompts for the structure to find, you can type a sequence of structures to look for.

The variable *WRAP-SEARCH* controls whether or not SEdit wraps around from the end of the structure being edited and continues searching from the beginning.

Control-Meta-F Find a specified structure, searching in reverse from the position of the caret.

HELP

Meta-H Show the argument list for the function currently selected, or currently being typed in, in the SEdit prompt window. If the argument list will not fit in the SEdit prompt window, it is displayed in the main Prompt Window.

Meta-I Inspect the current selection.

Meta-J Join any number of sequential Lisp objects of the same type into a single object of that type. Join is supported for atoms, strings, lists, and comments. In addition, SEdit permits joining of a sequence of atoms and strings, since either type can easily be coerced into the other. In this case, the result of the Join will be an atom if the first object in the selection is an atom, otherwise the result will be a string.

Control-L Redisplay the structure being edited.

SKIP-NEXT

Meta-N Select next gap in the structure.

Meta-O Edit the definition of the current selection. If the selected name has more than one type of definition, SEdit asks for the type to edit. If the selection has no definition, a menu pops up. This menu lets you specify the type of definition to create.

Control-Meta-O Perform a fast edit by calling ED with the CURRENT option.

Meta-P Change the current package for this edit. Prompt the user for a new package name. SEdit will redisplay atoms with respect to that package.

AGAIN

Meta-R Redo the edit change that was just undone. Redo only works directly following an Undo. Any number of Undo commands can be sequentially redone.

SHIFT-FIND

Meta-S Substitute a structure, or sequence of structures within the current selection. SEdit prompts you in the SEdit prompt window for the structures to replace, and the structures to replace with. The selection to substitute within must be a structure selection.

Control-Meta-S Remove all occurrences of a structure or sequence of structures within the current selection. SEdit prompts you for the structures to delete.

UNDO

Meta-U Undo the last edit. All changes in the the edit session are remembered, and can be undone sequentially.

Control-W Delete the previous atom or structure. If the caret is in the middle of an atom, deletes backward to the beginning of the atom only.

Control-X Tell SEdit that this session is complete. The SEdit window remains open.

EXPAND

Meta-X Replaces the current selection with its definition. This command can be used to expand macros and translate CLISP.

Control-Meta-X Tell SEdit that this session is complete Close the SEdit window.

Meta-Z Mutate. Prompt for a function and call this function with the current selection as the argument. The result is inserted into SEdit and made the current selection.

For example, you can replace a structure with its value by selecting it and mutating by EVAL.

Meta-; Convert old style comments in the selected structure to new style comments. The converter notices any list that begins with the symbol IL:* as an old style comment. Section 16.1.18, Options, describes the converter options.

Control-Meta-; Put the contents of a structure selection into a comment. This provides an easy way to "comment out" a chunk of code. The Extract command can be used to reverse this process, returning the comment to the structures contained therein.

Meta-/ Extract one level of structure from the current selection. If there is no selection, but there is a structure caret, the list containing the caret is used. This command can be used to strip the parentheses off a list, or to unquote a quoted structure, or to replace a comment with the contained structures.

Meta-'

Meta-`

Meta-,

Meta-.

Meta-@ or **Meta-2**

Meta-# or **Meta-3**

Meta-. Quote the current selection with the specified kind of quote.

Meta-Space

Meta-Return Scroll the current selection to the center of the window. Similarly, the Space or Return key can be used to normalize the caret.

Meta-)

Meta-0 Parenthesize the current selection, position the caret after the new list.

INTERLISP-D REFERENCE MANUAL

- Meta- (**
Meta-9 ParenthesizE the current selection, position the caret at the beginning of the new list.
- Meta-M** Attach a menu of common commands to the top of the SEdit window. Each SEdit window can have its own menu.

SEdit Command Mnemonics

Abort	Meta-A
Change Print Base	Meta-B
Complete	Control-X
Compile & Complete	Control-C
Close, Compile & Complete	Control-Meta-C
Convert Comment	Meta-;
Make Selection Comment	Control-Meta-;
Previous Delete	Control-W
Selection Delete	DELETE
Selection Dot Comma	Meta-.
Selection At Comma	Meta-@
Edit	Meta-O
Fast Edit	Control-Meta-O
Selection Eval	Meta-E
Macro Expand	Meta-X
Forward Find	Meta-F
Reverse Find	Control-Meta-F
Next Gap	Meta-N
Arglist Help	Meta-H
Inspect	Meta-I
Join	Meta-J
Attach Menu	Meta-M
Expression Mutate	Meta-Z
Change Package	Meta-P
Selection Left Parenthesize	Meta-(
Selection Right Parenthesize	Meta-)
Selection Pop	Meta-/
Selection Back Quote	Meta-'
Selection Hash Quote	Meta-#
Selection Quote	Meta-'
Redisplay	Control-L
Redo	Meta-R
Remove	Control-Meta-S
Substitute	Meta-S
Undo	Meta-U

SEdit Command Menu

When the mouse cursor is in the SEdit title bar and you press middle mouse button, a Help Menu of commands pops up. The menu looks like this:

Commands	
Abort	M-A
Done	C-X
Done & Compile	C-C
Done & Close	M-C-X
Done, Compile, & Close	M-C-C
Undo	M-U
Redo	M-R
Find	M-F
Reverse Find	M-C-F
Remove	M-C-S
Substitute	M-S
Find Gap	M-N
Arglist	M-H
Convert Comment	M-;
Edit	M-O
Eval	M-E
Expand	M-X
Extract	M-/
Inspect	M-I
Join	M-J
Mutate	M-Z
Parenthesize	M-(
Quote	M-'
Set Print-Base	M-B
Set Package	M-P
Attach Menu	M-M

The Help Menu lists each command and its corresponding Command Key. (C- stands for Control, M- for Meta.) The menu pops up with the mouse cursor next to the last command you used from the menu. This makes it easy to repeat a command.

SEdit Attached Menu

SEdit's Attached Command Menu contains the commonly used commands. Use the Meta-M keyboard command to bring up this menu. The menu can be closed, independently of the SEdit window. The menu looks like:

SEdit Command Menu					
EXIT	DONE	ABORT	PAREN	QUOTE	EXTRACT
UNDO	REDO	ARGLIST	EDIT	EVAL	EXPAND
PRINT-BASE 10 PACKAGE XCL-USER					
FIND:					
SUBSTITUTE:					

Menu commands work like the corresponding keyboard commands, except for Find and Substitute.

INTERLISP-D REFERENCE MANUAL

For Find, SEdit prompts in the menu window, next to the Find button, for the structures to find. Type in the structures then select Find again. The search begins from the caret position in the SEdit window.

Similarly, Substitute prompts next to the Find button for the structures to find, and next to the Substitute button for the structures to replace them with. After both have been typed in, selecting Substitute replaces all occurrences of the Find structures with the Substitute structures, within the current selection.

To selectively substitute, use Find to find the next potential substitution target. If you want to replace it, select Substitute. Otherwise, select Find again to go on.

Selecting either Find or Substitute with the right mouse button erases the old structure to find or substitute from the menu, and prompts for a new one.

SEdit Programmer's Interface

The following sections describe SEdit's programmer's interface. All symbols are external in the package `SEdit`.

SEdit Window Region Manager

SEdit provides user redefinable functions which control how SEdit chooses the region for a new edit window. In the following text there are a few concepts that you will have to be familiar with. They are:

The region stack. This is a stack of old used regions. The reason to keep these around is that the user probably was comfortable with the old position of the window, so when he starts a new SEdit it is a good bet that he will be happy with the old placement.

SEdit uses the respective value of the symbols `SEdit::DEFAULT-FONT`, `SEdit::ITALIC-FONT`, `SEdit::KEYWORD-FONT`, `SEdit::COMMENT-FONT`, and `SEdit::BROKEN-ATOM-FONT` when displaying an expression. The value of these symbols have to be font descriptors.

(GET-WINDOW-REGION *context reason name type*) [Function]

This function is called when SEdit wants to know where to place a window it is about to open. This happens whenever the user starts a new SEdit or expands an Sedit icon. The default behavior is to pop a window region off SEdit's stack of regions that have been used in the past. If the stack is empty, SEdit prompts for a new region.

context is the current editor context.

reason is one of `:CREATE` or `:EXPAND` depending on what action prompted the call to `GET-WINDOW-REGION`

name is the name of the structure to be edited.

type is the edit type of the calling context.

(SAVE-WINDOW-REGION *context reason name type region*) [Function]

This function is called whenever SEdit is finished with a region and wants to make the region available for other SEdits. This happens whenever an SEdit window is closed or shrunk, or when an SEdit Icon is closed. The default behavior is simply to push the region onto SEdit's stack of regions.

context is the current editor context.

reason is one of :CLOSE, :SHRINK, or :CLOSE-ICON or depending on what action prompted the call to SAVE-WINDOW-REGION

name is the name of the structure to be edited.

type is the edit type of the calling context.

region is the region to be pushed onto the region stack. If region is NIL the old region of the SEdit will be pushed top the region stack.

KEEP-WINDOW-REGION [Variable]

Default T. This flag determines the behavior of the default SEdit region manager, explained above, for shrinking and expanding windows. When set to T, shrinking an SEdit window will not give up that window's region; the icon will always expand back into the same region. When set to NIL, the window's region is made available for other SEdits when the window is shrunk. Then when an SEdit icon is expanded, the window will be reshaped to the next available region.

This variable is only used by the default implementations of the functions get-window-region and save-window-region. If these functions are redefined, this flag is no longer used.

Options

The following parameters can be set as desired.

WRAP-PARENS [Variable]

This SEdit pretty printer flag determines whether or not trailing close parenthesis characters,), are forced to be visible in the window without scrolling. By default it is set to NIL, meaning that close parens are allowed to "fall off" the right edge of the window. If set to T, the pretty printer will start a new line before the structure preceding the close parens, so that all the parens will be visible.

WRAP-SEARCH [Variable]

This flag determines whether or not SEdit find will wrap around to the top of the structure when it reaches the end, or vice versa in the case of reverse find. The default is NIL.

INTERLISP-D REFERENCE MANUAL

CLEAR-LINEAR-ON-COMPLETION [Variable]

This flag determines whether or not SEdit completely re-pretty prints the structure being edited when you complete the edit. The default value is `NIL`, meaning that SEdit reuses the pretty printing.

IGNORE-CHANGES-ON-COMPLETION [Variable]

Sometimes the structure that you are editing is changed by the system upon completion. Editdates are an example of this behavior. When this flag is `NIL`, the default, SEdit will redisplay the new structure, capturing the changes. When `T`, SEdit will ignore the fact that changes were made by the system and keep the old structure.

CONVERT-UPGRADE [Variable]

Default 100. When using Meta-; to convert old-style single- asterisk comments, if the length of the comment exceeds convert-upgrade characters, the comment is converted into a double semicolon comment. Otherwise, the comment is converted into a single semicolon comment.

Old-style double-asterisk comments are always converted into new-style triple-semicolon comments.

Control Functions

(RESET) [Function]

This function recomputes the SEdit edit environment. Any changes made in the font profile, or any changes made to SEdit's commands are captured by resetting. Close all SEdit windows before calling this function.

(ADD-COMMAND *key-code form &optional scroll? key-name command-name help-string*) [Function]

This function allows you to write your own SEdit keyboard commands. You can add commands to new keys, or you can redefine keys that SEdit already uses as command keys. If you mistakenly redefine an SEdit command, the function `Reset-Commands` will remove all user-added commands, leaving SEdit with its default set of commands.

key-code can be a character code, or any form acceptable to `il:charcode`.

form determines the function to be called when the key command is typed. It can be a symbol naming a function, or a list, whose first element is a symbol naming a function and the rest of the elements are extra arguments to the function. When the command is invoked, SEdit will apply the function to the edit context (SEdit's main data structure), the charcode that was typed, and any extra arguments supplied in *form*. The extra arguments do not get evaluated, but are useful as keywords or flags, depending on how the command was invoked. The command function must return `T` if it handled the command. If the function returns `NIL`, SEdit will ignore the command and insert the character typed.

The first optional argument, *scroll?*, determines whether or not SEdit scrolls the window after running the command. This argument defaults to *NIL*, meaning don't scroll. If the value of *scroll?* is *T*, SEdit will scroll the window to ensure that the caret is visible.

The rest of the optional arguments are used to add this command to SEdit's middle button menu. When the item is selected from the menu, the command function will be called as described above, with the *charcode* argument set to *NIL*.

key-name is a string to identify the key (combination) to be typed to invoke the command. For example "M-A" to represent the Meta-A key combination, and "C-M-A" for Control-Meta-A.

command-name is a string to identify the command function, and will appear in the menu next to the *key-name*.

help-string is a string to be printed in the prompt window when a mouse button is held down over the menu item.

After adding all the commands that you want, you must call *Reset-Commands* to install them.

For example:

```
(ADD-COMMAND "^U" (MY-CHANGE-CASE T) )  
(ADD-COMMAND "^Y" (MY-CHANGE-CASE NIL) )  
(ADD-COMMAND "1,R" MY-REMOVE-NIL  
  "M-R" "REMOVE NIL"  
  "REMOVE NIL FROM THE SELECTED STRUCTURE" ) )  
(RESET-COMMANDS)
```

will add three commands.

Suppose *MY-CHANGE-CASE* takes the arguments *context*, *charcode*, and *upper-case?*. *upper-case?* will be set to *T* when *MY-CHANGE-CASE* is called from Control-U, and *NIL* when called from Control-Y. *MY-REMOVE-NIL* will be called with only *context* and *charcode* arguments when you type Meta-R.

(RESET-COMMANDS) [Function]

This function installs all commands added by *add-command*. SEdits which are open at the time of the *reset-commands* will not see the new commands; only new SEdits will have the new commands available.

(DEFAULT-COMMANDS) [Function]

This function removes all commands added by *add-command*, leaving SEdit with its default set of commands. As in *reset-commands*, open SEdits will not be changed; only new SEdits will have the user commands removed.

INTERLISP-D REFERENCE MANUAL

(**GET-PROMPT-WINDOW** *context*) [Function]

Returns the attached prompt window for a particular SEdit.

(**GET-SELECTION** *context*) [Function]

This function returns two values: the selected structure, and the type of selection, one of NIL, T, or :SUB-LIST. The selection type NIL means there is not a valid selection (in this case the structure is meaningless). T means the selection is one complete structure. :SUB-LIST means a series of elements in a list is selected, in which case the structure returned is a list of the elements selected.

(**REPLACE-SELECTION** *context structure selection-type*) [Function]

This function replaces the current selection with a new structure, or multiple structures, by deleting the selection and then inserting the new structure(s). The selection-type argument must be one of T or :SUB-LIST. If T, the structure is inserted as one complete structure. If :SUB-LIST, the structure is treated as a list of elements, each of which is inserted.

EDIT-FN [Variable]

This function is called with the selected structure and the edit specified as arguments to Sedit options as its arguments from the Edit (M-O) command. It should start the editor as appropriate, or generate an error if the selection is not editable.

COMPILE-FN [Variable]

This function is called with the arguments *name*, *type*, and *body*, from the compile/completion commands. It should compile the definition, *body*, and install the code as appropriate.

(**SEdit** *structure props options*) [Function]

This function provides a means of starting SEdit directly. *structure* is the structure to be edited.

props is a property list, which may specify the following properties:

:NAME - the name of the object being edited

:TYPE - the file manager type of the object being edited. If NIL, SEdit will not call the file manager when it tries to refetch the definition it is editing. Instead, it will just continue to use the structure that it has.

:COMPLETION-FN - the function to be called when the edit session is completed. This function is called with the *context*, *structure*, and *changed?* arguments. *context* is SEdits main data structure. *structure* is the structure being edited. *changed?* specifies if any changes have been made, and is one of NIL, T, or :ABORT, where :ABORT means the user is aborting the edit and throwing away any changes made. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments for the function.

:ROOT-CHANGED-FN - the function to be called when the entire structure being edited is replaced with a new structure. This function is called with the new structure as its argument. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments that the function is applied to following the structure argument.

options is one or a list of any number of the following keywords:

:CLOSE-ON-COMPLETION - This option specifies that SEdit cannot remain active for multiple completions. That is, the SEdit window cannot be shrunk, and the completion commands that normally leave the window open will in this case close the window and terminate the edit.

:COMPILE-ON-COMPLETION - This option specifies that SEdit should call the *COMPILE-FN* to compile the definition being edited upon completion, regardless of the completion command used.

The TTY Editor

This editor the main code editor in pre-window-system versions of Interlisp. For that task, it has been replaced by SEdit.

However, the TTY Editor provides an excellent language for manipulating list structure and making large-scale code changes. For example, several tools for cleaning up code are written using TTY Editor calls to do the actual work.

TTY Editor Local Attention-Changing Commands

This section describes commands that change the current expression (i.e., change the edit chain) thereby "shifting the editor's attention." These commands depend only on the *structure* of the edit chain, as compared to the search commands (presented later), which search the contents of the structure.

UP

[Editor Command]

UP modifies the edit chain so that the old current expression (i.e., the one at the time UP was called) is the first element in the new current expression. If the current expression is the first element in the next higher expression UP simply does a 0. Otherwise UP adds the corresponding tail to the edit chain.

If a P command would cause the editor to type . . . before typing the current expression, ie., the current expression is a tail of the next higher expression, UP has no effect.

For example:

INTERLISP-D REFERENCE MANUAL

```

*PP
(COND ((NULL X) (RETURN Y)))
*1 P
COND
*UP P
(COND (& &))
*-1 P
(NULL X) (RETURN Y)
*UP P
... ((NULL X) (RETURN Y))
*UP P
... ((NULL X) (RETURN Y))
*F NULL P
(NULL X)
*UP P
(NULL X) (RETURN Y)
*UP P
... ((NULL X) (RETURN Y))

```

The execution of UP is straightforward, except in those cases where the current expression appears more than once in the next higher expression. For example, if the current expression is (A NIL B NIL C NIL) and you perform 4 followed by UP, the current expression should then be ... NIL C NIL). UP can determine which tail is the correct one because the commands that descend save the last tail on an internal editor variable, LASTAIL. Thus after the 4 command is executed, LASTAIL is (NIL C NIL). When UP is called, it first determines if the current expression is a tail of the next higher expression. If it is, UP is finished. Otherwise, UP computes (MEMB *CURRENT-EXPRESSION* *NEXT-HIGHER-EXPRESSION*) to obtain a tail beginning with the current expression. The current expression should *always* be either a tail or an element of the next higher expression. If it is neither, for example you have directly (and incorrectly) manipulated the edit chain, UP generates an error. If there are no other instances of the current expression in the next higher expression, this tail is the correct one. Otherwise UP uses LASTAIL to select the correct tail.

Occasionally you can get the edit chain into a state where LASTAIL cannot resolve the ambiguity, for example if there were two non-atomic structures in the same expression that were EQ, and you descended more than one level into one of them and then tried to come back out using UP. In this case, UP prints LOCATION UNCERTAIN and generates an error. Of course, we could have solved this problem completely in our implementation by saving at each descent *both* elements and tails. However, this would be a costly solution to a situation that arises infrequently, and when it does, has no detrimental effects. The LASTAIL solution is cheap and resolves almost all of the ambiguities.

$N(N > 1)$

[Editor Command]

Adds the N th element of the current expression to the front of the edit chain, thereby making it be the new current expression. Sets LASTAIL for use by UP. Generates an error if the current expression is not a list that contains at least N elements.

`-N (N >= 1)`

[Editor Command]

Adds the N th element from the end of the current expression to the front of the edit chain, thereby making it be the new current expression. Sets `LASTAIL` for use by `UP`. Generates an error if the current expression is not a list that contains at least N elements.

`0`

[Editor Command]

Sets the edit chain to `CDR` of the edit chain, thereby making the next higher expression be the new current expression. Generates an error if there is no higher expression, i.e., `CDR` of edit chain is `NIL`.

Note that `0` usually corresponds to going back to the next higher left parenthesis, but not always. For example:

```
*P
(A B C D E F B)
*3 UP P
... C D E F G)
*3 UP P
... E F G)
*0 P
... C D E F G)
```

If the intention is to go back to the next higher left parenthesis, regardless of any intervening tails, the command `!0` can be used.

`!0`

[Editor Command]

Does repeated `0`'s until it reaches a point where the current expression is *not* a tail of the next higher expression, i.e., always goes back to the next higher left parenthesis.

`↑`

[Editor Command]

Sets the edit chain to `LAST` of edit chain, thereby making the top level expression be the current expression. Never generates an error.

`NX`

[Editor Command]

Effectively does an `UP` followed by a `2`, thereby making the current expression be the next expression. Generates an error if the current expression is the last one in a list. (However, `!NX` described below will handle this case.)

`BK`

[Editor Command]

Makes the current expression be the previous expression in the next higher expression. Generates an error if the current expression is the first expression in a list.

INTERLISP-D REFERENCE MANUAL

For example:

```
*PP
(COND ((NULL X) (RETURN Y)))
*F RETURN P
(RETURN Y)
*BK P
(NULL X)
```

Both NX and BK operate by performing a !0 followed by an appropriate number, i.e., there won't be an extra tail above the new current expression, as there would be if NX operated by performing an UP followed by a 2.

(NX *N*) [Editor Command]

(*N* >= 1) Equivalent to *N* NX commands, except if an error occurs, the edit chain is not changed.

(BK *N*) [Editor Command]

(*N* >= 1) Equivalent to *N* BK commands, except if an error occurs, the edit chain is not changed.

Note: (NX -*N*) is equivalent to (BK *N*), and vice versa.

!NX [Editor Command]

Makes the current expression be the next expression at a higher level, i.e., goes through any number of right parentheses to get to the next expression. For example:

```
*PP
(PROG ((L L)
      (UF L))
 LP (COND
      ((NULL (SETQ L (CDR L)))
       (ERROR!))
      ([NULL (CDR (FMEMB (CAR L) (CADR L)
                        (GO LP)))]
       (EDITCOM (QUOTE NX))
       (SETQ UNFIND UF)
       (RETURN L)))
 *F CDR P
(CDR L)
*NX

NX ?
*!NX P
(ERROR!)
*!NX P
((NULL &) (GO LP))
*!NX P
(EDITCOM (QUOTE NX))
*
```

!NX operates by doing 0's until it reaches a stage where the current expression is *not* the last expression in the next higher expression, and then does a NX. Thus !NX always goes through at least one unmatched right parenthesis, and the new current expression is always on a different level, i.e., !NX and NX always produce different results. For example using the previous current expression:

```
*F CAR P
(CAR L)
*!NX P
(GO LP)
*\P P
(CAR L)
*NX P
(CADR L)
*
```

(NTH N)

[Editor Command]

(N ~= 0) Equivalent to N followed by UP, i.e., causes the list starting with the Mth element of the current expression (or Mth from the end if $N < 0$) to become the current expression. Causes an error if current expression does not have at least N elements.

(NTH 1) is a no-op, as is (NTH -L) where L is the length of the current expression.

line-feed

[Editor Command]

Moves to the "next" expression and prints it, i.e. performs a NX if possible, otherwise performs a !NX. (The latter case is indicated by first printing ">".)

Control-X

[Editor Command]

Control-X moves to the "previous" thing and then prints it, i.e. performs a BK if possible, otherwise a !0 followed by a BK.

Control-Z

[Editor Command]

Control-Z moves to the last expression and prints it, i.e. does -1 followed by P.

Line-feed, Control-X, and Control-Z are implemented as *immediate* read macros; as soon as they are read, they abort the current printout. They thus provide a convenient way of moving around in the editor. To facilitate using different control characters for those macros, the function SETTERMCHARS is provided (see below).

Commands That Search

All of the editor commands that search use the same pattern matching routine (the function `EDIT4E`, below). We will therefore begin our discussion of searching by describing the pattern match mechanism. A pattern *PAT* matches with *X* if any of the following conditions are true:

1. If *PAT* is EQ to *X*
2. If *PAT* is &
3. If *PAT* is a number and EQP to *X*
4. If *PAT* is a string and `(STREQUAL PAT X)` is true
5. If `(CAR PAT)` is the atom `*ANY*`, `(CDR PAT)` is a list of patterns, and one of the patterns on `(CDR PAT)` matches *X*.
6. If *PAT* is a literal atom or string containing one or more \$s (escapes), each \$ can match an indefinite number (including 0) of contiguous characters in the atom or string *X*, e.g., `VER$` matches both `VERYLONGATOM` and `"VERYLONGSTRING"` as do `$LONG$` (but not `$LONG`), and `$V$L$T$`. Note: the litatom \$ (escape) matches only with itself.
7. If *PAT* is a literal atom or string ending in \$\$ (escape, escape), *PAT* matches with the atom or string *X* if it is "close" to *PAT*, in the sense used by the spelling corrector (see Chapter 20). For example, `CONSS$$` matches with `CONS`, `CNONC$$` with `NCONC` or `NCONC1`.

The pattern matching routine always types a message of the form `=MATCHING-ITEM` to inform you of the object matched by a pattern of the above two types, unless `EDITQUIETFLG = T`. For example, if `VER$` matches `VERYLONGATOM`, the editor would print `=VERYLONGATOM`.

8. If `(CAR PAT)` is the atom `--`, *PAT* matches *X* if `(CDR PAT)` matches with some tail of *X*. For example, `(A -- (&))` will match with `(A B C (D))`, but not `(A B C D)`, or `(A B C (D) E)`. However, note that `(A -- (&) --)` will match with `(A B C (D) E)`. In other words, `--` can match any interior segment of a list.

If `(CDR PAT) = NIL`, i.e., *PAT* = `(--)`, then it matches any tail of a list. Therefore, `(A --)` matches `(A)`, `(A B C)` and `(A . B)`.

9. If `(CAR PAT)` is the atom `=`, *PAT* matches *X* if and only if `(CDR PAT)` is EQ to *X*.

This pattern is for use by programs that call the editor as a subroutine, since any non-atomic expression in a command *typed* in by you obviously cannot be EQ to already existing structure.

10. If `(CADR PAT)` is the atom `..` (two periods), *PAT* matches *X* if `(CAR PAT)` matches `(CAR X)` and `(CDDR PAT)` is contained in *X*, as described below.
11. Otherwise if *X* is a list, *PAT* matches *X* if `(CAR PAT)` matches `(CAR X)`, and `(CDR PAT)` matches `(CDR X)`.

When the editor is searching, the pattern matching routine is called to match with *elements* in the structure, unless the pattern begins with ... (three periods), in which case CDR of the pattern is matched against proper tails in the structure. Thus,

```
*P
      (A B C (B C))
      *F (B --)
      *P
      (B C)
      *O F (... B --)
      *P
      ... B C (B C)
```

Matching is also attempted with atomic tails (except for NIL). Thus,

```
*P
      (A (B . C))
      *F C
      *P
      ... . C)
```

Although the current expression is the atom C after the final command, it is printed as C) to alert you to the fact that C is a *tail*, not an element. Note that the pattern C will match with either instance of C in (A C (B . C)), whereas (... . C) will match only the second C. The pattern NIL will only match with NIL as an element, i.e., it will not match in (A B), even though CDDR of (A B) is NIL. However, (... . NIL) (or equivalently (...)) may be used to specify a NIL *tail*, e.g., (... . NIL) will match with CDR of the third subexpression of ((A . B) (C . D) (E)).

Search Algorithm

Searching begins with the current expression and proceeds in print order. Searching usually means find the next instance of this pattern, and consequently a match is not attempted that would leave the edit chain unchanged. At each step, the pattern is matched against the next element in the expression currently being searched, unless the pattern begins with ... (three periods) in which case it is matched against the next tail of the expression.

If the match is not successful, the search operation is recursive first in the CAR direction, and then in the CDR direction, i.e., if the element under examination is a list, the search descends into that list before attempting to match with other elements (or tails) at the same level. Note: A find command of the form (F PATTERN NIL) will only attempt matches at the top level of the current expression, i.e., it does not descend into elements, or ascend to higher expressions.

However, at no point is the total recursive depth of the search (sum of number of CARs and CDRs descended into) allowed to exceed the value of the variable MAXLEVEL. At that point, the search of that element or tail is abandoned, exactly as though the element or tail had been completely searched without finding a match, and the search continues with the element or tail for which the recursive depth is below MAXLEVEL. This feature is designed to enable you to search circular list structures (by setting MAXLEVEL small), as well as protecting him from accidentally encountering a circular list

INTERLISP-D REFERENCE MANUAL

structure in the course of normal editing. `MAXLEVEL` can also be set to `NIL`, which is equivalent to infinity. `MAXLEVEL` is initially set to 300.

If a successful match is not found in the current expression, the search automatically ascends to the next higher expression, and continues searching there on the next expression after the expression it just finished searching. If there is none, it ascends again, etc. This process continues until the entire edit chain has been searched, at which point the search fails, and an error is generated. If the search fails (or is aborted by Control-E), the edit chain is not changed (nor are any `CONSES` performed).

If the search is successful, i.e., an expression is found that the pattern matches, the edit chain is set to the value it would have had had you reached that expression via a sequence of integer commands.

If the expression that matched was a list, it will be the final link in the edit chain, i.e., the new current expression. If the expression that matched is not a list, e.g., is an atom, the current expression will be the tail beginning with that atom, unless the atom is a tail, e.g., `B` in `(A . B)`. In this case, the current expression will be `B`, but will print as `. . . . B)`. In other words, the search effectively does an `UP` (unless `UPFINDFLG = NIL` (initially `T`)). See "Form Oriented Editing" in this chapter.

Search Commands

All of the commands below set `LASTAIL` for use by `UP`, set `UNFIND` for use by `\` (below), and do not change the edit chain or perform any `CONSES` if they are unsuccessful or aborted.

`F PATTERN`

[Editor Command]

Actually two commands: the `F` informs the editor that the *next* command is to be interpreted as a pattern. This is the most common and useful form of the find command. If successful, the edit chain always changes, i.e., `F PATTERN` means find the next instance of `PATTERN`.

If `(MEMB PATTERN CURRENT-EXPRESSION)` is true, `F` does not proceed with a full recursive search. If the value of the `MEMB` is `NIL`, `F` invokes the search algorithm described above.

If the current expression is `(PROG NIL LP (COND (-- (GO LP1))) ... LP1 ...)`, then `F LP1` will find the `PROG` label, not the `LP1` inside of the `GO` expression, even though the latter appears first (in print order) in the current expression. Typing `1` (making the atom `PROG` be the current expression) followed by `F LP1` would find the first `LP1`.

`F PATTERNN`

[Editor Command]

Same as `F PATTERN`, i.e., Finds the Next instance of `PATTERN`, except that the `MEMB` check of `PATTERN` is not performed.

F PATTERN T

[Editor Command]

Similar to *F PATTERN*, except that it may succeed without changing the edit chain, and it does not perform the MEMB check. For example, if the current expression is (COND . . .) , *F COND* will look for the next COND, but (*F COND T*) will "stay here".

(*F PATTERN N*)

[Editor Command]

(*N* >= 1) Finds the *N*th place that *PATTERN* matches. Equivalent to (*F PATTERN T*) followed by (*F PATTERN N*) repeated *N*-1 times. Each time *PATTERN* successfully matches, *N* is decremented by 1, and the search continues, until *N* reaches 0. Note that *PATTERN* does not have to match with *N* identical expressions; it just has to match *N* times. Thus if the current expression is (FOO1 FOO2 FOO3), (*F FOO\$ 3*) will find FOO3.

If *PATTERN* does not match successfully *N* times, an error is generated and the edit chain is unchanged (even if *PATTERN* matched *N*-1 times).

(*F PATTERN*)

[Editor Command]

F PATTERN NIL

[Editor Command]

Similar to *F PATTERN*, except that it only matches with elements at the top level of the current expression, i.e., the search will not descend into the current expression, nor will it go outside of the current expression. May succeed without changing the edit chain.

For example, if the current expression is (PROG NIL (SETQ X (COND & &)) (COND & . . .) , the command *F COND* will find the COND inside the SETQ, whereas (*F (COND - -)*) will find the top level COND, i.e., the second one.

(*FS PATTERN . . . PATTERN*)

[Editor Command]

Equivalent to *F PATTERN* followed by *F PATTERN . . .* followed by *F PATTERN* , so that if *F PATTERN* fails, the edit chain is left at the place *PATTERN* matched.

(*F= EXPRESSION X*)

[Editor Command]

Equivalent to (*F (= . EXPRESSION) X*) , i.e., searches for a structure EQ to *EXPRESSION* (see above).

(*ORF PATTERN . . . PATTERN*)

[Editor Command]

Equivalent to (*F (*ANY*PATTERN . . . PATTERN) N*) , i.e., searches for an expression that is matched by either *PATTERN* , *PATTERN* , . . . or *PATTERN* (see above).

INTERLISP-D REFERENCE MANUAL

BF PATTERN

[Editor Command]

"Backwards Find". Searches in reverse print order, beginning with the expression immediately before the current expression (unless the current expression is the top level expression, in which case BF searches the entire expression, in reverse order).

BF uses the same pattern match routine as F, and MAXLEVEL and UPFINDFLG have the same effect, but the searching begins at the *end* of each list, and descends into each element before attempting to match that element. If unsuccessful, the search continues with the next previous element, etc., until the front of the list is reached, at which point BF ascends and backs up, etc.

For example, if the current expression is

```
(PROG NIL (SETQ X (SETQ Y (LIST Z))) (COND ((SETQ W --) --)) --),
```

the command F LIST followed by BF SETQ will leave the current expression as (SETQ Y (LIST Z)), as will F COND followed by BF SETQ.

BF PATTERN T

[Editor Command]

Similar to *BF PATTERN*, except that the search always includes the current expression, i.e., starts at the end of current expression and works backward, then ascends and backs up, etc.

Thus in the previous example, where F COND followed by BF SETQ found (SETQ Y (LIST Z)), F COND followed by (BF SETQ T) would find the (SETQ W --) expression.

(*BF PATTERN*)

[Editor Command]

BF PATTERN NIL

[Editor Command]

Same as *BF PATTERN*.

(*GO LABEL*)

[Editor Command]

Makes the current expression be the first thing after the PROG label *LABEL*, i.e. goes where an executed GO would go.

Location Specification

Many of the more sophisticated commands described later in this chapter use a more general method of specifying position called a "location specification." A location specification is a list of edit commands that are executed in the normal fashion with two exceptions. First, all commands not recognized by the editor are interpreted as though they had been preceded by F; normally such commands would cause errors. For example, the location specification (COND 2 3) specifies the 3rd element in the first clause of the next COND. Note that you could always write F COND followed by 2

and 3 for (COND 2 3) if you were not sure whether or not COND was the name of an atomic command.

Secondly, if an error occurs while evaluating one of the commands in the location specification, and the edit chain had been changed, i.e., was not the same as it was at the beginning of that execution of the location specification, the location operation will continue. In other words, the location operation keeps going unless it reaches a state where it detects that it is "looping", at which point it gives up. Thus, if (COND 2 3) is being located, and the first clause of the next COND contained only two elements, the execution of the command 3 would cause an error. The search would then continue by looking for the next COND. However, if a point were reached where there were no further CONDS, then the first command, COND, would cause the error; the edit chain would not have been changed, and so the entire location operation would fail, and cause an error.

The IF command (see above) in conjunction with the ## function (see below) provide a way of using arbitrary predicates applied to elements in the current expression. IF and ## will be described in detail later in the chapter, along with examples illustrating their use in location specifications.

Throughout this chapter, the meta-symbol @ is used to denote a location specification. Thus @ is a list of commands interpreted as described above. @ can also be atomic, in which case it is interpreted as (LIST @).

(LC . @) [Editor Command]

Provides a way of explicitly invoking the location operation, e.g., (LC COND 2 3) will perform the the search described above.

(LCL . @) [Editor Command]

Same as LC except the search is confined to the current expression, i.e., the edit chain is rebound during the search so that it looks as though the editor were called on just the current expression. For example, to find a COND containing a RETURN, one might use the location specification (COND (LCL RETURN) \) where the \ would reverse the effects of the LCL command, and make the final current expression be the COND.

(2ND . @) [Editor Command]

Same as (LC . @) followed by another (LC . @) except that if the first succeeds and second fails, no change is made to the edit chain.

(3ND . @) [Editor Command]

Similar to 2ND.

(← PATTERN) [Editor Command]

Ascends the edit chain looking for a link which matches PATTERN. In other words, it keeps doing 0's until it gets to a specified point. If PATTERN is atomic, it is matched with the first

INTERLISP-D REFERENCE MANUAL

element of each link, otherwise with the entire link. If no match is found, an error is generated, and the edit chain is unchanged.

If *PATTERN* is of the form (IF *EXPRESSION*), *EXPRESSION* is evaluated at each link, and if its value is NIL, or the evaluation causes an error, the ascent continues.

For example:

```
*PP
[PROG NIL
 (COND
  [(NULL (SETQ L (CDR L)))
   (COND
    (FLG (RETURN L])
    ([NULL (CDR (FMEMB (CAR L)
                      (CADR L])])
   *F CADR
  * (← COND)
  *P
  (COND (& &) (& &))
  *
```

Note that this command differs from BF in that it does not search *inside* of each link, it simply ascends. Thus in the above example, F CADR followed by BF COND would find (COND (FLG (RETURN L))), not the higher COND.

(BELOW *COM X*)

[Editor Command]

Ascends the edit chain looking for a link specified by *COM*, and stops *X* links below that (only links that are elements are counted, not tails). In other words BELOW keeps doing 0's until it gets to a specified point, and then backs off *X* 0's.

Note that *X* is evaluated, so one can type (BELOW *COM* (IPLUS *X Y*)).

(BELOW *COM*)

[Editor Command]

Same as (BELOW *COM* 1).

For example, (BELOW COND) will cause the COND *clause* containing the current expression to become the new current expression. Thus if the current expression is as shown above, F CADR followed by (BELOW COND) will make the new expression be ([NULL (CDR (FMEMB (CAR L) (CADR L]) (GO LP))), and is therefore equivalent to 0 0 0 0.

The BELOW command is useful for locating a substructure by specifying something it contains. For example, suppose you are editing a list of lists, and want to find a sublist that contains a FOO (at any depth). He simply executes F FOO (BELOW \).

(NEX COM)

[Editor Command]

Same as (BELOW COM) followed by NX.

For example, if you are deep inside of a SELECTQ clause, you can advance to the next clause with (NEX SELECTQ).

NEX

[Editor Command]

Same as (NEX ←).

The atomic form of NEX is useful if you will be performing repeated executions of (NEX COM). By simply MARKing (see the next section) the chain corresponding to COM, you can use NEX to step through the sublists.

(NTH COM)

[Editor Command]

Generalized NTH command. Effectively performs (LCL . COM), followed by (BELOW \), followed by UP.

If the search is unsuccessful, NTH generates an error and the edit chain is not changed.

Note that (NTH NUMBER) is just a special case of (NTH COM), and in fact, no special check is made for COM a number; both commands are executed identically.

In other words, NTH locates COM, using a search restricted to the current expression, and then backs up to the current level, where the new current expression is the tail whose first element contains, however deeply, the expression that was the terminus of the location operation. For example:

```
*P
(PROG (& &) LP (COND & &) (EDITCOM &) (SETQ UNFIND
UF) (RETURN L))
* (NTH UF)
*P
... (SETQ UNFIND UF) (RETURN L))
*
```

PATTERN . . @

[Editor Command]

For example, (COND . . RETURN). Finds a COND that contains a RETURN, at any depth. Equivalent to (but more efficient than) (F PATTERN N), (LCL . @) followed by (← PATTERN).

INTERLISP-D REFERENCE MANUAL

An infix command, "." is not a meta-symbol, it is the name of the command. @ is CDDR of the command. Note that (PATTERN . . @) can also be used directly as an edit pattern as described above, e.g. F (PATTERN . . @).

For example, if the current expression is

```
(PROG NIL [COND ((NULL L) (COND (FLG (RETURN L) --)),
```

then (COND . . RETURN) will make (COND (FLG (RETURN L))) be the current expression. Note that it is the innermost COND that is found, because this is the first COND encountered when ascending from the RETURN. In other words, (PATTERN . . @) is not *always* equivalent to (F PATTERN), followed by (LCL . . @) followed by \.

Note that @ is a location specification, not just a pattern. Thus (RETURN . . COND 2 3) can be used to find the RETURN which contains a COND whose first clause contains (at least) three elements. Note also that since @ permits any edit command, you can write commands of the form (COND . . (RETURN . . COND)), which will locate the first COND that contains a RETURN that contains a COND.

Commands That Save and Restore the Edit Chain

Several facilities are available for saving the current edit chain and later retrieving it: MARK, which marks the current chain for future reference, ←, which returns to the last mark without destroying it, and ←←, which returns to the last mark and also erases it.

MARK [Editor Command]

Adds the current edit chain to the front of the list MARKLIST.

← [Editor Command]

Makes the new edit chain be (CAR MARKLIST). Generates an error if MARKLIST is NIL, i.e., no MARKs have been performed, or all have been erased.

This is an atomic command; do not confuse it with the list command (← PATTERN).

←← [Editor Command]

Similar to ← but also erases the last MARK, i.e., performs (SETQ MARKLIST (CDR MARKLIST)).

If you have two chains marked, and wish to return to the first chain, you must perform ←←, which removes the second mark, and then ←. However, the second mark is then no longer

accessible. If you want to be able to return to either of two (or more) chains, you can use the following generalized MARK:

(MARK *SYMBOL*) [Editor Command]

Sets *SYMBOL* to the current edit chain,

(\ *SYMBOL*) [Editor Command]

Makes the current edit chain become the value of *SYMBOL*.

If you did not prepare in advance for returning to a particular edit chain, you may still be able to return to that chain with a single command by using \ or \P.

\ [Editor Command]

Makes the edit chain be the value of UNFIND. Generates an error if UNFIND = NIL.

UNFIND is set to the current edit chain by each command that makes a "big jump", i.e., a command that usually performs more than a single ascent or descent, namely ↑, ←, ←←, !NX, all commands that involve a search, e.g., F, LC, .., BELOW, et al and \ and \P themselves. One exception is that UNFIND is not reset when the current edit chain is the top level expression, since this could always be returned to via the ↑ command.

For example, if you type F COND, and then F CAR, \ would take you back to the COND. Another \ would take you back to the CAR, etc.

\P [Editor Command]

Restores the edit chain to its state as of the last print operation, i.e., P, ?, or PP. If the edit chain has not changed since the last printing, \P restores it to its state as of the printing before that one, i.e., two chains are always saved.

For example, if you type P followed by 3 2 1 P, \P returns to the first P, i.e., would be equivalent to 0 0 0. Another \P would then take you back to the second P, i.e., you could use \P to flip back and forth between the two edit chains.

If you had typed P followed by F COND, you could use *either* \ or \P to return to the P, i.e., the action of \ and \P are independent.

S *SYMBOL* @ [Editor Command]

Sets *SYMBOL* (using SETQ) to the current expression after performing (LC . @). The edit chain is not changed.

INTERLISP-D REFERENCE MANUAL

Thus (S FOO) will set FOO to the current expression, and (S FOO -1 1) will set FOO to the first element in the last element of the current expression.

Commands That Modify Structure

The basic structure modification commands in the editor are:

(N) (N >= 1) [Editor Command]

Deletes the corresponding element from the current expression.

(N E ... E) (N >= 1) [Editor Command]

Replaces the Nth element in the current expression with E ... E .

(-N E ... E) (N >= 1) [Editor Command]

Inserts E ... E before the Nth element in the current expression.

(N E ... E) [Editor Command]

Attaches E ... E at the end of the current expression.

As mentioned earlier: *all structure modification done by the editor is destructive, i.e., the editor uses RPLACA and RPLACD to physically change the structure it was given.* However, all structure modification is undoable, see UNDO .

All of the above commands generate errors if the current expression is not a list, or in the case of the first three commands, if the list contains fewer than N elements. In addition, the command (1), i.e., delete the first element, will cause an error if there is only one element, since deleting the first element must be done by replacing it with the second element, and then deleting the second element. Or, to look at it another way, deleting the first element when there is only one element would require changing a list to an atom (i.e., to NIL) which cannot be done. However, the command DELETE will work even if there is only one element in the current expression, since it will ascend to a point where it *can* do the deletion.

If the value of CHANGESARRAY is a hash array, the editor will mark all structures that are changed by doing (PUTHASH STRUCTURE FN CHANGESARRAY), where FN is the name of the function. The algorithm used for marking is as follows:

1. If the expression is inside of another expression already marked as being changed, do nothing.
2. If the change is an insertion of or replacement with a list, mark the list as changed.

3. If the change is an insertion of or replacement with an atom, or a deletion, mark the parent as changed.

CHANGESARRAY is primarily for use by PRETTYPRINT (Chapter 26). When the value of CHANGECHAR is non-NIL, PRETTYPRINT, when printing to a file or display terminal, prints CHANGECHAR in the right margin while printing an expression marked as having been changed. CHANGECHAR is initially |.

Implementation

Note: Since all commands that insert, replace, delete or attach structure use the same low level editor functions, the remarks made here are valid for all structure changing commands.

For all replacement, insertion, and attaching at the end of a list, unless the command was typed in directly to the editor, *copies* of the corresponding structure are used, because of the possibility that the exact same command, (i.e., same list structure) might be used again. Thus if a program constructs the command (1 (A B C)) e.g., via (LIST 1 FOO), and gives this command to the editor, the (A B C) used for the replacement will *not* be EQ to FOO. You can circumvent this by using the I command, which computes the structure to be used. In the above example, the form of the command would be (I 1 FOO), which would replace the first element with the value of FOO itself.

Note: Some editor commands take as arguments a list of edit commands, e.g., (LP F FOO (1 (CAR FOO))). In this case, the command (1 (CAR FOO)) is not considered to have been "typed in" even though the LP command itself may have been typed in. Similarly, commands originating from macros, or commands given to the editor as arguments to EDITF, EDITV, et al, e.g., EDITF(FOO F COND (N --)) are not considered typed in.

The rest of this section is included for applications wherein the editor is used to modify a data structure, and pointers into that data structure are stored elsewhere. In these cases, the actual mechanics of structure modification must be known in order to predict the effect that various commands may have on these outside pointers. For example, if the value of FOO is CDR of the current expression, what will the commands (2), (3), (2 X Y Z), (-2 X Y Z), etc. do to FOO?

Deletion of the first element in the current expression is performed by replacing it with the second element and deleting the second element by patching around it. Deletion of any other element is done by patching around it, i.e., the previous tail is altered. Thus if FOO is EQ to the current expression which is (A B C D), and FIE is CDR of FOO, after executing the command (1), FOO will be (B C D) (which is EQUAL but not EQ to FIE). However, under the same initial conditions, after executing (2) FIE will be unchanged, i.e., FIE will still be (B C D) even though the current expression and FOO are now (A C D).

A general solution of the problem isn't possible, as it would require being able to make two lists EQ to each other that were originally different. Thus if FIE is CDR of the current expression, and FUM is CDDR of the current expression, performing (2) would have to make FIE be EQ to FUM if all subsequent operations were to update both FIE and FUM correctly.

Both replacement and insertion are accomplished by smashing both CAR and CDR of the corresponding tail. Thus, if FOO were EQ to the current expression, (A B C D), after (1 X Y Z),

INTERLISP-D REFERENCE MANUAL

FOO would be (X Y Z B C D). Similarly, if FOO were EQ to the current expression, (A B C D), then after (-1 X Y Z), FOO would be (X Y Z A B C D).

The N command is accomplished by smashing the last CDR of the current expression a la NCONC. Thus if FOO were EQ to any tail of the current expression, after executing an N command, the corresponding expressions would also appear at the end of FOO.

In summary, the only situation in which an edit operation will *not* change an external pointer occurs when the external pointer is to a *proper tail* of the data structure, i.e., to CDR of some node in the structure, and the operation is deletion. If all external pointers are to *elements* of the structure, i.e., to CAR of some node, or if only insertions, replacements, or attachments are performed, the edit operation will *always* have the same effect on an external pointer as it does on the current expression.

The A, B, and : Commands

In the (N), (N E ... E), and (-N E ... E) commands, the sign of the integer is used to indicate the operation. As a result, there is no direct way to express insertion after a particular element (hence the necessity for a separate N command). Similarly, you cannot specify deletion or replacement of the Nth element from the end of a list without first converting N to the corresponding positive integer. Accordingly, we have:

(B E ... E) [Editor Command]

Inserts E ... E before the current expression. Equivalent to UP followed by (-1 E ... E).

For example, to insert FOO before the last element in the current expression, perform -1 and then (B FOO).

(A E ... E) [Editor Command]

Inserts E ... E after the current expression. Equivalent to UP followed by (-2 E ... E) or (N E ... E), whichever is appropriate.

(: E ... E) [Editor Command]

Replaces the current expression by E ... E. Equivalent to UP followed by (1 E ... E).

DELETE [Editor Command]
(:) [Editor Command]

Deletes the current expression.

DELETE first tries to delete the current expression by performing an UP and then a (1). This works in most cases. However, if after performing UP, the new current expression contains only one element, the command (1) will not work. Therefore, DELETE starts over and

performs a BK, followed by UP, followed by (2). For example, if the current expression is (COND ((MEMB X Y)) (T Y)), and you perform -1, and then DELETE, the BK-UP-(2) method is used, and the new current expression will be ... ((MEMB X Y)).

However, if the next higher expression contains only one element, BK will not work. So in this case, DELETE performs UP, followed by (: NIL), i.e., it *replaces* the higher expression by NIL. For example, if the current expression is (COND ((MEMB X Y)) (T Y)) and you perform F MEMB and then DELETE, the new current expression will be ... NIL (T Y)) and the original expression would now be (COND NIL (T Y)). The rationale behind this is that deleting (MEMB X Y) from ((MEMB X Y)) changes a list of one element to a list of no elements, i.e., () or NIL.

If the current expression is a tail, then B, A, :, and DELETE all work exactly the same as though the current expression were the first element in that tail. Thus if the current expression were ... (PRINT Y) (PRINT Z)), (B (PRINT X)) would insert (PRINT X) before (PRINT Y), leaving the current expression ... (PRINT X) (PRINT Y) (PRINT Z)).

The following forms of the A, B, and : commands incorporate a location specification:

(INSERT E ... E BEFORE . @) [Editor Command]

(@ is (CDR (MEMBER 'BEFORE COMMAND))) Similar to (LC .@) followed by (B E ... E).

Warning: If @ causes an error, the location process does *not* continue as described above. For example, if @ = (COND 3) and the next COND does not have a thirdelement, the search stops and the INSERT fails. You can always write (LC COND 3) if you intend the search to continue.

```
*P
(PROG (& X) **COMMENT** (SELECTQ ATM & NIL) (OR &
&) (PRIN1 & T)
(PRIN1 & T) (SETQ X &

*(INSERT LABEL BEFORE PRIN1)
*P
(PROG (& X) **COMMENT** (SELECTQ ATM & NIL) (OR &
&) LABEL
(PRIN1 & T) (
user typed Control-E

*
```

Current edit chain is not changed, but UNFIND is set to the edit chain after the B was performed, i.e., \ will make the edit chain be that chain where the insertion was performed.

INTERLISP-D REFERENCE MANUAL

(INSERT *E* ... *E* AFTER . @) [Editor Command]

Similar to INSERT BEFORE except uses A instead of B.

(INSERT *E* ... *E* FOR . @) [Editor Command]

Similar to INSERT BEFORE except uses : for B.

(REPLACE @ BY *E* ... *E*) [Editor Command]

(REPLACE @ WITH *E* ... *E*) [Editor Command]

Here @ is the *segment* of the command between REPLACE and WITH. Same as (INSERT *E* ... *E* FOR . @).

Example: (REPLACE COND -1 WITH (T (RETURN L)))

(CHANGE @ TO *E* ... *E*) [Editor Command]

Same as REPLACE WITH.

(DELETE . @) [Editor Command]

Does a (LC . @) followed by DELETE (see warning about INSERT above). The current edit chain is not changed, but UNFIND is set to the edit chain after the DELETE was performed.

Note: The edit chain will be changed if the current expression is no longer a part of the expression being edited, e.g., if the current expression is ... C) and you perform (DELETE 1), the tail, (C), will have been cut off. Similarly, if the current expression is (CDR Y) and you perform (REPLACE WITH (CAR X)).

Example: (DELETE -1), (DELETE COND 3)

Note: If @ is NIL (i.e., empty), the corresponding operation is performed on the current edit chain.

For example, (REPLACE WITH (CAR X)) is equivalent to (: (CAR X)). For added readability, HERE is also permitted, e.g., (INSERT (PRINT X) BEFORE HERE) will insert (PRINT X) before the current expression (but not change the edit chain).

Note: @ does not have to specify a location within the current expression, i.e., it is perfectly legal to ascend to INSERT, REPLACE, or DELETE

For example, (INSERT (RETURN) AFTER ^ PROG -1) will go to the top, find the first PROG, and insert a (RETURN) at its end, and not change the current edit chain.

The A, B, and : commands, commands, (and consequently INSERT, REPLACE, and CHANGE), all make special checks in *E* thru *E* for expressions of the form (## . COMS). In this case, the expression used for inserting or replacing is a *copy* of the current expression after executing COMS, a list of edit commands (the execution of COMS does not change the current edit chain). For example, (INSERT (## F COND -1 -1) AFTER 3) will make a copy of the last form in the last clause of the next COND, and insert it after the third element of the current expression. Note that this is not the same as (INSERT F COND -1 (## -1) AFTER 3), which inserts four elements after the third element, namely F, COND, -1, and a copy of the last element in the current expression.

Form Oriented Editing and the Role of UP

The UP that is performed before A, B, and : commands (and therefore in INSERT, CHANGE, REPLACE, and DELETE commands after the location portion of the operation has been performed) makes these operations form-oriented. For example, if you type F SETQ, and then DELETE, or simply (DELETE SETQ), you will delete the entire SETQ expression, whereas (DELETE X) if X is a variable, deletes just the variable X. In both cases, the operation is performed on the corresponding *form*, and in both cases is probably what you intended. Similarly, if you type (INSERT (RETURN Y) BEFORE SETQ), you mean before the SETQ expression, not before the atom SETQ. A consequent of this procedure is that a pattern of the form (SETQ Y --) can be viewed as simply an elaboration and further refinement of the pattern SETQ. Thus (INSERT (RETURN Y) BEFORE SETQ) and (INSERT (RETURN Y) BEFORE (SETQ Y --)) perform the same operation (assuming the next SETQ is of the form (SETQ Y --)) and, in fact, this is one of the motivations behind making the current expression after F SETQ, and F (SETQ Y --) be the same.

Note: There is some ambiguity in (INSERT *EXPR* AFTER *FUNCTIONNAME*), as you might mean make *EXPR* be the function's first argument. Similarly, you cannot write (REPLACE SETQ WITH SETQQ) meaning change the name of the function. You must in these cases write (INSERT *EXPR* AFTER *FUNCTIONNAME* 1), and (REPLACE SETQ 1 WITH SETQQ).

Occasionally, however, you may have a data structure in which no special significance or meaning is attached to the position of an atom in a list, as Interlisp attaches to atoms that appear as CAR of a list, versus those appearing elsewhere in a list. In general, you may not even *know* whether a particular atom is at the head of a list or not. Thus, when you write (INSERT *EXPR* BEFORE FOO), you mean before the atom FOO, whether or not it is CAR of a list. By setting the variable UPFINDFLG to NIL (initially T), you can suppress the implicit UP that follows searches for atoms, and thus achieve the desired effect. With UPFINDFLG = NIL, following F FOO, for example, the current expression will be the atom FOO. In this case, the A, B, and : operations will operate with respect to the atom FOO. If you intend the operation to refer to the list which FOO heads, use the pattern (FOO --) instead.

Extract and Embed

Extraction involves replacing the current expression with one of its subexpressions (from any depth).

INTERLISP-D REFERENCE MANUAL

(XTR . @)

[Editor Command]

Replaces the original current expression with the expression that is current after performing (LCL . @) (see warning about INSERT above). If the current expression after (LCL . @) is a *tail* of a higher expression, its first element is used.

If the extracted expression is a list, then after XTR has finished, the current expression will be that list. If the extracted expression is not a list, the new current expression will be a tail whose first element is that non-list.

For example, if the current expression is (COND ((NULL X) (PRINT Y))), (XTR PRINT), or (XTR 2 2) will replace the COND by the PRINT. The current expression after the XTR would be (PRINT Y).

If the current expression is (COND ((NULL X) Y) (T Z)), then (XTR Y) will replace the COND with Y, even though the current expression after performing (LCL Y) is ... Y). The current expression after the XTR would be ... Y followed by whatever followed the COND.

If the current expression *initially* is a tail, extraction works exactly the same as though the current expression were the first element in that tail. Thus if the current expression is ... (COND ((NULL X) (PRINT Y))) (RETURN Z)), then (XTR PRINT) will replace the COND by the PRINT, leaving (PRINT Y) as the current expression.

The extract command can also incorporate a location specification:

(EXTRACT @ FROM . @)

[Editor Command]

Performs (LC . @) and then (XTR . @) (see warning about INSERT). The current edit chain is not changed, but UNFIND is set to the edit chain after the XTR was performed.

Note: @ is the *segment* between EXTRACT and FROM.

For example: If the current expression is (PRINT (COND ((NULL X) Y) (T Z))) then following (EXTRACT Y FROM COND), the current expression will be (PRINT Y). (EXTRACT 2 -1 FROM COND), (EXTRACT Y FROM 2), and (EXTRACT 2 -1 FROM 2) will all produce the same result.

While extracting replaces the current expression by a subexpression, embedding replaces the current expression with one containing *it* as a subexpression.

(MBD E ... E)

[Editor Command]

MBD substitutes the current expression for all instances of the atom & in E ... E , and replaces the current expression with the result of that substitution. As with SUBST, a fresh copy is used for each substitution.

If & does not appear in $E \dots E$, the MBD is interpreted as $(\text{MBD } (E \dots E \text{ \&}))$.

MBD leaves the edit chain so that the larger expression is the new current expression.

Examples:

If the current expression is $(\text{PRINT } Y)$, $(\text{MBD } (\text{COND } ((\text{NULL } X) \text{ \&}) ((\text{NULL } (\text{CAR } Y)) \text{ \& } (\text{GO } \text{LP}))))$ would replace $(\text{PRINT } Y)$ with $(\text{COND } ((\text{NULL } X) (\text{PRINT } Y)) ((\text{NULL } (\text{CAR } Y)) (\text{PRINT } Y) (\text{GO } \text{LP})))$.

If the current expression is $(\text{RETURN } X)$, $(\text{MBD } (\text{PRINT } Y) (\text{AND } \text{FLG } \text{\&}))$ would replace it with the two expressions $(\text{PRINT } Y)$ and $(\text{AND } \text{FLG } (\text{RETURN } X))$, i.e., if the $(\text{RETURN } X)$ appeared in the cond clause $(\text{T } (\text{RETURN } X))$, after the MBD, the clause would be $(\text{T } (\text{PRINT } Y) (\text{AND } \text{FLG } (\text{RETURN } X)))$.

If the current expression is $(\text{PRINT } Y)$, then $(\text{MBD } \text{SETQ } X)$ will replace it with $(\text{SETQ } X (\text{PRINT } Y))$. If the current expression is $(\text{PRINT } Y)$, $(\text{MBD } \text{RETURN})$ will replace it with $(\text{RETURN } (\text{PRINT } Y))$.

If the current expression *initially* is a tail, embedding works exactly the same as though the current expression were the first element in that tail. Thus if the current expression were $\dots (\text{PRINT } Y) (\text{PRINT } Z))$, $(\text{MBD } \text{SETQ } X)$ would replace $(\text{PRINT } Y)$ with $(\text{SETQ } X (\text{PRINT } Y))$.

The embed command can also incorporate a location specification:

$(\text{EMBED } @ \text{ IN } . X)$ [Editor Command]

(@ is the segment between EMBED and IN.) Does $(\text{LC } . @)$ and then $(\text{MBD } . X)$ (see warning about INSERT). Edit chain is not changed, but UNFIND is set to the edit chain after the MBD was performed.

Examples: $(\text{EMBED } \text{PRINT } \text{IN } \text{SETQ } X)$, $(\text{EMBED } 3 \ 2 \ \text{IN } \text{RETURN})$, $(\text{EMBED } \text{COND } 3 \ 1 \ \text{IN } (\text{OR } \text{\&} (\text{NULL } X)))$.

WITH can be used for IN, and SURROUND can be used for EMBED, e.g., $(\text{SURROUND } \text{NUMBERP } \text{WITH } (\text{AND } \text{\&} (\text{MINUSP } X)))$.

EDITEMBEDTOKEN [Variable]

The special atom used in the MBD and EMBED commands is the value of this variable, initially &.

INTERLISP-D REFERENCE MANUAL

The MOVE Command

The MOVE command allows you to specify the expression to be moved, the place it is to be moved to, and the operation to be performed there, e.g., insert it before, insert it after, replace, etc.

(MOVE @ TO COM . @) [Editor Command]

(@ is the segment between MOVE and TO.) COM is BEFORE, AFTER, or the name of a list command, e.g., :, N, etc. Performs (LC . @) (see warning about INSERT), and obtains the current expression there (or its first element, if it is a tail), which we will call EXPR; MOVE then goes back to the original edit chain, performs (LC . @) followed by (COM EXPR) (setting an internal flag so EXPR is not copied), then goes back to @ and deletes EXPR. The edit chain is not changed. UNFIND is set to the edit chain after (COM EXPR) was performed.

If @ specifies a location *inside of the expression to be moved*, a message is printed and an error is generated, e.g., (MOVE 2 TO AFTER X), where X is contained inside of the second element.

For example, if the current expression is (A B C D), (MOVE 2 TO AFTER 4) will make the new current expression be (A C D B). Note that 4 was executed as of the original edit chain, and that the second element had not yet been removed.

As the following examples taken from actual editing will show, the MOVE command is an extremely versatile and powerful feature of the editor.

```
*?
(PROG ((L L)) (EDLOC (CDDR C)) (RETURN (CAR L)))
*(MOVE 3 TO : CAR)
*?
(PROG ((L L)) (RETURN (EDLOC (CDDR C))))
*
*P
... (SELECTQ OBJPR & &) (RETURN &) LP2 (COND & &))
*(MOVE 2 TO N 1)
*P
... (SELECTQ OBJPR & & &) LP2 (COND & &))

*
*P
(OR (EQ X LASTAIL) (NOT &) (AND & & &))
*(MOVE 4 TO AFTER (BELOW COND))
*P
(OR (EQ X LASTAIL) (NOT &))
*\ P
... (& &) (AND & & &) (T & &))
*

*P
((NULL X) **COMMENT** (COND & &))
*(-3 (GO NXT])
*(MOVE 4 TO N (← PROG))
*P
```

```

( (NULL X) **COMMENT** (GO NXT) )
*\ P
(PROG (&) **COMMENT** (COND & & &) (COND & & &) (COND & & &))
*(INSERT NXT BEFORE -1)
*\ P
(PROG (&) **COMMENT** (COND & & &) (COND & & &) NXT (COND & & &))

```

In the last example, you could have added the PROG label NXT and moved the COND in one operation by performing (MOVE 4 TO N (← PROG) (N NXT)). Similarly, in the next example, in the course of specifying @ , the location where the expression was to be moved to, you also perform a structure modification, via (N (T)), thus creating the structure that will receive the expression being moved.

```

*\ P
( (CDR &) **COMMENT** (SETQ CL &) (EDITSMASH CL & &))
*\ MOVE 4 TO N 0 (N (T)) -1]
*\ P
( (CDR &) **COMMENT** (SETQ CL &))
*\ P
*(T (EDITSMASH CL & &))
*

```

If @ is NIL, or (HERE), the current position specifies where the operation is to take place. In this case, UNFIND is set to where the expression that was moved was originally located, i.e., @ . For example:

```

*\ P
(TENEX)
*(MOVE ↑ F APPLY TO N HERE)
*\ P
(TENEX (APPLY & &))
*

*\ P
(PROG (& & & ATM IND VAL) (OR & &) **COMMENT** (OR & &))
(PRIN1 & T) (
PRIN1 & T) (SETQ IND user typed Control-E)

*(MOVE * TO BEFORE HERE)
*\ P
(PROG (& & & ATM IND VAL) (OR & &) (OR & &) (PRIN1 &

*\ P
(T (PRIN1 C-EXP T))
*(MOVE ↑ BF PRIN1 TO N HERE)
*\ P
(T (PRIN1 C-EXP T) (PRIN1 & T))
*

```

Finally, if @ is NIL, the MOVE command allows you to specify where the *current expression* is to be moved to. In this case, the edit chain is changed, and is the chain where the current expression was moved to; UNFIND is set to where it was.

```

*\ P

```


INTERLISP-D REFERENCE MANUAL

```
(SELECTQ OBJPR (&) (PROGN & &))
*(MOVE TO BEFORE LOOP)
*P
... (SELECTQ OBJPR & &) LOOP (FRPLACA DFPRP &) (FRPLACD DFPRP
&) (SELECTQ          user typed Control-E
*

```

Commands That Move Parentheses

The commands presented in this section permit modification of the list structure itself, as opposed to modifying components thereof. Their effect can be described as inserting or removing a single left or right parenthesis, or pair of left and right parentheses. Of course, there will always be the same number of left parentheses as right parentheses in any list structure, since the parentheses are just a notational guide to the structure provided by `PRINT`. Thus, no command can insert or remove just one parenthesis, but this is suggestive of what actually happens.

In all six commands, *N* and *M* are used to specify an element of a list, usually of the current expression. In practice, *N* and *M* are usually positive or negative integers with the obvious interpretation. However, all six commands use the generalized `NTH` command (`NTH COM`) to find their element(s), so that *M*th element means the first element of the tail found by performing (`NTH N`). In other words, if the current expression is (`LIST (CAR X) (SETQ Y (CONS W Z))`), then (`BI 2 CONS`), (`BI X -1`), and (`BI X Z`) all specify the exact same operation.

All six commands generate an error if the element is not found, i.e., the `NTH` fails. All are undoable.

(`BI N M`) [Editor Command]

"Both In". Inserts a left parentheses before the *M*th element and after the *M*th element in the current expression. Generates an error if the *M*th element is not contained in the *M*th tail, i.e., the *M*th element must be "to the right" of the *M*th element.

Example: If the current expression is (`A B (C D E) F G`), then (`BI 2 4`) will modify it to be (`A (B (C D E) F) G`).

(`BI N`) [Editor Command]

Same as (`BI N N`).

Example: If the current expression is (`A B (C D E) F G`), then (`BI -2`) will modify it to be (`A B (C D E) (F) G`).

(`BO N`) [Editor Command]

"Both Out". Removes both parentheses from the *M*th element. Generates an error if *M*th element is not a list.

Example: If the current expression is (A B (C D E) F G), then (BO D) will modify it to be (A B C D E F G).

(LI N)

[Editor Command]

"Left In". Inserts a left parenthesis before the *N*th element (and a matching right parenthesis at the end of the current expression), i.e. equivalent to (BI *N* - 1).

Example: if the current expression is (A B (C D E) F G), then (LI 2) will modify it to be (A (B (C D E) F G)).

(LO N)

[Editor Command]

"Left Out". Removes a left parenthesis from the *N*th element. *All elements following the Nth element are deleted.* Generates an error if *N*th element is not a list.

Example: If the current expression is (A B (C D E) F G), then (LO 3) will modify it to be (A B C D E).

(RI NM)

[Editor Command]

"Right In". Inserts a right parenthesis after the *M*th element of the *N*th element. The rest of the *N*th element is brought up to the level of the current expression.

Example: If the current expression is (A (B C D E) F G), (RI 2 2) will modify it to be (A (B C) D E F G). Another way of thinking about RI is to read it as "move the right parenthesis at the end of the *N*th element *in* to after its *M*th element."

(RO N)

[Editor Command]

"Right Out". Removes the right parenthesis from the *N*th element, moving it to the end of the current expression. All elements following the *N*th element are moved inside of the *N*th element. Generates an error if *N*th element is not a list.

Example: If the current expression is (A B (C D E) F G), (RO 3) will modify it to be (A B (C D E F G)). Another way of thinking about RO is to read it as "move the right parenthesis at the end of the *N*th element *out* to the end of the current expression."

TO and THRU

EXTRACT, EMBED, DELETE, REPLACE, and MOVE can be made to operate on several contiguous elements, i.e., a segment of a list, by using in their respective location specifications the TO or THRU command.

INTERLISP-D REFERENCE MANUAL

(@ THRU @)

[Editor Command]

Does a (LC . @), followed by an UP, and then a (BI 1 @), thereby grouping the segment into a single element, and finally does a 1, making the final current expression be that element.

For example, if the current expression is (A (B (C D) (E) (F G H) I) J K), following (C THRU G), the current expression will be ((C D) (E) (F G H)).

(@ TO @)

[Editor Command]

Same as THRU except the last element not included, i.e., after the BI, an (RI 1 -2) is performed.

If both @ and @ are numbers, and @ is greater than @, then @ counts from the beginning of the current expression, the same as @. In other words, if the current expression is (A B C D E F G), (3 THRU 5) means (C THRU E) not (C THRU G). In this case, the corresponding BI command is (BI 1 @ -@ +1).

THRU and TO are not very useful commands by themselves; they are intended to be used in conjunction with EXTRACT, EMBED, DELETE, REPLACE, and MOVE. After THRU and TO have operated, they set an internal editor flag informing the above commands that the element they are operating on is actually a segment, and that the extra pair of parentheses should be removed when the operation is complete. Thus:

```
*P
      (PROG (& & ATM IND VAL WORD) (PRIN1 & T) (PRIN1 & T) (SETQ
IND &)
      (SETQ VAL &) **COMMENT** (SETQQ      user typed Control-E
```

```
* (MOVE (3 THRU 4) TO BEFORE 7)
      *P
      (PROG (& & ATM IND VAL WORD) (SETQ IND &) (SETQ VAL &)
      (PRIN1 & T)
      (PRIN1 & T) **COMMENT**      user typed Control-E
```

```
*
      *P
      (* FAIL RETURN FROM EDITOR. USER SHOULD NOTE THE VALUES OF
SOURCEEXPR
AND CURRENTFORM. CURRENTFORM IS THE LAST FORM IN SOURCEEXPR
WHICH WILL
HAVE BEEN TRANSLATED, AND IT CAUSED THE ERROR.)
      *(DELETE (USER THRU CURR$))
      =CURRENTFORM.
      *P
      (* FAIL RETURN FROM EDITOR.CURRENTFORM IS      user typed Control-E
```

```

*
  *P
  ... LP (SELECTO & & & NIL) (SETQ Y &) OUT (SETQ FLG &)
  (RETURN Y))
  *(MOVE (1 TO OUT) TO N HERE]
  *P
  ... OUT (SETQ FLG &) (RETURN Y) LP (SELECTQ & & & NIL)
  (SETQ Y &))
*

*PP
  [PROG (RF TEMP1 TEMP2)
    (COND
      ((NOT (MEMB REMARG LISTING))
        (SETQ TEMP1 (ASSOC REMARG NAMEDREMARKS))
        **COMMENT**
        (SETQ TEMP2 (CADR TEMP1))
        (GO SKIP))
      (T **COMMENT**
        (SETQ TEMP1 REMARG)))
    (NCONC1 LISTING REMARG)
    (COND
      ((NOT (SETQ TEMP2 (SASSOC

```

*(EXTRACT (SETQ THRU CADR) FROM COND)
 *P
 (PROG (RF TEMP1 TEMP2) (SETQ TEMP1 &) **COMMENT** (SETQ
 TEMP2 &) (NCONC1 LISTING REMARG) (COND & & *user typed Control-*
 E
 *

TO and THRU can also be used directly with XTR, because XTR involves a location specification while A, B, :, and MBD do not. Thus in the previous example, if the current expression had been the COND, e.g., you had first performed F COND, you could have used (XTR (SETQ THRU CADR)) to perform the extraction.

(@ TO)	[Editor Command]
(@ THRU)	[Editor Command]

Both are the same as (@ THRU -1), i.e., from @ through the end of the list.

Examples:

```

*P
(VALUE (RPLACA DEPRP &) (RPLACD &) (RPLACA VARSWORD
&) (RETURN))
*(MOVE (2 TO) TO N (← PROG))
*(N (GO VAR))
*P
(VALUE (GO VAR))
*P
(T **COMMENT** (COND &) **COMMENT** (EDITSMASH CL &
&) (COND &))

```

INTERLISP-D REFERENCE MANUAL

```

* (-3 (GO REPLACE))
* (MOVE (COND TO) TO N ↑ PROG (N REPLACE))
*P
(T **COMMENT** (GO REPLACE))
*\ P
(PROG (&) **COMMENT** (COND & & &) (COND & & &))
DELETE (COND & &) REPLACE
(COND &) **COMMENT** (EDITSMAASH CL & &) (COND &))
*

*PP
[LAMBDA (CLAUSALA X)
  (PROG (A D)
    (SETQ A CLAUSALA)
    LP (COND
      ((NULL A)
        (RETURN)))
      (SERCH X A)
      (RUMARK (CDR A))
      (NOTICECL (CAR A))
      (SETQ A (CDR A))
      (GO LP]
* (EXTRACT (SERCH THRU NOT$) FROM PROG)
=NOTICECL
*P
(LAMBDA (CLAUSALA X) (SERCH X A) (RUMARK &) (NOTICECL
&))
* (EMBED (SERCH TO) IN (MAP CLAUSALA (FUNCTION (LAMBDA
(A) *])
*PP
[LAMBDA (CLAUSALA X)
  (MAP CLAUSALA
    (FUNCTION (LAMBDA (A)
      (SERCH X A)
      (RUMARK (CDR A))
      (NOTICECL (CAR A))
    )
  )
)

```

The R Command

(R X Y)

[Editor Command]

Replaces all instances of *X* by *Y* in the current expression, e.g., (R CAADR CADAR). Generates an error if there is not at least one instance.

The R command operates in conjunction with the search mechanism of the editor. The search proceeds as described in the Search Algorithm section above, and *X* can employ any of the patterns shown in the Commands That Search section above. Each time *X* matches an element of the structure, the element is replaced by (a copy of) *Y*; each time *X* matches a tail of the structure, the tail is replaced by (a copy of) *Y*.

For example, if the current expression is (A (B C) (B . C)),

(R C D) will change it to (A (B D) (B . D)),

(R (. . . . C) D) will change it to (A (B C) (B . D)),

(R C (D E)) will change it to (A (B (D E)) (B D E)), and

(R (. . . . NIL) D) will change it to (A (B C . D) (B . C) . D).

If *X* is an atom or string containing \$s (escapes), \$s appearing in *Y* stand for the characters matched by the corresponding \$ in *X*. For example, (R FOO\$ FIE\$) means for all atoms or strings that begin with FOO, replace the characters "FOO" by "FIE". Applied to the list (FOO FOO2 XFOO1), (R FOO\$ FIE\$) would produce (FIE FIE2 XFOO1), and (R \$FOO\$ \$FIE\$) would produce (FIE FIE2 XFIE1). Similarly, (R \$D\$ \$A\$) will change (LIST (CADR X) (CADDR Y)) to (LIST (CAAR X) (CAADR)). Note that CADDR was *not* changed to CAAAR, i.e., (R \$D\$ \$A\$) does not mean replace every D with A, but replace the first D in every atom or string by A. If you wanted to replace every D by A, you could perform (LP (R \$D\$ \$A\$)).

You will be informed of all such \$ replacements by a message of the form *X*->*Y*, e.g., CADR->CAAR.

If *X* matches a string, it will be replaced by a string. It does not matter whether *X* or *Y* themselves are strings, i.e. (R \$D\$ \$A\$), (R "\$D\$" \$A\$), (R \$D\$ "\$A\$"), and (R "\$D\$" "\$A\$") are equivalent. *X* will never match with a number, i.e., (R \$1 \$2) will not change 11 to 12.

The \$ (escape) feature can be used to delete or add characters, as well as replace them. For example, (R \$1 \$) will delete the terminating 1's from all literal atoms and strings. Similarly, if an \$ in *X* does not have a mate in *Y*, the characters matched by the \$ are effectively deleted. For example, (R \$/\$ \$) will change AND/OR to AND. There is no similar operation for changing AND/OR to OR, since the first \$ in *Y* always corresponds to the first \$ in *X*, the second \$ in *Y* to the second in *X*, etc. *Y* can also be a list containing \$s, e.g., (R \$1 (CAR \$)) will change FOO1 to (CAR FOO), FIE1 to (CAR FIE).

If *X* does not contain \$s, \$ appearing in *Y* refers to the *entire* expression matched by *X*, e.g., (R LONGATOM '\$) changes LONGATOM to 'LONGATOM, (R (SETQ X &) (PRINT \$)) changes every (SETQ X &) to (PRINT (SETQ X &)). If *X* is a pattern containing an \$ pattern somewhere *within* it, the characters matched by the \$s are not available, and for the purposes of replacement, the effect is the same as though *X* did not contain any \$s. For example, if you type (R (CAR F\$) (PRINT \$)), the second \$ will refer to the entire expression matched by (CAR F\$).

Since (R \$X\$ \$Y\$) is a frequently used operation for **R**Replacing **C**haracters, the following command is provided:

INTERLISP-D REFERENCE MANUAL

(RC *X Y*) [Editor Command]

Equivalent to (R *\$X\$ \$Y\$*)

R and RC change all instances of *X* to *Y*. The commands R1 and RC1 are available for changing just one, (i.e., the first) instance of *X* to *Y*.

(R1 *X Y*) [Editor Command]

Find the first instance of *X* and replace it by *Y*.

(RC1 *X Y*) [Editor Command]

Equivalent to (R1 *\$X\$ \$Y\$*).

In addition, while R and RC only operate within the current expression, R1 and RC1 will continue searching, a la the F command, until they find an instance of *x*, even if the search carries them beyond the current expression.

(SW *N M*) [Editor Command]

Switches the *N*th and *M*th elements of the current expression.

For example, if the current expression is (LIST (CONS (CAR X) (CAR Y)) (CONS (CDR X) (CDR Y))), (SW 2 3) will modify it to be (LIST (CONS (CDR X) (CDR Y)) (CONS (CAR X) (CAR Y))). The relative order of *N* and *M* is not important, i.e., (SW 3 2) and (SW 2 3) are equivalent.

SW uses the generalized NTH command (NTH *COM*) to find the *N*th and *M*th elements, a la the BI-BO commands.

Thus in the previous example, (SW CAR CDR) would produce the same result.

(SWAP @ @) [Editor Command]

Like SW except switches the expressions specified by @ and @ , not the corresponding elements of the current expression, i.e. @ and @ can be at different levels in current expression, or one or both be outside of current expression.

Thus, using the previous example, (SWAP CAR CDR) would result in (LIST (CONS (CDR X) (CAR Y)) (CONS (CAR X) (CDR Y))).

Commands That Print

PP [Editor Command]

Prettyprints the current expression.

P [Editor Command]

Prints the current expression as though PRINTLEVEL (Chapter 25) were set to 2.

(P M) [Editor Command]

Prints the M th element of the current expression as though PRINTLEVEL were set to 2.

(P 0) [Editor Command]

Same as P.

(P M N) [Editor Command]

Prints the M th element of the current expression as though PRINTLEVEL were set to N .

(P 0 N) [Editor Command]

Prints the current expression as though PRINTLEVEL were set to N .

? [Editor Command]

Same as (P 0 100).

Both (P M) and (P M N) use the generalized NTH command (NTH COM) to obtain the corresponding element, so that M does not have to be a number, e.g., (P COND 3) will work. PP causes all comments to be printed as **COMMENT** (see Chapter 26). P and ? print as **COMMENT** only those comments that are (top level) elements of the current expression. Lower expressions are not really seen by the editor; the printing command simply sets PRINTLEVEL and calls PRINT.

PP* [Editor Command]

Prettyprints current expression, *including* comments.

PP* is equivalent to PP except that it first resets **COMMENT**FLG** to NIL (see Chapter 26).

INTERLISP-D REFERENCE MANUAL

PPV [Editor Command]

Prettyprints the current expression as a variable, i.e., no special treatment for LAMBDA, COND, SETQ, etc., or for CLISP.

PPT [Editor Command]

Prettyprints the current expression, printing CLISP translations, if any.

?= [Editor Command]

Prints the argument names and corresponding values for the current expression. Analogous to the ?= break command (Chapter 14). For example,

```
*P
(STRPOS "A0???" X N (QUOTE ?) T)
*?=
X = "A0???"
Y = X
START = N
SKIP = (QUOTE ?)
ANCHOR = T
TAIL =
```

The command MAKE (see below) is an imperative form of ?=. It allows you to specify a change to the element of the current expression that corresponds to a particular argument name.

All printing functions print to the terminal, regardless of the primary output file. All use the readable T. No printing function ever changes the edit chain. All record the current edit chain for use by \P (above). All can be aborted with Control-E.

Commands for Leaving the Editor

OK [Editor Command]

Exits from the editor.

STOP [Editor Command]

Exits from the editor with an error. Mainly for use in conjunction with TTY: commands (see next section) that you want to abort.

Since all of the commands in the editor are errorset protected, you must exit from the editor via a command. STOP provides a way of distinguishing between a successful and unsuccessful (from your standpoint) editing session. For example, if you are executing (MOVE 3 TO AFTER COND TTY:), and you exitsfrom the lower editor with an OK, the

MOVE command will then complete its operation. If you want to abort the MOVE command, you must make the TTY: command generate an error. Do this by exiting from the lower editor with a STOP command. In this case, the higher editor's edit chain will not be changed by the TTY: command.

Actually, it is also possible to exit the editor by typing Control-D. STOP is preferred even if you are editing at the EVALQT level, as it will perform the necessary "wrapup" to insure that the changes made while editing will be undoable.

SAVE

[Editor Command]

Exits from the editor and saves the "state of the edit" on the property list of the function or variable being edited under the property EDIT-SAVE. If the editor is called again on the same structure, the editing is effectively "continued," i.e., the edit chain, mark list, value of UNFIND and UNDOLST are restored.

For example:

```
*P
(NULL X)
*F COND P
(COND (& &) (T &))
*SAVE
FOO
← .
.
.
←EDITF(FOO)
EDIT
*P
(COND (& &) (T &))
*\ P
(NULL X)
*
```

SAVE is necessary only if you are editing many different expressions; an exit from the editor via OK always saves the state of the edit of that call to the editor on the property list of the atom EDIT, under the property name LASTVALUE. OK also remprops EDIT-SAVE from the property list of the function or variable being edited.

Whenever the editor is entered, it checks to see if it is editing the same expression as the last one edited. In this case, it restores the mark list and UNDOLST, and sets UNFIND to be the edit chain as of the previous exit from the editor. For example:

```
←EDITF(FOO)
EDIT
*P
(LAMBDA (X) (PROG & & LP & & & &))
.
.
.
```

INTERLISP-D REFERENCE MANUAL

```
*P
(COND & &)
*OK
FOO
← .
.
.
←EDITF(FOO)
EDIT
*P
(LAMBDA (X) (PROG & & LP & & & &))
*\ P
(COND & &)
*
```

*any number of LISPX inputs
except for calls to the editor*

Furthermore, as a result of the history feature, if the editor is called on the same expression within a certain number of LISPX inputs (namely, the size of the history list, which can be changed with CHANGESLICE, Chapter 13) the state of the edit of that expression is restored, regardless of how many other expressions may have been edited in the meantime. For example:

```
←EDITF(FOO)
EDIT
*
.
.
.
*P
(COND (& &) (& &) (&) (T &))
*OK
FOO
.
.
.
←EDITF(FOO)
EDIT
*\ P
(COND (& &) (& &) (&) (T &))
*
```

*a small number of LISPX inputs,
including editing*

Thus you can always continue editing, including undoing changes from a previous editing session, if one of the following occurs:

1. No other expressions have been edited since that session (since saving takes place at *exit* time, intervening calls that were aborted via Control-D or exited via STOP will not affect the editor's memory).
2. That session was "sufficiently" recent.
3. It was ended with a SAVE command.

Nested Calls to Editor

TTY: [Editor Command]

Calls the editor recursively. You can then type in commands, and have them executed. The TTY: command is completed when you exit from the lower editor (see OK and STOP above).

The TTY: command is extremely useful. It enables you to set up a complex operation, and perform interactive attention-changing commands part way through it. For example, the command (MOVE 3 TO AFTER COND 3 P TTY:) allows you to interact, in effect, *within* the MOVE command. You can then verify for yourself that the correct location has been found, or complete the specification "by hand." In effect, TTY: says "I'll tell you what you should do when you get there."

The TTY: command operates by printing TTY: and then calling the editor. The initial edit chain in the lower editor is the one that existed in the higher editor at the time the TTY: command was entered. Until you exit from the lower editor, any attention changing commands you execute only affect the lower editor's edit chain. Of course, if you perform any structure modification commands while under a TTY: command, these will modify the structure in both editors, since it is the same structure. When the TTY: command finishes, the lower editor's edit chain becomes the edit chain of the higher editor.

EF [Editor Command]
EV [Editor Command]
EP [Editor Command]

Calls EDITF or EDITV or EDITP on CAR of current expression.

Manipulating the Characters of an Atom or String

RAISE [Editor Command]

An edit macro defined as UP followed by (I 1 (U-CASE (## 1))), i.e., it raises to uppercase the current expression, or if a tail, the first element of the current expression.

LOWER [Editor Command]

Similar to RAISE, except uses L-CASE.

CAP [Editor Command]

First does a RAISE, and then lowers all but the first character, i.e., the first character is left capitalized.

INTERLISP-D REFERENCE MANUAL

RAISE, LOWER, and CAP are all no-ops if the corresponding atom or string is already in that state.

(RAISE X) [Editor Command]

Equivalent to (I R (L-CASE X) X), i.e., changes every lowercase X to uppercase in the current expression.

(LOWER X) [Editor Command]

Similar to RAISE, except performs (I R X (L-CASE X)).

In both (RAISE X) and (LOWER X), X should be typed in uppercase.

REPACK [Editor Command]

Permits the "editing" of an atom or string.

REPACK operates by calling the editor recursively on UNPACK of the current expression, or if it is a list, on UNPACK of its first element. If the lower editor is exited successfully, i.e., via OK as opposed to STOP, the list of atoms is made into a single atom or string, which replaces the atom or string being "repacked." The new atom or string is always printed.

Example:

```
*P
... "THIS IS A LOGN STRING")
*REPACK
*EDIT
P
(T H I S % I S % A % L O G N % S T R I N G)
*(S W G N)
*OK
"THIS IS A LONG STRING"
*
```

This could also have been accomplished by (R \$GN\$ \$NG\$) or simply (RC GN NG).

(REPACK @) [Editor Command]

Does (LC . @) followed by REPACK, e.g. (REPACK THIS\$).

Manipulating Predicates and Conditional Expressions

JOINC

[Editor Command]

Used to join two neighboring CONDS together, e.g. `(COND CLAUSE CLAUSE)` followed by `(COND CLAUSE CLAUSE)` becomes `(COND CLAUSE CLAUSE CLAUSE CLAUSE)`. JOINC does an `(F COND T)` first so that you don't have to be at the first COND.

`(SPLITC X)`

[Editor Command]

Splits one COND into two. *X* specifies the last clause in the first COND, e.g. `(SPLITC 3)` splits `(COND CLAUSE CLAUSE CLAUSE CLAUSE)` into `(COND CLAUSE CLAUSE)` `(COND CLAUSE CLAUSE)`. Uses the generalized NTH command `(NTH COM)`, so that *X* does not have to be a number, e.g., you can say `(SPLITC RETURN)`, meaning split after the clause containing RETURN. SPLITC also does an `(F COND T)` first.

NEGATE

[Editor Command]

Negates the current expression, i.e. performs `(MBD NOT)`, except that is smart about simplifying. For example, if the current expression is: `(OR (NULL X) (LISTP X))`, NEGATE would change it to `(AND X (NLISTP X))`.

NEGATE is implemented via the function NEGATE (Chapter 3).

SWAPC

[Editor Command]

Takes a conditional expression of the form `(COND (A B) (T C))` and rearranges it to an equivalent `(COND ((NOT A) C) (T B))`, or `(COND (A B) (C D))` to `(COND ((NOT A) (COND (C D))) (T B))`.

SWAPC is smart about negations (uses NEGATE) and simplifying CONDS. It always produces an equivalent expression. It is useful for those cases where one wants to insert extra clauses or tests.

History Commands in the Editor

All of your inputs to the editor are stored on the history list EDITHISTORY (see Chapter 13, the editor's history list, and all of the programmer's assistant commands for manipulating the history list, e.g. REDO, USE, FIX, NAME, etc., are available for use on events on EDITHISTORY. In addition, the following four history commands are recognized specially by the editor. They always operate on the last, i.e. most recent, event.

INTERLISP-D REFERENCE MANUAL

DO COM

[Editor Command]

Allows you to supply the command name when it was omitted.

USE is useful when a command name is *incorrect*.

For example, suppose you want to perform `(-2 (SETQ X (LIST Y Z)))` but instead types just `(SETQ X (LIST Y Z))`. The editor will type `SETQ ?`, whereupon you can type `DO -2`. The effect is the same as though you had typed `FIX`, followed by `(LI 1)`, `(-1 -2)`, and OK, i.e., the command `(-2 (SETQ X (LIST Y Z)))` is executed. DO also works if the command is a line command.

!F

[Editor Command]

Same as DO F.

In the case of !F, the previous command is always treated as though it were a line command, e.g., if you type `(SETQ X &)` and then !F, the effect is the same as though you had typed `F (SETQ X &)`, not `(F (SETQ X &))`.

!E

[Editor Command]

Same as DO E.

!N

[Editor Command]

Same as DO N.

Miscellaneous Commands

NIL

[Editor Command]

Unless preceded by F or BF, is always a no-op. Thus extra right parentheses or square brackets at the ends of commands are ignored.

CL

[Editor Command]

Clispifies the current expression (see Chapter 21).

DW

[Editor Command]

Dwimifies the current expression (see Chapter 21).

IFY

[Editor Command]

If the current statement is a COND statement (Chapter 9), replaces it with an equivalent IF statement.

GET*

[Editor Command]

If the current expression is a comment pointer (see Chapter 26), reads in the full text of the comment, and replaces the current expression by it.

(* . X)

[Editor Command]

X is the text of a comment. * ascends the edit chain looking for a "safe" place to insert the comment, e.g., in a COND clause, after a PROG statement, etc., and inserts (* . X) *after* that point, if possible, otherwise before. For example, if the current expression is (FACT (SUB1 N)) in

```
[COND
  ((ZEROP N) 1)
  (T (ITIMES N (FACT (SUB1 N)
```

then (* CALL FACT RECURSIVELY) would insert (* CALL FACT RECURSIVELY) *before* the ITIMES expression. If inserted after the ITIMES, the comment would then be (incorrectly) returned as the value of the COND. However, if the COND was itself a PROG statement, and hence its value was not being used, the comment could be (and would be) inserted after the ITIMES expression.

* does not change the edit chain, but UNFIND is set to where the comment was actually inserted.

GETD

[Editor Command]

Essentially "expands" the current expression in line:

1. If (CAR of) the current expression is the name of a macro, expands the macro in line;
2. If a CLISP word, translates the current expression and replaces it with the translation;
3. If CAR is the name of a function for which the editor can obtain a symbolic definition, either in-core or from a file, substitutes the argument expressions for the corresponding argument names in the body of the definition and replaces the current expression with the result;
4. If CAR of the current expression is an open lambda, substitutes the arguments for the corresponding argument names in the body of the lambda, and then removes the lambda and argument list.

INTERLISP-D REFERENCE MANUAL

Warning: When expanding a function definition or open lambda expression, GETD does a simple substitution of the actual arguments for the formal arguments. Therefore, if any of the function arguments are used in other ways in the function definition (as functions, as record fields, etc.), they will simply be replaced with the actual arguments.

(MAKEFN (FN . ACTUALARGS) ARGLIST N1 N2) [Editor Command]

The inverse of GETD: makes the current expression into a function. *FN* is the function name, *ARGLIST* its arguments. The argument names are substituted for the corresponding argument values in *ACTUALARGS*, and the result becomes the body of the function definition for *FN*. The current expression is then replaced with (FN . ACTUALARGS).

If *N* and *N* are supplied, (N THRU N) is used rather than the current expression; if just *N* is supplied, (N THRU -1) is used.

If *ARGLIST* is omitted, MAKEFN will make up some arguments, using elements of *ACTUALARGS*, if they are literal atoms, otherwise arguments selected from (X Y Z A B C . . .), avoiding duplicate argument names.

Example: If the current expression is (COND ((CAR X) (PRINT Y T)) (T (HELP))), then (MAKEFN (FOO (CAR X) Y) (A B)) will define FOO as (LAMBDA (A B) (COND (A (PRINT B T)) (T (HELP)))) and then replace the current expression with (FOO (CAR X) Y).

(MAKE ARGNAME EXP) [Editor Command]

Makes the value of *ARGNAME* be *EXP* in the call which is the current expression, i.e. a ?= command following a MAKE will always print *ARGNAME* = *EXP*. For example:

```
*P
(JSYS)
*?=
JSYS [N;AC1,AC2,AC3,RESULTAC]
*(MAKE N 10)
*(MAKE RESULTAC 3)
*P
(JSYS 10 NIL NIL NIL 3)
```

Q [Editor Command]

Quotes the current expression, i.e. MBD QUOTE.

D [Editor Command]

Deletes the current expression, then prints new current expression, i.e. (:) I P.

Commands That Evaluate

E

[Editor Command]

Causes the editor to call the Interlisp executive LISPX giving it the next input as argument.

Example:

```
*E BREAK (FIE FUM)
(FIE FUM)
*E (FOO)

(FIE BROKEN)
:
```

E only works when typed in, e.g, (INSERT D BEFORE E) will treat E as a pattern, and search for E.

(E X)

[Editor Command]

Evaluates X, i.e., performs (EVAL X), and prints the result on the terminal.

(E X T)

[Editor Command]

Same as (E x) but does not print.

The (E X) and (E X T) commands are mainly intended for use by macros and subroutine calls to the editor; you would probably type in a form for evaluation using the more convenient format of the (atomic) E command.

(I C X ... X)

[Editor Command]

Executes the *editor command* (C Y ... Y) where Y = (EVAL X). If C is not an atom, C is evaluated also.

Examples:

(I 3 (GETD 'FOO)) will replace the third element of the current expression with the definition of FOO.

(I N FOO (CAR FIE)) will attach the value of FOO and CAR of the value of FIE to the end of the current expression.

(I F = FOO T) will search for an expression EQ to the value of FOO.

INTERLISP-D REFERENCE MANUAL

(I (COND ((NULL FLG) '-1) (T 1)) FOO), if FLG is NIL, inserts the value of FOO before the first element of the current expression, otherwise replaces the first element by the value of FOO.

The I command sets an internal flag to indicate to the structure modification commands *not* to copy expression(s) when inserting, replacing, or attaching.

EVAL [Editor Command]

Does an EVAL of the current expression.

EVAL, line-feed, and the GO command together effectively allows you to "single-step" a program through its symbolic definition.

GETVAL [Editor Command]

Replaces the current expression by the result of evaluating it.

(## COM COM ... COM) [NLambda NoSpread Function]

An nlambda, nospread function (not a command). Its value is what the current expression would be after executing the edit commands COM ... COM starting from the present edit chain. Generates an error if any of COM thru COM cause errors. The current edit chain is never changed.

Note: The A, B, :, INSERT, REPLACE, and CHANGE commands make special checks for ## forms in the expressions used for inserting or replacing, and use a copy of ## form instead (see the A,B, and : Commands section above). Thus, (INSERT (## 3 2) AFTER 1) is equivalent to (I INSERT (COPY (## 3 2)) 'AFTER 1).

Example: (I R 'X (## (CONS .. Z))) replaces all X's in the current expression by the first CONS containing a Z.

The I command is not very convenient for computing an *entire* edit command for execution, since it computes the command name and its arguments separately. Also, the I command cannot be used to compute an atomic command. The following two commands provide more general ways of computing commands.

(COMS X ... X) [Editor Command]

Each X is evaluated and its value is executed as a command.

For example, (COMS (COND (X (LIST 1 X)))) will replace the first element of the current expression with the value of X if non-NIL, otherwise do nothing. The editor command NIL is a no-op (see the Miscellaneous Commands section above).

(COMSQ COM ... COM) [Editor Command]

Executes COM ... COM .

COMSQ is mainly useful in conjunction with the COMS command. For example, suppose you want to compute an entire list of commands for evaluation, as opposed to computing each command one at a time as does the COMS command. You would then write (COMS (CONS 'COMSQ X)) where X computed the list of commands, e.g., (COMS (CONS 'COMSQ (GETP FOO 'COMMANDS))).

Commands That Test

(IF X) [Editor Command]

Generates an error *unless* the value of (EVAL X) is true. In other words, if (EVAL X) causes an error or (EVAL X) = NIL, IF will cause an error.

For some editor commands, the occurrence of an error has a well defined meaning, i.e., they use errors to branch on, as COND uses NIL and non-NIL. For example, an error condition in a location specification may simply mean "not this one, try the next." Thus the location specification (IPLUS (E (OR (NUMBERP (## 3)) (ERROR!)) T)) specifies the first IPLUS whose second argument is a number. The IF command, by equating NIL to error, provides a more natural way of accomplishing the same result. Thus, an equivalent location specification is (IPLUS (IF (NUMBERP (## 3))))).

The IF command can also be used to select between two alternate lists of commands for execution.

(IF X COMS COMS) [Editor Command]

If (EVAL X) is true, execute COMS ; if (EVAL X) causes an error or is equal to NIL, execute COMS .

Thus IF is equivalent to

```
(COMS (CONS 'COMSQ
  (COND
    ((CAR (NLSETQ (EVAL X)))
      COMS )
    (T COMS ))))
```

INTERLISP-D REFERENCE MANUAL

For example, the command (IF (READP T) NIL (P)) will print the current expression provided the input buffer is empty.

(IF *X COMS*) [Editor Command]

If (EVAL *X*) is true, execute *COMS* ; otherwise generate an error.

(LP *COMS* ... *COMS*) [Editor Command]

Repeatedly executes *COMS* ... *COMS* until an error occurs.

For example, (LP F PRINT (N T)) will attach a T at the end of every PRINT expression. (LP F PRINT (IF (## 3) NIL ((N T)))) will attach a T at the end of each print expression which does not already have a second argument. The form (## 3) will cause an error if the edit command 3 causes an error, thereby selecting ((N T)) as the list of commands to be executed. The IF could also be written as (IF (CDDR (##)) NIL ((N T))).

When an error occurs, LP prints *N OCCURRENCES* where *N* is the number of times the commands were successfully executed. The edit chain is left as of the last complete successful execution of *COMS* ... *COMS* .

(LPQ *COMS* ... *COMS*) [Editor Command]

Same as LP but does not print the message *N OCCURRENCES*.

In order to prevent non-terminating loops, both LP and LPQ terminate when the number of iterations reaches MAXLOOP, initially set to 30. MAXLOOP can be set to NIL, which is equivalent to setting it to infinity. Since the edit chain is left as of the last successful completion of the loop, you can simply continue the LP command with REDO (see Chapter 13).

(SHOW *X*) [Editor Command]

X is a list of patterns. SHOW does a LPQ printing all instances of the indicated expression(s), e.g. (SHOW FOO (SETQ FIE &)) will print all FOOs and all (SETQ FIE &)s. Generates an error if there aren't any instances of the expression(s).

(EXAM *X*) [Editor Command]

Like SHOW except calls the editor recursively (via the TTY: command, see above) on each instance of the indicated expression(s) so that you can examine and/or change them.

(*ORR COMS* . . . *COMS*)

[Editor Command]

ORR begins by executing *COMS* , a list of commands. If no error occurs, *ORR* is finished. Otherwise, *ORR* restores the edit chain to its original value, and continues by executing *COMS* , etc. If none of the command lists execute without errors, i.e., the *ORR* "drops off the end", *ORR* generates an error. Otherwise, the edit chain is left as of the completion of the first command list which executes without an error.

NIL as a command list is perfectly legal, and will always execute successfully. Thus, making the last "argument" to *ORR* be *NIL* will insure that the *ORR* never causes an error. Any other atom is treated as (*ATOM*) , i.e., the above example could be written as (*ORR NX !NX NIL*) .

For example, (*ORR (NX) (!NX) NIL*) will perform a *NX*, if possible, otherwise a *!NX*, if possible, otherwise do nothing. Similarly, *DELETE* could be written as (*ORR (UP (1)) (BK UP (2)) (UP (: NIL))*) .

Edit Macros

Many of the more sophisticated branching commands in the editor, such as *ORR*, *IF*, etc., are most often used in conjunction with edit macros. The macro feature permits you to define new commands and thereby expand the editor's repertoire, or redefine existing commands (to refer to the original definition of a built-in command when redefining it via a macro, use the *ORIGINAL* command, below).

Macros are defined by using the *M* command:

(*M C COMS* . . . *COMS*)

[Editor Command]

For *C* an atom, *M* defines *C* as an atomic command. If a macro is redefined, its new definition replaces its old. Executing *C* is then the same as executing the list of commands *COMS* . . . *COMS* .

For example, (*M BP BK UP P*) will define *BP* as an atomic command which does three things, a *BK*, and *UP*, and a *P*. Macros can use commands defined by macros as well as built in commands in their definitions. For example, suppose *Z* is defined by (*M Z -1 (IF (READP T) NIL (P))*) , i.e., *Z* does a *-1*, and then if nothing has been typed, a *P*. Now we can define *ZZ* by (*M ZZ -1 Z*) , and *ZZZ* by (*M ZZZ -1 -1 Z*) or (*M ZZZ -1 ZZ*) .

Macros can also define list commands, i.e., commands that take arguments.

INTERLISP-D REFERENCE MANUAL

(M (C) (ARG ... ARG) COMS ... COMS) [Editor Command]

C an atom. M defines *C* as a list command. Executing (*C* *E* ... *E*) is then performed by substituting *E* for *ARG*, ... *E* for *ARG* throughout *COMS* ... *COMS*, and then executing *COMS* ... *COMS*.

For example, we could define a more general BP by (M (BP) (N) (BK N) UP P). Thus, (BP 3) would perform (BK 3), followed by an UP, followed by a P.

A list command can be defined via a macro so as to take a fixed or indefinite number of "arguments", as with spread vs. nospread functions. The form given above specified a macro with a fixed number of arguments, as indicated by its argument list. If the "argument list" is *atomic*, the command takes an indefinite number of arguments.

(M (C) ARG COMS ... COMS) [Editor Command]

If *C*, *ARG* are both atoms, this defines *C* as a list command. Executing (*C* *E* ... *E*) is performed by substituting (*E* ... *E*), i.e., CDR of the command, for *ARG* throughout *COMS* ... *COMS*, and then executing *COMS* ... *COMS*.

For example, the command 2ND (see the Location Specification section above), could be defined as a macro by (M (2ND) X (ORR ((LC . X) (LC . X)))).

For all editor commands, "built in" commands as well as commands defined by macros as atomic commands and list definitions are *completely* independent. In other words, the existence of an atomic definition for *C* in *no* way affects the treatment of *C* when it appears as CAR of a list command, and the existence of a list definition for *C* in *no* way affects the treatment of *C* when it appears as an atom. In particular, *C* can be used as the name of either an atomic command, or a list command, or both. In the latter case, two entirely different definitions can be used.

Once *C* is defined as an atomic command via a macro definition, it will *not* be searched for when used in a location specification, unless it is preceded by an F. Thus (INSERT -- BEFORE BP) would not search for BP, but instead perform a BK, and UP, and a P, and then do the insertion. The corresponding also holds true for list commands.

Occasionally, you will want to employ the S command in a macro to save some temporary result. For example, the SW command could be defined as:

```
(M (SW) (N M)
  (NTH N)
  (S FOO 1)
  MARK
  0
  (NTH M)
  (S FIE 1)
  (I 1 FOO))
```

```
←←
(I 1 FIE))
```

Since this version of SW sets FOO and FIE, using SW may have undesirable side effects, especially when the editor was called from deep in a computation, we would have to be careful to make up unique names for dummy variables used in edit macros, which is bothersome. Furthermore, it would be impossible to define a command that called itself recursively while setting free variables. The BIND command solves both problems.

```
(BIND COMS ... COMS )
```

[Editor Command]

Binds three dummy variables #1, #2, #3, (initialized to NIL), and then executes the edit commands *COMS ... COMS*. BIND uses a PROG to make these bindings, so they are only in effect while the commands are being executed and BINDs can be used recursively; the variables #1, #2, and #3 will be rebound each time BIND is invoked.

Thus, we can write SW safely as:

```
(M (SW) (N M)
  (BIND (NTH N)
    (S #1 1)
    MARK
    0
    (NTH M)
    (S #2 1)
    (I 1 #1)
    ←← (I 1 #2)))
```

```
(ORIGINAL COMS ... COMS )
```

[Editor Command]

Executes *COMS ... COMS* without regard to macro definitions. Useful for redefining a built in command in terms of itself., i.e. effectively allows you to "advise" edit commands.

User macros are stored on a list USERMACROS. The file package command USERMACROS (Chapter 17) is available for dumping all or selected user macros.

Undo

Each command that causes structure modification automatically adds an entry to the front of UNDO_{LIST} that contains the information required to restore all pointers that were changed by that command.

```
UNDO
```

[Editor Command]

Undoes the last, i.e., most recent, structure modification command that has not yet been undone, and prints the name of that command, e.g., MBD undone. The edit chain is then

INTERLISP-D REFERENCE MANUAL

exactly what it was before the "undone" command had been performed. If there are no commands to undo, UNDO types *nothing saved*.

!UNDO

[Editor Command]

Undoes all modifications performed during this editing session, i.e. this call to the editor. As each command is undone, its name is printed a la UNDO. If there is nothing to be undone, !UNDO prints *nothing saved*.

Undoing an event containing an I, E, or S command will also undo the side effects of the evaluation(s), e.g., undoing (I 3 (/NCONC FOO FIE)) will not only restore the third element but also restore FOO. Similarly, undoing an S command will undo the set. See the discussion of UNDO in Chapter 13. (If the I command was typed directly to the editor, /NCONC would automatically be substituted for NCONC as described in Chapter 13.)

Since UNDO and !UNDO cause structure modification, they also add an entry to UNDOLST. However, UNDO and !UNDO entries are skipped by UNDO, e.g., if you perform an INSERT, and then an MBD, the first UNDO will undo the MBD, and the second will undo the INSERT. However, you can also specify precisely which commands you want undone by identifying the corresponding entry. In this case, you can undo an UNDO command, e.g., by typing UNDO UNDO, or undo a !UNDO command, or undo a command other than that most recently performed.

Whenever you *continue* an editing session, the undo information of the previous session is protected by inserting a special blip, called an undo-block, on the front of UNDOLST. This undo-block will terminate the operation of a !UNDO, thereby confining its effect to the current session, and will similarly prevent an UNDO command from operating on commands executed in the previous session.

Thus, if you enter the editor continuing a session, and immediately execute an UNDO or !UNDO, the editor will type BLOCKED instead of NOTHING SAVED. Similarly, if you execute several commands and then undo them all, another UNDO or !UNDO will also cause BLOCKED to be typed.

UNBLOCK

[Editor Command]

Removes an undo-block. If executed at a non-blocked state, i.e., if UNDO or !UNDO *could* operate, types NOT BLOCKED.

TEST

[Editor Command]

Adds an undo-block at the front of UNDOLST.

Note that TEST together with !UNDO provide a "tentative" mode for editing, i.e., you can perform a number of changes, and then undo all of them with a single !UNDO command.

(UNDO *EventSpec*)

[Editor Command]

EventSpec is an event specification (see Chapter 13). Undoes the indicated event on the history list. In this case, the event does not have to be in the current editing session, even if the previous session has not been unblocked as described above. However, you do have to be editing the same expression as was being edited in the indicated event.

If the expressions differ, the editor types the warning message "different expression," and does not undo the event. The editor enforces this to avoid your accidentally undoing a random command by giving the wrong event specification.

EDITDEFAULT

Whenever a command is not recognized, i.e., is not "built in" or defined as a macro, the editor calls an internal function, EDITDEFAULT, to determine what action to take. Since EDITDEFAULT is part of the edit block, you cannot advise or redefine it as a means of augmenting or extending the editor. However, you can accomplish this via EDITUSERFN. If the value of the variable EDITUSERFN is T, EDITDEFAULT calls the function EDITUSERFN giving it the command as an argument. If EDITUSERFN returns a non-NIL value, its value is interpreted as a single command and executed. Otherwise, the error correction procedure described below is performed.

If a location specification is being executed, an internal flag informs EDITDEFAULT to treat the command as though it had been preceded by an F.

If the command is a list, an attempt is made to perform spelling correction on the CAR of the command (unless DWIMFLG = NIL) using EDITCOMSL, a list of all list edit commands. If spelling correction is successful, the correct command name is RPLACAed into the command, and the editor continues by executing the command. In other words, if you type (LP F PRINT (MBBD AND (NULL FLG))), only one spelling correction will be necessary to change MBBD to MBD. If spelling correction is not successful, an error is generated.

Note: When a macro is defined via the M command, the command name is added to EDITCOMSA or EDITCOMSL, depending on whether it is an atomic or list command. The USERMACROS file package command is aware of this, and provides for restoring EDITCOMSA and EDITCOMSL.

If the command is atomic, the procedure followed is a little more elaborate.

1. If the command is one of the list commands, i.e., a member of EDITCOMSL, and there is additional input on the same terminal line, treat the entire line as a single list command. The line is read using READLINE (see Chapter 13), so the line can be terminated by a square bracket, or by a carriage return not preceded by a space. You may omit parentheses for any list command typed in at the top level (provided the command is not also an atomic command, e.g. NX, BK). For example,

INTERLISP-D REFERENCE MANUAL

```
*P
      (COND (& &) (T &))
      *XTR 3 2]
      *MOVE TO AFTER LP
      *
```

If the command is on the list EDITCOMSL but no additional input is on the terminal line, an error is generated. For example:

```
*P
      (COND (& &) (T &))
      *MOVE

      MOVE ?
      *
```

If the command is on EDITCOMSL, and *not* typed in directly, e.g., it appears as one of the commands in a LP command, the procedure is similar, with the rest of the command stream at that level being treated as "the terminal line", e.g. (LP F (COND (T &)) XTR 2 2).

If the command is being executed in location context, EDITDEFAULT does not get this far, e.g., (MOVE TO AFTER COND XTR 3) will search for XTR, *not* execute it. However, (MOVE TO AFTER COND (XTR 3)) will work.

2. If the command was typed in and the first character in the command is an 8, treat the 8 as a mistyped left parenthesis, and the rest of the line as the arguments to the command, e.g.,

```
*P
      (COND (& &) (T &))
      *8-2 (Y (RETURN Z)))
      =(-2
      *P
      (COND (Y &) (& &) (T &))
```

3. If the command was typed in, is the name of a function, and is followed by NIL or a list CAR of which is not an edit command, assume you forgot to type E and intend to apply the function to its arguments, type =E and the function name, and perform the indicated computation, e.g.

```
*BREAK(FOO)
      =E BREAK
      (FOO)
      *
```

4. If the last character in the command is P, and the first N-1 characters comprise a number, assume that you intended two commands, e.g.,

```
*P
      (COND (& &) (T &))
      *0P
      =0 P
```

```
(SETQ X (COND & &))
```

5. Attempt spelling correction using EDITCOMSA, and if successful, execute the corrected command.
6. If there is additional input on the same line, or command stream, spelling correct using EDITCOMSL as a spelling list, e.g.,

```
*MBBD SETQ X  
      =MBD  
      *
```

7. Otherwise, generate an error.

Time Stamps

Whenever a function is edited, and changes were made, the function is time-stamped (by EDITE), which consists of inserting a comment of the form (** USERS-INITIALS DATE*). *USERS-INITIALS* is the value of the variable INITIALS. After greeting (see Chapter 12), the function SETINITIALS is called. SETINITIALS searches INITIALSLST, a list of elements of the form (*USERNAME . INITIALS*) or (*USERNAME FIRSTNAME INITIALS*). If your name is found, INITIALS is set accordingly. If your username name is *not* found on INITIALSLST, INITIALS is set to the value of DEFAULTINITIALS, initially edited:. Thus, the default is to always time stamp. To suppress time stamping, you must either include an entry of the form (*USERNAME*) on INITIALSLST, or set DEFAULTINITIALS to NIL before greeting, i.e. in your user profile, or else, *after* greeting, explicitly set INITIALS to NIL.

If you want your functions to be time stamped with your initials when edited, include a file package command of the form (ADDVARS (INITIALSLST (*USERNAME . INITIALS*))) in your INIT.LISP file (see Chapter 12).

The following three functions may be of use for specialized applications with respect to time-stamping: (FIXEDITDATE *EXPR*) which, given a lambda expression, inserts or smashes a time-stamp comment; (EDITDATE? *COMMENT*) which returns T if *COMMENT* is a time stamp; and (EDITDATE *OLDDATE INITLS*) which returns a new time-stamp comment. If *OLDDATE* is a time-stamp comment, it will be reused.

Warning with Declarations

CAUTION: There is a feature of the BYTECOMPILER that is not supported by SEdit or the XCL compiler. It is possible to insert a comment at the beginning of your function that looks like

```
(* DECLARATIONS: --)
```

The tail, or -- section, of this comment is taken as a set of local record declarations which are then used by the compiler in that function just as if they had been declared globally. See the "Compiler" section in Chapter 3 of these Notes for additional behavior in XCL.

SEdit does not recognize such declarations. Thus, if the "Expand" command is used, the expansion will not be done with these record declarations in effect. The code that you see in SEdit will not be the same code compiled by the BYTECOMPILER.

[This page intentionally left blank]

17. FILE MANAGER

Warning: The subsystem within Medley used for managing collections of definitions (of functions, variables, etc.) is known as the "File Manager." This terminology is confusing, because the word "file" is also used in the more conventional sense as meaning a collection of data stored on some physical media. Unfortunately, it is not possible to change this terminology at this time, because many functions and variables (MAKEFILE, FILEPKGTYPES, etc.) incorporate the word "file" in their names.

Most implementations of Lisp treat symbolic files as unstructured text, much as they are treated in most conventional programming environments. Function definitions are edited with a character-oriented text editor, and then the changed definitions (or sometimes the entire file) is read or compiled to install those changes in the running memory image. Interlisp incorporates a different philosophy. A symbolic file is considered as a database of information about a group of data objects---function definitions, variable values, record declarations, etc. The text in a symbolic file is never edited directly. Definitions are edited only after their textual representations on files have been converted to data-structures that reside inside the Lisp address space. The programs for editing definitions inside Medley can therefore make use of the full set of data-manipulation capabilities that the environment already provides, and editing operations can be easily intermixed with the processes of evaluation and compilation.

Medley is thus a "resident" programming environment, and as such it provides facilities for moving definitions back and forth between memory and the external databases on symbolic files, and for doing the bookkeeping involved when definitions on many symbolic files with compiled counterparts are being manipulated. The file manager provides those capabilities. It shoulders the burden of keeping track of where things are and what things have changed so that you don't have to. The file manager also keeps track of which files have been modified and need to be updated and recompiled.

The file manager is integrated into many other system packages. For example, if only the compiled version of a file is loaded and you attempt to edit a function, the file manager will attempt to load the source of that function from the appropriate symbolic file. In many cases, if a datum is needed by some program, the file manager will automatically retrieve it from a file if it is not already in your working environment.

Some of the operations of the file manager are rather complex. For example, the same function may appear in several different files, or the symbolic or compiled files may be in different directories, etc. Therefore, this chapter does not document how the file manager works in each and every situation, but instead makes the deliberately vague statement that it does the "right" thing with respect to keeping track of what has been changed, and what file operations need to be performed in accordance with those changes.

For a simple illustration of what the file manager does, suppose that the symbolic file FOO contains the functions FOO1 and FOO2, and that the file BAR contains the functions BAR1 and BAR2. These two files could be loaded into the environment with the function LOAD:

```
← (LOAD 'FOO)
```

INTERLISP-D REFERENCE MANUAL

```
FILE CREATED 4-MAR-83 09:26:55
FOOCOMS
{DSK}FOO.;1

← (LOAD 'BAR)
FILE CREATED 4-MAR-83 09:27:24
BARCOMS
{DSK}BAR.;1
```

Now, suppose that we change the definition of FOO2 with the editor, and we define two new functions, NEW1 and NEW2. At that point, the file manager knows that the in-memory definition of FOO2 is no longer consistent with the definition in the file FOO, and that the new functions have been defined but have not yet been associated with a symbolic file and saved on permanent storage. The function FILES? summarizes this state of affairs and enters into an interactive dialog in which we can specify what files the new functions are to belong to.

```
← (FILES?)
FOO...to be dumped.
      plus the functions: NEW1,NEW2
want to say where the above go ? Yes
(functions)
NEW1  File name: BAR
NEW2  File name: ZAP
      new file ? Yes
NIL
```

The file manager knows that the file FOO has been changed, and needs to be dumped back to permanent storage. This can be done with MAKEFILE.

```
← (MAKEFILE 'FOO)
{DSK}FOO.;2
```

Since we added NEW1 to the old file BAR and established a new file ZAP to contain NEW2, both BAR and ZAP now also need to be dumped. This is confirmed by a second call to FILES?:

```
← (FILES?)
BAR, ZAP...to be dumped.
FOO...to be listed.
FOO...to be compiled
NIL
```

We are also informed that the new version we made of FOO needs to be listed (sent to a printer) and that the functions on the file must be compiled.

Rather than doing several MAKEFILEs to dump the files BAR and ZAP, we can simply call CLEANUP. Without any further user interaction, this will dump any files whose definitions have been modified. CLEANUP will also send any unlisted files to the printer and recompile any files which need to be recompiled. CLEANUP is a useful function to use at the end of a debugging session. It will call FILES? if any new objects have been defined, so you do not lose the opportunity to say explicitly where those belong. In effect, the function CLEANUP executes all the operations necessary to make the your permanent files consistent with the definitions in the current core-image.

```
← (CLEANUP)
```



```

FOO...compiling {DSK}FOO.;2
.
.
.
BAR...compiling {DSK}BAR.;2
.
.
.
ZAP...compiling {DSK}ZAP.;1
.
.
.

```

In addition to the definitions of functions, symbolic files in Interlisp can contain definitions of a variety of other types, e.g. variable values, property lists, record declarations, macro definitions, hash arrays, etc. In order to treat such a diverse assortment of data uniformly from the standpoint of file operations, the file manager uses the concept of a *typed definition*, of which a function definition is just one example. A typed definition associates with a name (usually a symbol), a definition of a given type (called the file manager type). Note that the same name may have several definitions of different types. For example, a symbol may have both a function definition and a variable definition. The file manager also keeps track of the files that a particular typed definition is stored on, so one can think of a typed definition as a relation between four elements: a name, a definition, a type, and a file.

Symbolic files on permanent storage devices are referred to by names that obey the naming conventions of those devices, usually including host, directory, and version fields. When such definition groups are noticed by the file manager, they are assigned simple *root names* and these are used by all file manager operations to refer to those groups of definitions. The root name for a group is computed from its full permanent storage name by applying the function `ROOTFILENAME`; this strips off the host, directory, version, etc., and returns just the simple name field of the file. For each file, the file manager also has a data structure that describes what definitions it contains. This is known as the commands of the file, or its "filecoms". By convention, the filecoms of a file whose root name is *X* is stored as the value of the symbol *XCOMS*. For example, the value of `FOOCOMS` is the filecoms for the file `FOO`. This variable can be directly manipulated, but the file manager contains facilities such as `FILES?` which make constructing and updating filecoms easier, and in some cases automatic. See the Functions for Manipulating File Command Lists section.

The file manager is able to maintain its databases of information because it is notified by various other routines in the system when events take place that may change that database. A file is "noticed" when it is loaded, or when a new file is stored (though there are ways to explicitly notice files without completely loading all their definitions). Once a file is noticed, the file manager takes it into account when modifying filecoms, dumping files, etc. The file manager also needs to know what typed definitions have been changed or what new definitions have been introduced, so it can determine which files need to be updated. This is done by "marking changes". All the system functions that perform file manager operations (`LOAD`, `TCOMPL`, `PRETTYDEF`, etc.), as well as those functions that define or change data, (`EDITF`, `EDITV`, `EDITP`, `DWIM` corrections to user functions) interact with the file manager. Also, *typed-in* assignment of variables or property values is noticed by the file manager. (Note that modifications to variable or property values during the execution of a function body are

INTERLISP-D REFERENCE MANUAL

not noticed.) In some cases the marking procedure can be subtle, e.g. if you edit a property list using EDITP, only those properties whose values are actually changed (or added) are marked.

All file manager operations can be disabled with FILEPKGFLG.

FILEPKGFLG

[Variable]

The file manager can be disabled by setting FILEPKGFLG to NIL. This will turn off noticing files and marking changes. FILEPKGFLG is initially T.

The rest of this chapter goes into further detail about the file manager. Functions for loading and storing symbolic files are presented first, followed by functions for adding and removing typed definitions from files, moving typed definitions from one file to another, determining which file a particular definition is stored in, and so on.

Loading Files

The functions below load information from symbolic files into the Interlisp environment. A symbolic file contains a sequence of Interlisp expressions that can be evaluated to establish specified typed definitions. The expressions on symbolic files are read using FILERDTBL as the read table.

The loading functions all have an argument *LDFLG*. *LDFLG* affects the operation of DEFINE, DEFINEQ, RPAQ, RPAQ?, and RPAQQ. While a source file is being loaded, DFNFLG (Chapter 10) is rebound to *LDFLG*. Thus, if *LDFLG* = NIL, and a function is redefined, a message is printed and the old definition saved. If *LDFLG* = T, the old definition is simply overwritten. If *LDFLG* = PROP, the functions are stored as "saved" definitions on the property lists under the property EXPR instead of being installed as the active definitions. If *LDFLG* = ALLPROP, not only function definitions but also variables set by RPAQQ, RPAQ, RPAQ? are stored on property lists (except when the variable has the value NOBIND, in which case they are set to the indicated value regardless of DFNFLG).

Another option is available for loading systems for others to use and who wish to suppress the saving of information used to aid in development and debugging. If *LDFLG* = SYSLOAD, LOAD will:

1. Rebind `DFNFLG` to `T`, so old definitions are simply overwritten
2. Rebind `LISPXHIST` to `NIL`, thereby making the `LOAD` not be undoable and eliminating the cost of saving undo information (Chapter 13)
3. Rebind `ADDSPELLFLG` to `NIL`, to suppress adding to spelling lists
4. Rebind `FILEPKGFLG` to `NIL`, to prevent the file from being "noticed" by the file manager
5. Rebind `BUILDMAPFLG` to `NIL`, to prevent a file map from being constructed
6. After the load has completed, set the `filecoms` variable and any `filevars` variables to `NOBIND`
7. Add the file name to `SYSFILES` rather than `FILELST`

A `filevars` variable is any variable appearing in a file manager command of the form (*FILECOM* * *VARIABLE*) (see the `FileVars` section). Therefore, if the `filecoms` includes (`FNS` * `FOOFNS`), `FOOFNS` is set to `NOBIND`. If you want the value of such a variable to be retained, even when the file is loaded with `LDFLG` = `SYSLOAD`, then you should replace the variable with an equivalent, *non-atomic* expression, such as (`FNS` * (`PROGN` `FOOFNS`)).

All functions that have `LDFLG` as an argument perform spelling correction using `LOADOPTIONS` as a spelling list when `LDFLG` is not a member of `LOADOPTIONS`. `LOADOPTIONS` is initially (`NIL` `T` `PROP` `ALLPROP` `SYSLOAD`).

(**LOAD** *FILE* *LDFLG* *PRINTFLG*) [Function]

Reads successive expressions from *FILE* (with `FILERDTBL` as read table) and evaluates each as it is read, until it reads either `NIL`, or the single atom `STOP`. Note that `LOAD` can be used to load both symbolic and compiled files. Returns *FILE* (full name).

If *PRINTFLG* = `T`, `LOAD` prints the value of each expression; otherwise it does not.

(**LOAD?** *FILE* *LDFLG* *PRINTFLG*) [Function]

Similar to `LOAD` except that it does not load *FILE* if it has already been loaded, in which case it returns `NIL`.

`LOAD?` loads *FILE* except when the *same* version of the file has been loaded (either from the same place, or from a copy of it from a different place). Specifically, `LOAD?` considers that *FILE* has already been loaded if the full name of *FILE* is on `LOADEDFILELST` (see the `Noticing Files` section) or the date stored on the `FILEDATES` property of the root file name of *FILE* is the same as the `FILECREATED` expression on *FILE*.

(**LOADFNS** *FNS* *FILE* *LDFLG* *VAR*) [Function]

Permits selective loading of definitions. *FNS* is a list of function names, a single function name, or `T`, meaning to load all of the functions on the file. *FILE* can be either a compiled

INTERLISP-D REFERENCE MANUAL

or symbolic file. If a compiled definition is loaded, so are all compiler-generated subfunctions. The interpretation of *LDFLG* is the same as for *LOAD*.

If *FILE* = *NIL*, *LOADFNS* will use *WHEREIS* (see the Storing Files section) to determine where the first function in *FNS* resides, and load from that file. Note that the file must previously have been "noticed". If *WHEREIS* returns *NIL*, and the *WHEREIS* library package has been loaded, *LOADFNS* will use the *WHEREIS* data base to find the file containing *FN*.

*VAR*s specifies which non-*DEFINEQ* expressions are to be loaded (i.e., evaluated). It is interpreted as follows:

T Means to load all non-*DEFINEQ* expressions.

NIL Means to load none of the non-*DEFINEQ* expressions.

VARs Means to evaluate all variable assignment expressions (beginning with *RPAQ*, *RPAQQ*, or *RPAQ?*, see the Functions Used Within Source Files section).

Any other symbol Means the same as specifying a list containing that atom.

A list If *VAR*s is a list that is not a valid function definition, each element in *VAR*s is "matched" against each non-*DEFINEQ* expression, and if any elements in *VAR*s "match" successfully, the expression is evaluated. "Matching" is defined as follows: If an element of *VAR*s is an atom, it matches an expression if it is *EQ* to either the *CAR* or the *CADR* of the expression. If an element of *VAR*s is a list, it is treated as an edit pattern (see Chapter 16), and matched with the entire expression (using *EDIT4E*, described in Chapter 16). For example, if *VAR*s was (*FOOCOMS DECLARE: (DEFLIST & (QUOTE MACRO))*), this would cause (*RPAQQ FOOCOMS ...*), all *DECLARE:s*, and all *DEFLIST*s which set up *MACROS* to be read and evaluated.

A function definition If *VAR*s is a list and a valid function definition ((*FNTYP VAR*s) is true), then *LOADFNS* will invoke that function on every non-*DEFINEQ* expression being considered, applying it to two arguments, the first and second elements in the expression. If the function returns *NIL*, the expression will be skipped; if it returns a non-*NIL* symbol (e.g., *T*), the expression will be evaluated; and if it returns a list, this list is evaluated instead of the expression. The file pointer is set to the very beginning of the expression before calling the *VAR*s function definition, so it may read the entire expression if necessary. If the function returns a symbol, the file pointer is reset and the expression is *READ* or *SKREAD*. However, the file pointer is not reset when the function returns a list, so the

function must leave it set immediately after the expression that it has presumably read.

LOADFNS returns a list of:

1. The names of the functions that were found
2. A list of those functions not found (if any) headed by the symbol NOT-FOUND:
3. All of the expressions that were evaluated
4. A list of those members of *VAR*s for which no corresponding expressions were found (if any), again headed by the symbol NOT-FOUND:

For example:

```
← (LOADFNS ' (FOO FIE FUM) FILE NIL ' (BAZ (DEFLIST &)) )
(FOO FIE (NOT-FOUND: FUM) (RPAQ BAZ ...) (NOT-FOUND:
(DEFLIST &)) )
```

(LOADVARS *VAR*s *FILE* *LDFLG*) [Function]

Same as (LOADFNS NIL *FILE* *LDFLG* *VAR*s).

(LOADFROM *FILE* *FNS* *LDFLG*) [Function]

Same as (LOADFNS *FNS* *FILE* *LDFLG* T).

Once the file manager has noticed a file, you can edit functions contained in the file without explicitly loading them. Similarly, those functions which have not been modified do not have to be loaded in order to write out an updated version of the file. Files are normally noticed (i.e., their contents become known to the file manager) when either the symbolic or compiled versions of the file are loaded. If the file is *not* going to be loaded completely, the preferred way to notice it is with LOADFROM. You can also load some functions at the same time by giving LOADFROM a second argument, but it is normally used simply to inform the file manager about the existence and contents of a particular file.

(LOADBLOCK *FN* *FILE* *LDFLG*) [Function]

Calls LOADFNS on those functions contained in the block declaration containing *FN* (see Chapter 18). LOADBLOCK is designed primarily for use with symbolic files, to load the EXPRS for a given block. It will not load a function which already has an in-core EXPR definition, and it will not load the block name, unless it is also one of the block functions.

(LOADCOMP *FILE* *LDFLG*) [Function]

Performs all operations on *FILE* associated with compilation, i.e. evaluates all expressions under a DECLARE: EVAL@COMPILE, and "notifies" the function and variable names by adding them to the lists NOFIXFNSLST and NOFIXVARSLST (see Chapter 21).

INTERLISP-D REFERENCE MANUAL

Thus, if building a system composed of many files with compilation information scattered among them, all that is required to compile one file is to `LOADCOMP` the others.

(**LOADCOMP?** *FILE* *LDLFG*) [Function]

Similar to `LOADCOMP`, except it does not load if file has already been loaded (with `LOADCOMP`), in which case its value is `NIL`.

`LOADCOMP?` will load the file even if it has been loaded with `LOAD`, `LOADFNS`, etc. The only time it will not load the file is if the file has already been loaded with `LOADCOMP`.

`FILESLOAD` provides an easy way for you to load a series of files, setting various options:

(**FILESLOAD** *FILE* ... *FILE*) [NLambda NoSpread Function]

Loads the files *FILE* ... *FILE* (all arguments unevaluated). If any of these arguments are lists, they specify certain loading options for all following files (unless changed by another list). Within these lists, the following commands are recognized:

FROM *DIR* Search the specified directories for the file. *DIR* can either be a single directory, or a list of directories to search in order. For example, (`FILESLOAD` (`FROM` {`ERIS`}<`LISPCORE`>`SOURCES`>) ...) will search the directory {`ERIS`}<`LISPCORE`>`SOURCES`> for the files. If this is not specified, the default is to search the contents of `DIRECTORIES` (see Chapter 24).

If `FROM` is followed by the key word `VALUEOF`, the following word is evaluated, and the value is used as the list of directories to search. For example, (`FILESLOAD` (`FROM` `VALUEOF` `FOO`) ...) will search the directory list that is the value of the variable `FOO`.

As a special case, if *DIR* is a symbol, and the symbol `DIRDIRECTORIES` is bound, the value of this variable is used as the directory search list. For example, since the variable `LISPUSERSDIRECTORIES` (see Chapter 24) is commonly used to contain a list of directories containing "library" packages, (`FILESLOAD` (`FROM` `LISPUSERS`) ...) can be used instead of (`FILESLOAD` (`FROM` `VALUEOF` `LISPUSERSDIRECTORIES`) ...)

If a `FILESLOAD` is read and evaluated while loading a file, and it doesn't contain a `FROM` expression, the default is to search the directory containing the `FILESLOAD` expression before the value of `DIRECTORIES`. `FILESLOAD` expressions can be dumped on files using the `FILES` file manager command.

SOURCE	Load the source version of the file rather than the compiled version.
COMPILED	Load the compiled version of the file. If COMPILED is specified, the compiled version will be loaded, if it is found. The source will not be loaded. If neither SOURCE or COMPILED is specified, the compiled version of the file will be loaded if it is found, otherwise the source will be loaded if it is found.
LOAD	Load the file by calling LOAD , if it has not already been loaded. This is the default unless LOADCOMP or LOADFROM is specified. If LOAD is specified, FILESLOAD considers that the file has already been loaded if the root name of the file has a non-NIL FILEDATES property. This is a somewhat different algorithm than LOAD? uses. In particular, FILESLOAD will not load a newer version of a file that has already been loaded.
LOADCOMP	Load the file with LOADCOMP? rather than LOAD . Automatically implies SOURCE .
LOADFROM	Load the file with LOADFROM rather than LOAD .
NIL, T, PROP ALLPROP	
SYSLOAD	The loading function is called with its <i>LDFLG</i> argument set to the specified token. <i>LDFLG</i> affects the operation of the loading functions by resetting DFNFLG (see Chapter 10) to <i>LDFLG</i> during the loading. If none of these tokens are specified, the value of the variable <i>LDFLG</i> is used if it is bound, otherwise NIL is used.
NOERROR	If NOERROR is specified, no error occurs when a file is not found.

Each list determines how all further files in the lists are loaded, unless changed by another list. The tokens above can be joined together in a single list. For example,

```
(FILESLOAD (LOADCOMP) NET (SYSLOAD FROM VALUEOF
NEWDIRECTORIES) CJSYS)
```

will call **LOADCOMP?** to load the file **NET** searching the value of **DIRECTORIES**, and then call **LOADCOMP?** to load the file **CJSYS** with *LDFLG* set to **SYSLOAD**, searching the directory list that is the value of the variable **NEWDIRECTORIES**.

INTERLISP-D REFERENCE MANUAL

FILESLOAD expressions can be dumped on files using the FILES file manager command.

Storing Files

(**MAKEFILE** *FILE* *OPTIONS* *REPRINTFNS* *SOURCEFILE*)

[Function]

Makes a new version of the file *FILE*, storing the information specified by *FILE*'s filecoms. Notices *FILE* if not previously noticed. Then, it adds *FILE* to NOTLISTEDFILES and NOTCOMPILEDFILES.

OPTIONS is a symbol or list of symbols which specify options. By specifying certain options, MAKEFILE can automatically compile or list *FILE*. Note that if *FILE* does not contain any function definitions, it is not compiled even when *OPTIONS* specifies C or RC. The options are spelling corrected using the list MAKEFILEOPTIONS. If spelling correction fails, MAKEFILE generates an error. The options are interpreted as follows:

C

RC After making *FILE*, MAKEFILE will compile *FILE* by calling TCOMPL (if C is specified) or RECOMPILE (if RC is specified). If there are any block declarations specified in the filecoms for *FILE*, BCOMPL or BRECOMPILE will be called instead.

If F, ST, STF, or S is the *next* item on *OPTIONS* following C or RC, it is given to the compiler as the answer to the compiler's question LISTING? (see Chapter 18). For example, (MAKEFILE 'FOO' (C F LIST)) will dump FOO, then TCOMPL or BCOMPL it specifying that functions are not to be redefined, and finally list the file.

LIST After making *FILE*, MAKEFILE calls LISTFILES to print a hardcopy listing of *FILE*.

CLISPIFY MAKEFILE calls PRETTYDEF with CLISPIFYPRETTYFLG = T (see Chapter 21). This causes CLISPIFY to be called on each function defined as an EXPR before it is prettyprinted.

Alternatively, if *FILE* has the property FILETYPE with value CLISP or a list containing CLISP, PRETTYDEF is called with CLISPIFYPRETTYFLG reset to CHANGES, which will cause CLISPIFY to be called on all functions marked as having been changed. If *FILE* has property FILETYPE with value CLISP, the compiler will DWIMIFY its functions before compiling them (see Chapter 18).

FAST MAKEFILE calls PRETTYDEF with PRETTYFLG = NIL (see Chapter 26). This causes data objects to be printed rather than prettyprinted, which is much faster.

REMAKE MAKEFILE "remakes" *FILE*: The prettyprinted definitions of functions that have not changed are copied from an earlier version of the symbolic file. Only those functions that have changed are prettyprinted.

NEW MAKEFILE does *not* remake *FILE*. If MAKEFILEREMAKEFLG = T (the initial setting), the default for all calls to MAKEFILE is to remake. The NEW option can be used to override this default.

REPRINTFNS and *SOURCEFILE* are used when remaking a file.

FILE is not added to NOTLISTEDFILES if *FILE* has on its property list the property FILETYPE with value DON'TLIST, or a list containing DON'TLIST. *FILE* is not added to NOTCOMPILEDFILES if *FILE* has on its property list the property FILETYPE with value DON'TCOMPILE, or a list containing DON'TCOMPILE. Also, if *FILE* does not contain any function definitions, it is not added to NOTCOMPILEDFILES, and it is not compiled even when *OPTIONS* specifies C or RC.

If a remake is *not* being performed, MAKEFILE checks the state of *FILE* to make sure that the entire source file was actually LOADED. If *FILE* was loaded as a compiled file, MAKEFILE prints the message CAN'T DUMP: ONLY THE COMPILED FILE HAS BEEN LOADED. Similarly, if only some of the symbolic definitions were loaded via LOADFNS or LOADFROM, MAKEFILE prints CAN'T DUMP: ONLY SOME OF ITS SYMBOLICS HAVE BEEN LOADED. In both cases, MAKEFILE will then ask you if it should dump anyway; if you decline, MAKEFILE does not call PRETTYDEF, but simply returns (*FILE*NOT DUMPED) as its value.

You can indicate that *FILE* must be block compiled together with other files as a unit by putting a list of those files on the property list of each file under the property FILEGROUP. If *FILE* has a FILEGROUP property, the compiler will not be called until all files on this property have been dumped that need to be.

MAKEFILE operates by rebinding PRETTYFLG, PRETTYTRANFLG, and CLISPIFYPRETTYFLG, evaluating each expression on MAKEFILEFORMS (under errorset protection), and then calling PRETTYDEF.

PrettyDEF calls PRETTYPRINT with its second argument PRETTYDEFLG = T, so whenever PRETTYPRINT (and hence MAKEFILE) start printing a new function, the name of that function is printed if more than 30 seconds (real time) have elapsed since the last time it printed the name of a function.

INTERLISP-D REFERENCE MANUAL

(**MAKEFILES** *OPTIONS FILES*) [Function]

Performs (MAKEFILE *FILE OPTIONS*) for each file on *FILES* that needs to be dumped. If *FILES* = NIL, FILELST is used. For example, (MAKEFILES 'LIST) will make and list all files that have been changed. In this case, if any typed definitions for any items have been defined or changed and they are *not* contained in one of the files on FILELST, MAKEFILES calls ADDTOFILES? to allow you to specify where these go. MAKEFILES returns a list of all files that are made.

(**CLEANUP** *FILE FILE ... FILE*) [NLambda NoSpread Function]

Dumps, lists, and recompiles (with RECOMPILE or BRECOMPILE) any of the specified files (unevaluated) requiring the corresponding operation. If no files are specified, FILELST is used. CLEANUP returns NIL.

CLEANUP uses the value of the variable CLEANUPOPTIONS as the *OPTIONS* argument to MAKEFILE. CLEANUPOPTIONS is initially (RC), to indicate that the files should be recompiled. If CLEANUPOPTIONS is set to (RC F), no listing will be performed, and no functions will be redefined as the result of compiling. Alternatively, if *FILE* is a list, it will be interpreted as the list of options regardless of the value of CLEANUPOPTIONS.

(**FILES?**) [Function]

Prints on the terminal the names of those files that have been modified but not dumped, dumped but not listed, dumped but not compiled, plus the names of any functions and other typed definitions (if any) that are not contained in any file. If there are any, FILES? then calls ADDTOFILES? to allow you to specify where these go.

(**ADDTOFILES?** —) [Function]

Called from MAKEFILES, CLEANUP, and FILES? when there are typed definitions that have been marked as changed which do not belong to any file. ADDTOFILES? lists the names of the changed items, and asks if you want to specify where these items should be put. If you answer N(o), ADDTOFILES? returns NIL without taking any action. If you answer], this is taken to be an answer to each question that would be asked, and all the changed items are marked as dummy items to be ignored. Otherwise, ADDTOFILES? prints the name of each changed item, and accepts one of the following responses:

A file name

A filevar If you give a file name or a variable whose value is a list (a filevar), the item is added to the corresponding file or list, using ADDTOFILE.

If your response is not the name of a file on FILELST or a variable whose value is a list, you will be asked whether it is a new file. If you say no, then ADDTOFILES? will check whether the item is the name of a list, i.e., whether its value is a list. If not, you will be asked whether it is a new list.

line-feed Same as your previous response.

space
carriage return Take no action.

-] The item is marked as a dummy item by adding it to NILCOMS. This tells the file manager simply to ignore this item.
- [The "definition" of the item in question is prettyprinted to the terminal, and then you are asked again about its disposition.
- (ADDTOFILES? prompts with "LISTNAME: (", you type in the name of a list, i.e. a variable whose value is a list, terminated by a). The item will then only be added to (under) a command in which the named list appears as a filevar. If none are found, a message is printed, and you are asked again. For example, you define a new function FOO3. When asked where it goes, you type (FOOFNS). If the command (FNS * FOOFNS) is found, FOO3 will be added to the value of FOOFNS. If instead you type (FOOCOMS), and the command (COMS * FOOCOMS) is found, then FOO3 will be added to a command for dumping functions that is contained in FOOCOMS.

If the named list is not also the name of a file, you can simply type it in without parenthesis as described above.
- @ ADDTOFILES? prompts with "Near: (", you type in the name of an object, and the item is then inserted in a command for dumping objects (of its type) that contains the indicated name. The item is inserted immediately after the indicated name.

(**LISTFILES** *FILE FILE ... FILE*)

[NLambda NoSpread Function]

Lists each of the specified files (unevaluated). If no files are given, NOTLISTEDFILES is used. Each file listed is removed from NOTLISTEDFILES if the listing is completed. For each file not found, LISTFILES prints the message *FILENAME* NOT FOUND and proceeds to the next file.

LISTFILES calls the function LISTFILES1 on each file to be listed. Normally, LISTFILES1 is defined to simply call SEND.FILE.TO.PRINTER (see Chapter 29), but you can advise or redefine LISTFILES1 for more specialized applications.

Any lists inside the argument list to LISTFILES are interpreted as property lists that set the various printing options, such as the printer, number of copies, banner page name, etc (see see Chapter 29). Later properties override earlier ones. For example,

(LISTFILES FOO (HOST JEDI) FUM (#COPIES 3) FIE)

INTERLISP-D REFERENCE MANUAL

will cause one copy of FOO to be printed on the default printer, and one copy of FUM and three copies of FIE to be printed on the printer JEDI.

(**COMPILEFILES** *FILE FILE ... FILE*) [NLambda NoSpread Function]

Executes the RC and C options of MAKEFILE for each of the specified files (unevaluated). If no files are given, NOTCOMPILEDFILES is used. Each file compiled is removed from NOTCOMPILEDFILES. If *FILE* is a list, it is interpreted as the *OPTIONS* argument to MAKEFILES. This feature can be used to supply an answer to the compiler's LISTING? question, e.g., (COMPILEFILES (STF)) will compile each file on NOTCOMPILEDFILES so that the functions are redefined without the EXPRs definitions being saved.

(**WHEREIS** *NAME TYPE FILES FN*) [Function]

TYPE is a file manager type. WHEREIS sweeps through all the files on the list *FILES* and returns a list of all files containing *NAME* as a *TYPE*. WHEREIS knows about and expands all file manager commands and file manager macros. *TYPE* = NIL defaults to FNS (to retrieve function definitions). If *FILES* is not a list, the value of FILELST is used.

If *FN* is given, it should be a function (with arguments *NAME*, *FILE*, and *TYPE*) which is applied for every file in *FILES* that contains *NAME* as a *TYPE*. In this case, WHEREIS returns NIL.

If the WHEREIS library package has been loaded, WHEREIS is redefined so that *FILES* = T means to use the whereis package data base, so WHEREIS will find *NAME* even if the file has not been loaded or noticed. *FILES* = NIL always means use FILELST.

Remaking a Symbolic File

Most of the time that a symbolic file is written using MAKEFILE, only a few of the functions that it contains have been changed since the last time the file was written. Rather than prettprinting all of the functions, it is often considerably faster to "remake" the file, copying the prettprinted definitions of unchanged functions from an earlier version of the symbolic file, and only prettyprinting those functions that have been changed.

MAKEFILE will remake the symbolic file if the REMAKE option is specified. If the NEW option is given, the file is not remade, and all of the functions are prettprinted. The default action is specified by the value of MAKEFILEREMAKEFLG: if T (its initial value), MAKEFILE will remake files unless the NEW option is given; if NIL, MAKEFILE will not remake unless the REMAKE option is given.

Note: If the file has never been loaded or dumped, for example if the filecoms were simply set up in memory, then MAKEFILE will never attempt to remake the file, regardless of the setting of MAKEFILEREMAKEFLG, or whether the REMAKE option was specified.

When MAKEFILE is remaking a symbolic file, you can explicitly indicate the functions which are to be prettyprinted and the file to be used for copying the rest of the function definitions from via the *REPRINTFNS* and *SOURCEFILE* arguments to MAKEFILE. Normally, both of these arguments are defaulted to NIL. In this case, *REPRINTFNS* will be set to those functions that have been changed since the last version of the file was written. For *SOURCEFILE*, MAKEFILE obtains the full name of the most recent version of the file (that it knows about) from the FILEDATES property of the file, and checks to make sure that the file still exists and has the same file date as that stored on the FILEDATES property. If it does, MAKEFILE uses that file as *SOURCEFILE*. This procedure permits you to LOAD or LOADFROM a file in a different directory, and still be able to remake the file with MAKEFILE. In the case where the most recent version of the file cannot be found, MAKEFILE will attempt to remake using the *original* version of the file (i.e., the one first loaded), specifying as *REPRINTFNS* the union of all changes that have been made since the file was first loaded, which is obtained from the FILECHANGES property of the file. If both of these fail, MAKEFILE prints the message "CAN'T FIND EITHER THE PREVIOUS VERSION OR THE ORIGINAL VERSION OF FILE, SO IT WILL HAVE TO BE WRITTEN ANEW", and does not remake the file, i.e. will prettyprint all of the functions.

When a remake is specified, MAKEFILE also checks to see how the file was originally loaded. If the file was originally loaded as a compiled file, MAKEFILE will call LOADVARS to obtain those DECLARE: expressions that are contained on the symbolic file, but not the compiled file, and hence have not been loaded. If the file was loaded by LOADFNS (but not LOADFROM), then LOADVARS is called to obtain any non-DEFINEQ expressions. Before calling LOADVARS to re-load definitions, MAKEFILE asks you, e.g. "Only the compiled version of FOO was loaded, do you want to LOADVARS the (DECLARE: .. DONTCOPY ..) expressions from {DSK}<MYDIR>FOO.;3?". You can respond Yes to execute the LOADVARS and continue the MAKEFILE, No to proceed with the MAKEFILE without performing the LOADVARS, or Abort to abort the MAKEFILE. You may wish to skip the LOADVARS if you had circumvented the file manager in some way, and loading the old definitions would overwrite new ones.

Remaking a symbolic file is considerably faster if the earlier version has a *file map* indicating where the function definitions are located (see the File Maps section), but it does not depend on this information.

Loading Files in a Distributed Environment

Each Interlisp source and compiled code file contains the full filename of the file, including the host and directory names, in a FILECREATED expression at the beginning of the file. The compiled code file also contains the full file name of the source file it was created from. In earlier versions of Interlisp, the file manager used this information to locate the appropriate source file when "remaking" or recompiling a file.

This turned out to be a bad feature in distributed environments, where users frequently move files from one place to another, or where files are stored on removable media. For example, suppose you MAKEFILE to a floppy, and then copy the file to a file server. If you loaded and edited the file from a file server, and tried to do MAKEFILE, it would try to locate the source file on the floppy, which is probably no longer loaded.

INTERLISP-D REFERENCE MANUAL

Currently, the file manager searches for sources file on the connected directory, and on the directory search path (on the variable `DIRECTORIES`). If it is not found, the host/directory information from the `FILECREATED` expression be used.

Warning: One situation where the new algorithm does the wrong thing is if you explicitly `LOADFROM` a file that is not on your directory search path. Future `MAKEFILES` and `CLEANUPS` will search the connected directory and `DIRECTORIES` to find the source file, rather than using the file that the `LOADFROM` was done from. Even if the correct file is on the directory search path, you could still create a bad file if there is another version of the file in an earlier directory on the search path. In general, you should either explicitly specify the `SOURCEFILE` argument to `MAKEFILE` to tell it where to get the old source, or connect to the directory where the correct source file is.

Marking Changes

The file manager needs to know what typed definitions have been changed, so it can determine which files need to be updated. This is done by "marking changes". All the system functions that perform file manager operations (`LOAD`, `TCOMPL`, `PRETTYDEF`, etc.), as well as those functions that define or change data, (`EDITF`, `EDITV`, `EDITP`, `DWIM` corrections to user functions) interact with the file manager by marking changes. Also, *typed-in* assignment of variables or property values is noticed by the file manager. (If a program modifies a variable or property value, this is not noticed.) In some cases the marking procedure can be subtle, e.g. if you edit a property list using `EDITP`, only those properties whose values are actually changed (or added) are marked.

The various system functions which create or modify objects call `MARKASCHANGED` to mark the object as changed. For example, when a function is defined via `DEFINE` or `DEFINEQ`, or modified via `EDITF`, or a `DWIM` correction, the function is marked as being a changed object of type `FNS`. Similarly, whenever a new record is declared, or an existing record redeclared or edited, it is marked as being a changed object of type `RECORDS`, and so on for all of the other file manager types.

You can also call `MARKASCHANGED` directly to mark objects of a particular file manager type as changed:

(MARKASCHANGED NAME TYPE REASON) [Function]

Marks *NAME* of type *TYPE* as being changed. `MARKASCHANGED` returns *NAME*. `MARKASCHANGED` is undoable.

REASON is a symbol that indicated how *NAME* was changed. `MARKASCHANGED` recognizes the following values for *REASON*:

DEFINED Used to indicate the creation of *NAME*, e.g. from `DEFINEQ` (Chapter 10).

CHANGED Used to indicate a change to *NAME*, e.g. from the editor.

DELETED Used to indicate the deletion of *NAME*, e.g. by `DELDEF`.

CLISP Used to indicate the modification of *NAME* by CLISP translation.

For backwards compatibility, `MARKASCHANGED` also accepts a *REASON* of `T` (`=DEFINED`) and `NIL` (`=CHANGED`). New programs should avoid using these values.

The variable `MARKASCHANGEDFNS` is a list of functions that `MARKASCHANGED` calls (with arguments *NAME*, *TYPE*, and *REASON*). Functions can be added to this list to "advise" `MARKASCHANGED` to do additional work for all types of objects. The `WHENCHANGED` file manager type property (see the Defining New File Manager Types section) can be used to specify additional actions when `MARKASCHANGED` gets called on specific types of objects.

(**UNMARKASCHANGED** *NAME TYPE*)

[Function]

Unmarks *NAME* of type *TYPE* as being changed. Returns *NAME* if *NAME* was marked as changed and is now unmarked, `NIL` otherwise. `UNMARKASCHANGED` is undoable.

(**FILEPKGCHANGES** *TYPE LST*)

[NoSpread Function]

If *LST* is not specified (as opposed to being `NIL`), returns a list of those objects of type *TYPE* that have been marked as changed but not yet associated with their corresponding files (see the File Manager Types section). If *LST* is specified, `FILEPKGCHANGES` sets the corresponding list. (`FILEPKGCHANGES`) returns a list of *all* objects marked as changed as a list of elements of the form (*TYPENAME* . *CHANGEDOBJECTS*).

Some properties (e.g. `EXPR`, `ADVICE`, `MACRO`, `I.S.OPR`, etc.) are used to implement other file manager types. For example, if you change the value of the property `I.S.OPR`, you are really changing an object of type `I.S.OPR`. The effect is the same as though you had redefined the `i.s.opr` via a direct call to the function `I.S.OPR`. If a property whose value has been changed or added does not correspond to a specific file manager type, then it is marked as a changed object of type `PROPS` whose *name* is (*VARIABLENAME* *PROPNAME*) (except if the property name has a property `PROPTYPE` with value `IGNORE`).

Similarly, if you change a variable which implements the file manager type `ALISTS` (as indicated by the appearance of the property `VARTYPE` with value `ALIST` on the variable's property list), only those entries that are actually changed are marked as being changed objects of type `ALISTS`. The "name" of the object will be (*VARIABLENAME* *KEY*) where *KEY* is `CAR` of the entry on the alist that is being marked. If the variable corresponds to a specific file manager type other than `ALISTS`, e.g., `USERMACROS`, `LISPMACROS`, etc., then an object of that type is marked. In this case, the name of the changed object will be `CAR` of the corresponding entry on the alist. For example, if you edit `LISPMACROS` and change a definition for `PL`, then the object `PL` of type `LISPMACROS` is marked as being changed.

INTERLISP-D REFERENCE MANUAL

Noticing Files

Already existing files are "noticed" by `LOAD` or `LOADFROM` (or by `LOADFNS` or `LOADVARS` when the `VARS` argument is `T`). New files are noticed when they are constructed by `MAKEFILE`, or when definitions are first associated with them via `FILES?` or `ADDTOFILES?`. Noticing a file updates certain lists and properties so that the file manager functions know to include the file in their operations. For example, `CLEANUP` will only dump files that have been noticed.

You can explicitly tell the file manager to notice a newly-created file by defining the filecoms for the file, and calling `ADDFILE`:

(**ADDFILE** *FILE*) [Function]

Tells the file manager that *FILE* should be recognized as a file; it adds *FILE* to `FILELST`, and also sets up the *FILE* property of *FILE* to reflect the current set of changes which are "registered against" *FILE*.

The file manager uses information stored on the property list of the root name of noticed files. The following property names are used:

FILE [Property Name]

When a file is noticed, the property `FILE`, value `((FILECOMS . LOADTYPE))` is added to the property list of its root name. *FILECOMS* is the variable containing the filecoms of the file. *LOADTYPE* indicates *how* the file was loaded, e.g., completely loaded, only partially loaded as with `LOADFNS`, loaded as a compiled file, etc.

The property `FILE` is used to determine whether or not the corresponding file has been modified since the last time it was loaded or dumped. `CDR` of the `FILE` property records by type those items that have been changed since the last `MAKEFILE`. Whenever a file is dumped, these items are moved to the property `FILECHANGES`, and `CDR` of the `FILE` property is reset to `NIL`.

FILECHANGES [Property Name]

The property `FILECHANGES` contains a list of all changed items since the file was loaded (there may have been several sequences of editing and rewriting the file). When a file is dumped, the changes in `CDR` of the `FILE` property are added to the `FILECHANGES` property.

FILEDATES [Property Name]

The property `FILEDATES` contains a list of version numbers and corresponding file dates for this file. These version numbers and dates are used for various integrity checks in connection with *remaking* a file.

FILEMAP

[Property Name]

The property **FILEMAP** is used to store the filemap for the file. This is used to directly load individual functions from the middle of a file.

To compute the root name, **ROOTFILENAME** is applied to the name of the file as indicated in the **FILECREATED** expression appearing at the front of the file, since this name corresponds to the name the file was originally made under. The file manager detects that the file being noticed is a compiled file (regardless of its name), by the appearance of more than one **FILECREATED** expressions. In this case, each of the files mentioned in the following **FILECREATED** expressions are noticed. For example, if you perform `(BCOMPL ' (FOO FIE))`, and subsequently loads `FOO.DCOM`, both `FOO` and `FIE` will be noticed.

When a file is noticed, its root name is added to the list **FILELST**:

FILELST

[Variable]

Contains a list of the root names of the files that have been noticed.

LOADEDFILELST

[Variable]

Contains a list of the actual names of the files as loaded by `LOAD`, `LOADFNS`, etc. For example, if you perform `(LOAD ' <NEWLISP>EDITA.COM;3)`, `EDITA` will be added to **FILELST**, but `<NEWLISP>EDITA.COM;3` is added to **LOADEDFILELST**. **LOADEDFILELST** is not used by the file manager; it is maintained solely for your benefit.

Distributing Change Information

Periodically, the function **UPDATEFILES** is called to find which file(s) contain the elements that have been changed. **UPDATEFILES** is called by **FILES?**, **CLEANUP**, and **MAKEFILES**, i.e., any procedure that requires the **FILE** property to be up to date. This procedure is followed rather than updating the **FILE** property after each change because scanning **FILELST** and examining each file manager command can be a time-consuming process; this is not so noticeable when performed in conjunction with a large operation like loading or writing a file.

UPDATEFILES operates by scanning **FILELST** and interrogating the file manager commands for each file. When (if) any files are found that contain the corresponding typed definition, the name of the element is added to the value of the property **FILE** for the corresponding file. Thus, after **UPDATEFILES** has completed operating, the files that need to be dumped are simply those files on **FILELST** for which **CDR** of their **FILE** property is non-NIL. For example, if you load the file `FOO` containing definitions for `FOO1`, `FOO2`, and `FOO3`, edit `FOO2`, and then call **UPDATEFILES**, `(GETPROP 'FOO 'FILE)` will be `((FOOCOMS . T) (FNS FOO2))`. If any objects marked as changed have not been transferred to the **FILE** property for some file, e.g., you define a new function but forget (or declines) to add it to the file manager commands for the corresponding file, then both **FILES?**

INTERLISP-D REFERENCE MANUAL

andCENPwl rn ann esgs n hncl DTFLS opri ■■■t pcf nwihflsteeiesbln.■■■ a loivk PAEIE iety
cify on which files these items belong.

You can also invoke UPDATEFILES directly:

(**UPDATEFILES** — —) [Function]

(UPDATEFILES) will update the FILE properties of the noticed files.

File Manager Types

In addition to the definitions of functions and values of variables, source files in Interlisp can contain a variety of other information, e.g. property lists, record declarations, macro definitions, hash arrays, etc. In order to treat such a diverse assortment of data uniformly from the standpoint of file operations, the file manager uses the concept of a *typed definition*, of which a function definition is just one example. A typed definition associates with a name (usually a symbol), a definition of a given type (called the file manager type). Note that the same name may have several definitions of different types. For example, a symbol may have both a function definition and a variable definition. The file manager also keeps track of the file that a particular typed definition is stored on, so one can think of a typed definition as a relation between four elements: a name, a definition, a type, and a file.

A file manager type is an abstract notion of a class of objects which share the property that every object of the same file manager type is stored, retrieved, edited, copied etc., by the file manager in the same way. Each file manager type is identified by a symbol, which can be given as an argument to the functions that manipulate typed definitions. You may define new file manager types, as described in the Defining New Package Types section.

FILEPKGTYPES [Variable]

The value of FILEPKGTYPES is a list of all file manager types, including any that you may have defined.

The file manager is initialized with the following built-in file manager types:

ADVICE [File Manager Type]

Used to access "advice" modifying a function (see Chapter 15).

ALISTS [File Manager Type]

Used to access objects stored on an association list that is the value of a symbol (see Chapter 3).

A variable is declared to have an association list as its value by putting on its property list the property VARTYPE with value ALIST. In this case, each dotted pair on the list is an object of type ALISTS. When the value of such a variable is changed, only those entries in the association list that are actually changed or added are marked as changed objects of

type ALISTS (with "name" (SYMBOL KEY)). Objects of type ALISTS are dumped via the ALISTS or ADDVARS file manager commands.

Note that some association lists are used to "implement" other file manager types. For example, the value of the global variable USERMACROS implements the file manager type USERMACROS and the values of LISPXMACROS and LISPXHISTORYMACROS implement the file manager type LISPXMACROS. This is indicated by putting on the property list of the variable the property VARTYPE with value a list of the form (ALIST FILEPKGTYPE). For example, (GETPROP 'LISPXHISTORYMACROS 'VARTYPE) => (ALIST LISPXMACROS).

COURIERPROGRAMS [File Manager Type]

Used to access Courier programs (see Chapter 31).

EXPRESSIONS [File Manager Type]

Used to access lisp expressions that are put on a file by using the REMEMBER programmers assistant command (Chapter 13), or by explicitly putting the P file manager command on the filecoms.

FIELDS [File Manager Type]

Used to access fields of records. The "definition" of an object of type FIELDS is a list of all the record declarations which contain the name. See Chapter 8.

FILEPKGCOMS [File Manager Type]

Used to access file manager commands and types. A single name can be defined both as a file manager type and a file manager command. The "definition" of an object of type FILEPKGCOMS is a list structure of the form ((COM . COMPROPS) (TYPE . TYPEPROPS)), where COMPROPS is a property list specifying how the name is defined as a file manager command by FILEPKGCOM (see the Defining New File Manager Commands section), and TYPEPROPS is a property list specifying how the name is defined as a file manager type by FILEPKGTYPE (see the Defining New File Manager Types section).

FILES [File Manager Type]

Used to access files. This file manager type is most useful for renaming files. The "definition" of a file is not a useful structure.

FILEVARS [File Manager Type]

Used to access Filevars (see the FileVars section).

FNS [File Manager Type]

Used to access function definitions.

INTERLISP-D REFERENCE MANUAL

I.S.OPRS	[File Manager Type]
Used to access the definitions of iterative statement operators (see Chapter 9).	
LISPMACROS	[File Manager Type]
Used to access programmer's assistant commands defined on the variables LISPMACROS and LISPMHISTORYMACROS (see Chapter 13).	
MACROS	[File Manager Type]
Used to access macro definitions (see Chapter 10).	
PROPS	[File Manager Type]
Used to access objects stored on the property list of a symbol (see Chapter 2). When a property is changed or added, an object of type PROPS, with "name" (SYMBOL <i>PROPNAME</i>) is marked as being changed.	
Note that some symbol properties are used to implement other file manager types. For example, the property MACRO implements the file manager type MACROS, the property ADVICE implements ADVICE, etc. This is indicated by putting the property PROPTYPE, with value of the file manager type on the property list of the property name. For example, (GETPROP 'MACRO 'PROPTYPE) => MACROS. When such a property is changed or added, an object of the corresponding file manager type is marked. If (GETPROP <i>PROPNAME</i> 'PROPTYPE) => IGNORE, the change is ignored. The FILE, FILEMAP, FILEDATES, etc. properties are all handled this way. (IGNORE cannot be the name of a file manager type implemented as a property).	
RECORDS	[File Manager Type]
Used to access record declarations (see Chapter 8).	
RESOURCES	[File Manager Type]
Used to access resources (see Chapter 12).	
TEMPLATES	[File Manager Type]
Used to access Masterscope templates (see Chapter 19).	
USERMACROS	[File Manager Type]
Used to access user edit macros (see Chapter 16).	
VAR	[File Manager Type]
Used to access top-level variable values.	

Functions for Manipulating Typed Definitions

The functions described below can be used to manipulate typed definitions, without needing to know how the manipulations are done. For example, (GETDEF 'FOO 'FNS) will return the function definition of FOO, (GETDEF 'FOO 'VARS) will return the variable value of FOO, etc. All of the functions use the following conventions:

1. All functions which make destructive changes are undoable.
2. Any argument that expects a list of symbols will also accept a single symbol, operating as though it were enclosed in a list. For example, if the argument *FILES* should be a list of files, it may also be a single file.
3. *TYPE* is a file manager type. *TYPE* = NIL is equivalent to *TYPE* = FNS. The singular form of a file manager type is also recognized, e.g. *TYPE* = VAR is equivalent to *TYPE* = VARS.
4. *FILES* = NIL is equivalent to *FILES* = FILELST.
5. *SOURCE* is used to indicate the source of a definition, that is, where the definition should be found. *SOURCE* can be one of:

CURRENT Get the definition currently in effect.

SAVED Get the "saved" definition, as stored by SAVEDEF.

FILE Get the definition contained on the (first) file determined by WHEREIS.

WHEREIS is called with *FILES* = T, so that if the WHEREIS library package is loaded, the WHEREIS data base will be used to find the file containing the definition.

? Get the definition currently in effect if there is one, else the saved definition if there is one, otherwise the definition from a file determined by WHEREIS. Like specifying CURRENT, SAVED, and FILE in order, and taking the first definition that is found.

a file name
a list of file names Get the definition from the first of the indicated files that contains one.

NIL In most cases, giving *SOURCE* = NIL (or not specifying it at all) is the same as giving ?, to get either the current, saved, or filed definition. However, with HASDEF, *SOURCE* = NIL is interpreted as equal to *SOURCE* = CURRENT, which only tests if there is a current definition.

INTERLISP-D REFERENCE MANUAL

The operation of most of the functions described below can be changed or extended by modifying the appropriate properties for the corresponding file manager type using the function `FILEPKGTYPE`, described in the Defining New File Manager Types section.

(**GETDEF** *NAME TYPE SOURCE OPTIONS*) [Function]

Returns the definition of *NAME*, of type *TYPE*, from *SOURCE*. For most types, `GETDEF` returns the expression which would be pretty printed when dumping *NAME* as *TYPE*. For example, for *TYPE* = `FNS`, an `EXPR` definition is returned, for *TYPE* = `VARS`, the value of *NAME* is returned, etc.

OPTIONS is a list which specifies certain options:

NOERROR `GETDEF` causes an error if an appropriate definition cannot be found, unless *OPTIONS* is or contains `NOERROR`. In this case, `GETDEF` returns the value of the `NULLDEF` file manager type property (see the Defining New File Manager Types section), usually `NIL`.

a string If *OPTIONS* is or contains a string, that string will be returned if no definition is found (and `NOERROR` is not among the options). The caller can thus determine whether a definition was found, even for types for which `NIL` or `NOBIND` are acceptable definitions.

NOCOPY `GETDEF` returns a copy of the definition unless *OPTIONS* is or contains `NOCOPY`.

EDIT If *OPTIONS* is or contains `EDIT`, `GETDEF` returns a copy of the definition unless it is possible to edit the definition "in place." With some file manager types, such as functions, it is meaningful (and efficient) to edit the definition by destructively modifying the list structure, without calling `PUTDEF`. However, some file manager types (like records) need to be "installed" with `PUTDEF` after they are edited. The default `EDITDEF` (see the Defining New File Manager Types section) calls `GETDEF` with *OPTIONS* of (`EDIT NOCOPY`), so it doesn't use a copy unless it has to, and only calls `PUTDEF` if the result of editing is not `EQUAL` to the old definition.

NODWIM A `FNS` definition will be dwimified if it is likely to contain `CLISP` unless *OPTIONS* is or contains `NODWIM`.

(**PUTDEF** *NAME TYPE DEFINITION REASON*) [Function]

Defines *NAME* of type *TYPE* with *DEFINITION*. For *TYPE* = `FNS`, does a `DEFINE`; for *TYPE* = `VARS`, does a `SAVESET`, etc.

For *TYPE* = FILES, PUTDEF establishes the command list, notices *NAME*, and then calls MAKEFILE to actually dump the file *NAME*, copying functions if necessary from the "old" file (supplied as part of *DEFINITION*).

PUTDEF calls MARKASCHANGED (see the Marking Changes section) to mark *NAME* as changed, giving a reason of *REASON*. If *REASON* is NIL, the default is DEFINED.

If *TYPE* = FNS, PUTDEF prints a warning if you try to redefine a function on the list UNSAFE.TO.MODIFY.FNS (see Chapter 10).

(**HASDEF** *NAME TYPE SOURCE SPELLFLG*) [Function]

Returns (OR *NAME* T) if *NAME* is the name of something of type *TYPE*. If not, attempts spelling correction if *SPELLFLG* = T, and returns the spelling-corrected *NAME*. Otherwise returns NIL. HASDEF for type FNS (or NIL) indicates that *NAME* has an editable source definition. If *NAME* is a function that exists on a file for which you have loaded only the compiled version and not the source, HASDEF returns NIL.

(HASDEF NIL *TYPE*) returns T if NIL has a valid definition.

If *SOURCE* = NIL, HASDEF interprets this as equal to *SOURCE* = CURRENT, which only tests if there is a current definition.

(**TYPESOF** *NAME POSSIBLETYPES IMPOSSIBLETYPES SOURCE*) [Function]

Returns a list of the types in *POSSIBLETYPES* but not in *IMPOSSIBLETYPES* for which *NAME* has a definition. FILEPKGTYPES is used if *POSSIBLETYPES* is NIL.

(**COPYDEF** *OLD NEW TYPE SOURCE OPTIONS*) [Function]

Defines *NEW* to have a copy of the definition of *OLD* by doing PUTDEF on a copy of the definition retrieved by (GETDEF *OLD TYPE SOURCE OPTIONS*). *NEW* is substituted for *OLD* in the copied definition, in a manner that may depend on the *TYPE*.

For example, (COPYDEF 'PDQ 'RST 'FILES) sets up RSTCOMS to be a copy of PDQCOMS, changes things like (VARS * PDQVARS) to be (VARS * RSTVARS) in RSTCOMS, and performs a MAKEFILE on RST such that the appropriate definitions get copied from PDQ.

COPYDEF disables the NOCOPY option of GETDEF, so *NEW* will always have a copy of the definition of *OLD*.

COPYDEF substitutes *NEW* for *OLD* throughout the definition of *OLD*. This is usually the right thing to do, but in some cases, e.g., where the old name appears within a quoted expression but was not used in the same context, you must re-edit the definition.

(**DELDEF** *NAME TYPE*) [Function]

Removes the definition of *NAME* as a *TYPE* that is currently in effect.

INTERLISP-D REFERENCE MANUAL

(**SHOWDEF** *NAME TYPE FILE*) [Function]

Prettyprints the definition of *NAME* as a *TYPE* to *FILE*. This shows you how *NAME* would be written to a file. Used by ADDTOFILES? (see the Storing Files section).

(**EDITDEF** *NAME TYPE SOURCE EDITCOMS*) [Function]

Edits the definition of *NAME* as a *TYPE*. Essentially performs

```
(PUTDEF NAME TYPE
  (EDITE (GETDEF NAME TYPE SOURCE)
    EDITCOMS )
```

(**SAVEDEF** *NAME TYPE DEFINITION*) [Function]

Sets the "saved" definition of *NAME* as a *TYPE* to *DEFINITION*. If *DEFINITION* = NIL, the current definition of *NAME* is saved.

If *TYPE* = FNS (or NIL), the function definition is saved on *NAME*'s property list under the property EXPR, or CODE (depending on the FNTYP of the function definition). If (GETD *NAME*) is non-NIL, but (FNTYP *FN*) = NIL, SAVEDEF saves the definition on the property name LIST. This can happen if a function was somehow defined with an illegal expr definition, such as (LAMMMMDA (X) ...).

If *TYPE* = VARS, the definition is stored as the value of the VALUE property of *NAME*. For other types, the definition is stored in an internal data structure, from where it can be retrieved by GETDEF or UNSAVEDEF.

(**UNSAVEDEF** *NAME TYPE*) [Function]

Restores the "saved" definition of *NAME* as a *TYPE*, making it be the current definition. Returns *PROP*.

If *TYPE* = FNS (or NIL), UNSAVEDEF unsaves the function definition from the EXPR property if any, else CODE, and returns the property name used. UNSAVEDEF also recognizes *TYPE* = EXPR, CODE, or LIST, meaning to unsave the definition only from the corresponding property only.

If DFNFLG is not T (see Chapter 10), the current definition of *NAME*, if any, is saved using SAVEDEF. Thus one can use UNSAVEDEF to switch back and forth between two definitions.

(**LOADDEF** *NAME TYPE SOURCE*) [Function]

Equivalent to (PUTDEF *NAME TYPE* (GETDEF *NAME TYPE SOURCE*)). LOADDEF is essentially a generalization of LOADFNS, e.g. it enables loading a single record declaration from a file. (LOADDEF *FN*) will give *FN* an EXPR definition, either obtained from its property list or a file, unless it already has one.

(**CHANGECALLERS** *OLD NEW TYPES FILES METHOD*)

[Function]

Finds all of the places where *OLD* is used as any of the types in *TYPES* and changes those places to use *NEW*. For example, (CHANGECALLERS 'NLSETQ 'ERSETQ) will change all calls to NLSETQ to be calls to ERSETQ. Also changes occurrences of *OLD* to *NEW* inside the filecoms of any file, inside record declarations, properties, etc.

CHANGECALLERS attempts to determine if *OLD* might be used as more than one type; for example, if it is both a function and a record field. If so, rather than performing the transformation *OLD* -> *NEW* automatically, you are allowed to edit all of the places where *OLD* occurs. For each occurrence of *OLD*, you are asked whether you want to make the replacement. If you respond with anything except Yes or No, the editor is invoked on the expression containing that occurrence.

There are two different methods for determining which functions are to be examined. If *METHOD* = EDITCALLERS, EDITCALLERS is used to search *FILES* (see Chapter 16). If *METHOD* = MASTERSCOPE, then the Masterscope database is used instead. *METHOD* = NIL defaults to MASTERSCOPE if the value of the variable DEFAULTRENAMEMETHOD is MASTERSCOPE and a Masterscope database exists, otherwise it defaults to EDITCALLERS.

(**RENAME** *OLD NEW TYPES FILES METHOD*)

[Function]

First performs (COPYDEF *OLD NEW TYPE*) for all *TYPE* inside *TYPES*. It then calls CHANGECALLERS to change all occurrences of *OLD* to *NEW*, and then "deletes" *OLD* with DELDEF. For example, if you have a function FOO which you now wish to call FIE, simply perform (RENAME 'FOO 'FIE), and FIE will be given FOO's definition, and all places that FOO are called will be changed to call FIE instead.

METHOD is interpreted the same as the *METHOD* argument to CHANGECALLERS, above.

(**COMPARE** *NAME NAME TYPE SOURCE SOURCE*)

[Function]

Compares the definition of *NAME* with that of *NAME* , by calling COMPARELISTS (Chapter 3) on (GETDEF *NAME TYPE SOURCE*) and (GETDEF *NAME TYPE SOURCE*), which prints their differences on the terminal.

For example, if the current value of the variable A is (A B C (D E F) G), and the value of the variable B on the file <lisp>FOO is (A B C (D F E) G), then:

```
←(COMPARE 'A 'B 'VARS 'CURRENT '<lisp>FOO)
A from CURRENT and B from <lisp>TEST differ:
(E -> F) (F -> E)
T
```

(**COMPAREDEFS** *NAME TYPE SOURCES*)

[Function]

Calls COMPARELISTS (Chapter 3) on all pairs of definitions of *NAME* as a *TYPE* obtained from the various *SOURCES* (interpreted as a list of source specifications).

INTERLISP-D REFERENCE MANUAL

Defining New File Manager Types

All manipulation of typed definitions in the file manager is done using the type-independent functions `GETDEF`, `PUTDEF`, etc. Therefore, to define a new file manager type, it is only necessary to specify (via the function `FILEPKGTYPE`) what these functions should do when dealing with a typed definition of the new type. Each file manager type has the following properties, whose values are functions or lists of functions:

These functions are defined to take a *TYPE* argument so that you may have the same function for more than one type.

GETDEF [File Manager Type Property]

Value is a function of three arguments, *NAME*, *TYPE*, and *OPTIONS*, which should return the current definition of *NAME* as a type *TYPE*. Used by `GETDEF` (see the Functions for Manipulating Typed Definitions section), which passes its *OPTIONS* argument.

If there is no `GETDEF` property, a file manager command for dumping *NAME* is created (by `MAKENEWCOM`). This command is then used to write the definition of *NAME* as a type *TYPE* onto the file `FILEPKG.SCRATCH` (in Medley, this file is created on the `{CORE}` device). This expression is then read back in and returned as the current definition.

In some situations, the function `HASDEF` needs to call `GETDEF` to determine whether a definition exists. In this case, *OPTIONS* will include the symbol `HASDEF`, and it is permissible for a `GETDEF` function to return `T` or `NIL`, rather than creating a complex structure which will not be used.

NULLDEF [File Manager Type Property]

The value of the `NULLDEF` property is returned by `GETDEF` (see the Functions for Manipulating Typed Definitions section) when there is no definition and the `NOERROR` option is supplied. For example, the `NULLDEF` of `VARS` is `NOBIND`.

FILEGETDEF [File Manager Type Property]

This enables you to provide a way of obtaining definitions from a file that is more efficient than the default procedure used by `GETDEF` (see the Functions for Manipulating Typed Definitions section). Value is a function of four arguments, *NAME*, *TYPE*, *FILE*, and *OPTIONS*. The function is applied by `GETDEF` when it is determined that a typed definition is needed from a particular file. The function must open and search the given file and return any *TYPE* definition for *NAME* that it finds.

CANFILEDEF [File Manager Type Property]

If the value of this property is non-`NIL`, this indicates that definitions of this file manager type are not loaded when a file is loaded with `LOADFROM` (see the Loading Files section). The default is `NIL`. Initially, only `FNS` has this property set to non-`NIL`.

PUTDEF [File Manager Type Property]

Value is a function of three arguments, *NAME*, *TYPE*, and *DEFINITION*, which should store *DEFINITION* as the definition of *NAME* as a type *TYPE*. Used by PUTDEF (see the Functions for Manipulating Typed Definitions section).

HASDEF [File Manager Type Property]

Value is a function of three arguments, *NAME*, *TYPE*, and *SOURCE*, which should return (OR *NAME* T) if *NAME* is the name of something of type *TYPE*. *SOURCE* is as interpreted by HASDEF (see the Functions for Manipulating Typed Definitions section), which uses this property.

EDITDEF [File Manager Type Property]

Value is a function of four arguments, *NAME*, *TYPE*, *SOURCE*, and *EDITCOMS*, which should edit the definition of *NAME* as a type *TYPE* from the source *SOURCE*, interpreting the edit commands *EDITCOMS*. If successful, should return *NAME* (or a spelling-corrected *NAME*). If it returns NIL, the "default" editor is called. Used by EDITDEF (see the Functions for Manipulating Typed Definitions section).

DELDEF [File Manager Type Property]

Value is a function of two arguments, *NAME*, and *TYPE*, which removes the definition of *NAME* as a *TYPE* that is currently in effect. Used by DELDEF (see the Functions for Manipulating Typed Definitions section).

NEWCOM [File Manager Type Property]

Value is a function of four arguments, *NAME*, *TYPE*, *LISTNAME*, and *FILE*. Specifies how to make a new (instance of a) file manager command to dump *NAME*, an object of type *TYPE*. The function should return the new file manager command. Used by ADDTOFILE and SHOWDEF.

If *LISTNAME* is non-NIL, this means that you specified *LISTNAME* as the filevar in interaction with ADDTOFILES? (see the FileVars section).

If no NEWCOM is specified, the default is to call DEFAULTTAKENEWCOM, which will construct and return a command of the form (*TYPE NAME*). You can advise or redefine DEFAULTTAKENEWCOM.

WHENCHANGED [File Manager Type Property]

Value is a list of functions to be applied to *NAME*, *TYPE*, and *REASON* when *NAME*, an instance of type *TYPE*, is changed or defined (see MARKASCHANGED, in the Marking Changes section). Used for various applications, e.g. when an object of type I.S.OPRS changes, it is necessary to clear the corresponding translations from CLISPARRAY.

The WHENCHANGED functions are called before the object is marked as changed, so that it can, in fact, decide that the object is *not* to be marked as changed, and execute (RETFROM 'MARKASCHANGED).

INTERLISP-D REFERENCE MANUAL

The *REASON* argument passed to *WHENCHANGED* functions is either *DEFINED* or *CHANGED*.

WHENFILED [File Manager Type Property]

Value is a list of functions to be applied to *NAME*, *TYPE*, and *FILE* when *NAME*, an instance of type *TYPE*, is added to *FILE*.

WHENUNFILED [File Manager Type Property]

Value is a list of functions to be applied to *NAME*, *TYPE*, and *FILE* when *NAME*, an instance of type *TYPE*, is removed from *FILE*.

DESCRIPTION [File Manager Type Property]

Value is a string which describes instances of this type. For example, for type *RECORDS*, the value of *DESCRIPTION* is the string "record declarations".

The function *FILEPKGTYPE* is used to define new file manager types, or to change the properties of existing types. It is possible to redefine the attributes of system file manager types, such as *FNS* or *PROPS*.

(**FILEPKGTYPE** *TYPE PROP VAL ... PROP VAL*) [NoSpread Function]

Nospread function for defining new file manager types, or changing properties of existing file manager types. *PROP* is one of the property names given above; *VAL* is the value to be given to that property. Returns *TYPE*.

(*FILEPKGTYPE TYPE PROP*) returns the value of the property *PROP*, without changing it.

(*FILEPKGTYPE TYPE*) returns a property list of all of the defined properties of *TYPE*, using the property names as keys.

Specifying *TYPE* as the symbol *TYPE* can be used to define one file manager type as a synonym of another. For example, (*FILEPKGTYPE 'R 'TYPE 'RECORDS*) defines *R* as a synonym for the file manager type *RECORDS*.

File Manager Commands

The basic mechanism for creating symbolic files is the function *MAKEFILE* (see the Storing Files section). For each file, the file manager has a data structure known as the "filecoms", which specifies what typed descriptions are contained in the file. A filecoms is a list of file manager commands, each of which specifies objects of a certain file manager type which should be dumped. For example, the filecoms

```
((FNS FOO)
 (VARS FOO BAR BAZ)
 (RECORDS XZZZY))
```

has a FNS, a VARS, and a RECORDS file manager command. This filecoms specifies that the function definition for FOO, the variable values of FOO, BAR, and BAZ, and the record declaration for XYZZY should be dumped.

By convention, the filecoms of a file *X* is stored as the value of the symbol *XCOMS*. For example, (MAKEFILE 'FOO. ;27) will use the value of FOOCOMS as the filecoms. This variable can be directly manipulated, but the file manager contains facilities which make constructing and updating filecoms easier, and in some cases automatic (see the Functions for Manipulating File Command Lists section).

A file manager command is an instruction to MAKEFILE to perform an explicit, well-defined operation, usually printing an expression. Usually there is a one-to-one correspondence between file manager types and file manager commands; for each file manager type, there is a file manager command which is used for writing objects of that type to a file, and each file manager command is used to write objects of a particular type. However, in some cases, the same file manager type can be dumped by several different file manager commands. For example, the file manager commands PROP, IFPROP, and PROPS all dump out objects with the file manager type PROPS. This means if you change an object of file manager type PROPS via EDITP, a typed-in call to PUTPROP, or via an explicit call to MARKASCHANGED, this object can be written out with any of the above three commands. Thus, when the file manager attempts to determine whether this typed object is contained on a particular file, it must look at instances of all three file manager commands PROP, IFPROP, and PROPS, to see if the corresponding atom and property are specified. It is also permissible for a single file manager command to dump several different file manager types. For example, you can define a file manager command which dumps both a function definition and its macro. Conversely, some file manager commands do not dump any file manager types at all, such as the E command.

For each file manager command, the file manager must be able to determine what typed definitions the command will cause to be printed so that the file manager can determine on what file (if any) an object of a given type is contained (by searching through the filecoms). Similarly, for each file manager type, the file manager must be able to construct a command that will print out an object of that type. In other words, the file manager must be able to map file manager commands into file manager types, and vice versa. Information can be provided to the file manager about a particular file manager command via the function FILEPKGCOM (see the Defining New File Manager Commands section), and information about a particular file manager type via the function FILEPKGTYPE (see the prior section). In the absence of other information, the default is simply that a file manager command of the form (*X NAME*) prints out the definition of *NAME* as a type *X*, and, conversely, if *NAME* is an object of type *X*, then *NAME* can be written out by a command of the form (*X NAME*).

If a file manager function is given a command or type that is not defined, it attempts spelling correction using FILEPKGCOMSPLST as a spelling list (unless DWIMFLG or NOSPELLFLG = NIL; see Chapter 20). If successful, the corrected version of the list of file manager commands is written (again) on the output file, since at this point, the uncorrected list of file manager commands would already have been printed on the output file. When the file is loaded, this will result in FILECOMS being reset, and may cause a message to be printed, e.g., (FOOCOMS RESET). The value of FOOCOMS would then be the corrected version. If the spelling correction is unsuccessful, the file manager functions generate an error, BAD FILE PACKAGE COMMAND.

INTERLISP-D REFERENCE MANUAL

File package commands can be used to save on the output file definitions of functions, values of variables, property lists of atoms, advised functions, edit macros, record declarations, etc. The interpretation of each file manager command is documented in the following sections.

(**USERMACROS** *SYMBOL* ... *SYMBOL*) [File Manager Command]

Each symbol *SYMBOL* is the name of a user edit macro. Writes expressions to add the edit macro definitions of *SYMBOL* to USERMACROS, and adds the names of the commands to the appropriate spelling lists.

If *SYMBOL* is not a user macro, a warning message "no EDIT MACRO for *SYMBOL* " is printed.

Functions and Macros

(**FNS** *FN* ... *FN*) [File Manager Command]

Writes a DEFINEQ expression with the function definitions of *FN* ... *FN* .

You should never print a DEFINEQ expression directly onto a file (by using the P file manager command, for example), because MAKEFILE generates the filemap of function definitions from the FNS file manager commands (see the File Maps section).

(**ADVISE** *FN* ... *FN*) [File Manager Command]

For each function *FN* , writes expressions to reinstate the function to its advised state when the file is loaded. See Chapter 15.

When advice is applied to a function programmatically or by hand, it is additive. That is, if a function already has some advice, further advice is added to the already-existing advice. However, when advice is applied to a function as a result of loading a file with an ADVISE file manager command, the new advice replaces any earlier advice. ADVISE works this way to prevent problems with loading different versions of the same advice. If you really want to apply additive advice, a file manager command such as (P (ADVISE ...)) should be used (see the Miscellaneous File Manager Commands section).

(**ADVISE** *FN* ... *FN*) [File Manager Command]

For each function *FN* , writes a PUTPROPS expression which will put the advice back on the property list of the function. You can then use READADVISE (see Chapter 15) to reactivate the advice.

(**MACROS** *SYMBOL* ... *SYMBOL*) [File Manager Command]

Each *SYMBOL* is a symbol with a MACRO definition (and/or a DMACRO, 10MACRO, etc.). Writes out an expression to restore all of the macro properties for each *SYMBOL* , embedded in a DECLARE: EVAL@COMPILE so the macros will be defined when the file is compiled. See Chapter 10.

Variables

(VARS VAR ... VAR) [File Manager Command]

For each *VAR* , writes an expression to set its top level value when the file is loaded. If *VAR* is atomic, VARS writes out an expression to set *VAR* to the top-level value it had at the time the file was written. If *VAR* is non-atomic, it is interpreted as (*VAR FORM*) , and VARS write out an expression to set *VAR* to the value of *FORM* (evaluated when the file is loaded).

VARS prints out expressions using RPAQQ and RPAQ, which are like SETQQ and SETQ except that they also perform some special operations with respect to the file manager (see the Functions Used within Source Files section).

VARS cannot be used for putting arbitrary variable values on files. For example, if the value of a variable is an array (or many other data types), a symbol which represents the array is dumped in the file instead of the array itself. The HORRIBLEVARS file manager command provides a way of saving and reloading variables whose values contain re-entrant or circular list structure, user data types, arrays, or hash arrays.

(INITVARS VAR ... VAR) [File Manager Command]

INITVARS is used for initializing variables, setting their values only when they are currently NOBIND. A variable value defined in an INITVARS command will not change an already established value. This means that re-loading files to get some other information will not automatically revert to the initialization values.

The format of an INITVARS command is just like VARS. The only difference is that if *VAR* is atomic, the current value is not dumped; instead NIL is defined as the initialization value. Therefore, (INITVARS FOO (FUM 2)) is the same as (VARS (FOO NIL) (FUM 2)) , if FOO and FUM are both NOBIND.

INITVARS writes out an RPAQ? expression on the file instead of RPAQ or RPAQQ.

(ADDVARS (VAR . LST) ... (VAR . LST)) [File Manager Command]

For each (*VAR . LST*) , writes an ADDTOVAR (see the Functions Used Within Source Files section) to add each element of *LST* to the list that is the value of *VAR* at the time the file is loaded. The new value of *VAR* will be the union of its old value and *LST* . If the value of *VAR* is NOBIND, it is first set to NIL.

For example, (ADDVARS (DIRECTORIES LISP LISPUSERS)) will add LISP and LISPUSERS to the value of DIRECTORIES.

If *LST* is not specified, *VAR* is initialized to NIL if its current value is NOBIND. In other words, (ADDVARS (*VAR*)) will initialize *VAR* to NIL if *VAR* has not previously been set.

INTERLISP-D REFERENCE MANUAL

(APPENDVARS (VAR . LST) ... (VAR . LST)) [File Manager Command]

The same as ADDVARS, except that the values are added to the end of the lists (using APPENDTOVAR, in the Functions Used Within Source Files section), rather than at the beginning.

(UGLYVARS VAR ... VAR) [File Manager Command]

Like VARS, except that the value of each VAR may contain structures for which READ is not an inverse of PRINT, e.g. arrays, readtables, user data types, etc. Uses HPRINT (see Chapter 25).

(HORRIBLEVARS VAR ... VAR) [File Manager Command]

Like UGLYVARS, except structures may also contain circular pointers. Uses HPRINT (see Chapter 25). The values of VAR ... VAR are printed in the same operation, so that they may contain pointers to common substructures.

UGLYVARS does not do any checking for circularities, which results in a large speed and internal-storage advantage over HORRIBLEVARS. Thus, if it is known that the data structures do *not* contain circular pointers, UGLYVARS should be used instead of HORRIBLEVARS.

(ALISTS (VAR KEY KEY ...) ... (VAR KEY KEY ...)) [File Manager Command]

VAR is a variable whose value is an association list, such as EDITMACROS, BAKTRACELST, etc. For each VAR, ALISTS writes out expressions which will restore the values associated with the specified keys. For example, (ALISTS (BREAKMACROS BT BTV)) will dump the definition for the BT and BTV commands on BREAKMACROS.

Some association lists (USERMACROS, LISPXMACHOS, etc.) are used to implement other file manager types, and they have their own file manager commands.

(SPECVARS VAR ... VAR) [File Manager Command]

(LOCALVARS VAR ... VAR) [File Manager Command]

(GLOBALVARS VAR ... VAR) [File Manager Command]

Outputs the corresponding compiler declaration embedded in a DECLARE: DOEVAL@COMPILE DONTCOPY. See Chapter 18.

(CONSTANTS VAR ... VAR) [File Manager Command]

Like VARS, for each VAR writes an expression to set its top level value when the file is loaded. Also writes a CONSTANTS expression to declare these variables as constants (see Chapter 18). Both of these expressions are wrapped in a (DECLARE: EVAL@COMPILE ...) expression, so they can be used by the compiler.

Like VARS, VAR can be non-atomic, in which case it is interpreted as (VAR FORM), and passed to CONSTANTS (along with the variable being initialized to FORM).

Symbol Properties

(**PROP** *PROPNAME* *SYMBOL* ... *SYMBOL*) [File Manager Command]

Writes a PUTPROPS expression to restore the value of the *PROPNAME* property of each symbol *SYMBOL* when the file is loaded.

If *PROPNAME* is a list, expressions will be written for each property on that list. If *PROPNAME* is the symbol *ALL*, the values of all user properties (on the property list of each *SYMBOL*) are saved. *SYSPROPS* is a list of properties used by system functions. Only properties *not* on that list are dumped when the *ALL* option is used.

If *SYMBOL* does not have the property *PROPNAME* (as opposed to having the property with value *NIL*), a warning message "NO *PROPNAME* PROPERTY FOR *SYMBOL* " is printed. The command *IFPROP* can be used if it is not known whether or not an atom will have the corresponding property.

(**IFPROP** *PROPNAME* *SYMBOL* ... *SYMBOL*) [File Manager Command]

Same as the *PROP* file manager command, except that it only saves the properties that actually appear on the property list of the corresponding atom. For example, if *FOO1* has property *PROP1* and *PROP2*, *FOO2* has *PROP3*, and *FOO3* has property *PROP1* and *PROP3*, then (*IFPROP* (*PROP1* *PROP2* *PROP3*) *FOO1* *FOO2* *FOO3*) will save only those five property values.

(**PROPS** (*SYMBOL* *PROPNAME*) ... (*SYMBOL* *PROPNAME*)) [File Manager Command]

Similar to *PROP* command. Writes a PUTPROPS expression to restore the value of *PROPNAME* for each *SYMBOL* when the file is loaded.

As with the *PROP* command, if *SYMBOL* does not have the property *PROPNAME* (as opposed to having the property with *NIL* value), a warning message "NO *PROPNAME* PROPERTY FOR *SYMBOL* " is printed.

Miscellaneous File Manager Commands

(**RECORDS** *REC* ... *REC*) [File Manager Command]

Each *REC* is the name of a record (see Chapter 8). Writes expressions which will redeclare the records when the file is loaded.

(**INITRECORDS** *REC* ... *REC*) [File Manager Command]

Similar to *RECORDS*, *INITRECORDS* writes expressions on a file that will, when loaded, perform whatever initialization/allocation is necessary for the indicated records. However, the record declarations themselves are not written out. This facility is useful for building systems on top of Interlisp, in which the implementor may want to eliminate the record declarations from a production version of the system, but the allocation for these records must still be done.

INTERLISP-D REFERENCE MANUAL

(**LISPXMACROS** *SYMBOL* ... *SYMBOL*) [File Manager Command]

Each *SYMBOL* is defined on LISPXMACROS or LISPXHISTORYMACROS (see Chapter 13). Writes expressions which will save and restore the definition for each macro, as well as making the necessary additions to LISPXCOMS

(**I.S.OPRS** *OPR* ... *OPR*) [File Manager Command]

Each *OPR* is the name of a user-defined i.s.opr (see Chapter 9). Writes expressions which will redefine the i.s.oprs when the file is loaded.

(**RESOURCES** *RESOURCE* ... *RESOURCE*) [File Manager Command]

Each *RESOURCES* is the name of a resource (see Chapter 12). Writes expressions which will redeclare the resource when the file is loaded.

(**INITRESOURCES** *RESOURCE* ... *RESOURCE*) [File Manager Command]

Parallel to INITRECORDS, INITRESOURCES writes expressions on a file to perform whatever initialization/allocation is necessary for the indicated resources, without writing the resource declaration itself.

(**COURIERPROGRAMS** *NAME* ... *NAME*) [File Manager Command]

Each *NAME* is the name of a Courier program (see Chapter 31). Writes expressions which will redeclare the Courier program when the file is loaded.

(**TEMPLATES** *SYMBOL* ... *SYMBOL*) [File Manager Command]

Each *SYMBOL* is a symbol which has a Masterscope template (see Chapter 19). Writes expressions which will restore the templates when the file is loaded.

(**FILES** *FILE* ... *FILE*) [File Manager Command]

Used to specify auxiliary files to be loaded in when the file is loaded. Dumps an expression calling FILESLOAD (see the Loading Files section), with *FILE* ... *FILE* as the arguments. FILESLOAD interprets *FILE* ... *FILE* as files to load, possibly interspersed with lists used to specify certain loading options.

(**FILEPKGCOMS** *SYMBOL* ... *SYMBOL*) [File Manager Command]

Each symbol *SYMBOL* is either the name of a user-defined file manager command or a user-defined file manager type (or both). Writes expressions which will restore each command/type.

If *SYMBOL* is not a file manager command or type, a warning message "no FILE PACKAGE COMMAND for *SYMBOL* " is printed.

(***** . *TEXT*) [File Manager Command]

Used for inserting comments in a file. The file manager command is simply written on the output file; it will be ignored when the file is loaded.

If the first element of *TEXT* is another ***, a form-feed is printed on the file before the comment.

(**P** *EXP* ... *EXP*) [File Manager Command]

Writes each of the expressions *EXP* ... *EXP* on the output file, where they will be evaluated when the file is loaded.

(**E** *FORM* ... *FORM*) [File Manager Command]

Each of the forms *FORM* ... *FORM* is evaluated at *output* time, when MAKEFILE interpretes this file manager command.

(**COMS** *COM* ... *COM*) [File Manager Command]

Each of the commands *COM* ... *COM* is interpreted as a file manager command.

(**ORIGINAL** *COM* ... *COM*) [File Manager Command]

Each of the commands *COM* will be interpreted as a file manager command without regard to any file manager macros (as defined by the **MACRO** property of the **FILEPKGCOM** function, in the Defining New File Manager Commands section). Useful for redefining a built-in file manager command in terms of itself.

Some of the "built-in" file manager commands are defined by file manager macros, so interpreting them (or new user-defined file manager commands) with **ORIGINAL** will fail. **ORIGINAL** was never intended to be used outside of a file manager command macro.

DECLARE:

(**DECLARE** : . *FILEPKGCOMS/FLAGS*) [File Manager Command]

Normally expressions written onto a symbolic file are evaluated when loaded; copied to the compiled file when the symbolic file is compiled (see Chapter 18); and not evaluated at compile time. **DECLARE** : allows you to override these defaults.

FILEPKGCOMS/FLAGS is a list of file manager commands, possibly interspersed with "tags". The output of those file manager commands within *FILEPKGCOMS/FLAGS* is embedded in a **DECLARE** : expression, along with any tags that are specified. For example, (**DECLARE** : **EVAL@COMPILE** **DONTCOPY** (**FNS** ...) (**PROP** ...)) would produce (**DECLARE** : **EVAL@COMPILE** **DONTCOPY** (**DEFINEQ** ...) (**PUTPROPS** ...)). **DECLARE** : is *defined* as an *nlambda* *nospread* function, which processes its arguments by evaluating or not evaluating each expression depending on the setting of internal state variables. The initial setting is to evaluate, but this can be overridden by specifying the **DONTEVAL@LOAD** tag.

DECLARE : expressions are specially processed by the compiler. For the purposes of compilation, **DECLARE** : has two principal applications: to specify forms that are to be evaluated at compile time, presumably to affect the compilation, e.g., to set up macros; and /or to indicate which expressions appearing in the symbolic file are *not* to be copied to the output file. (Normally, expressions are *not* evaluated and *are* copied.) Each expression

INTERLISP-D REFERENCE MANUAL

in CDR of a `DECLARE:` form is either evaluated/not-evaluated and copied/not-copied depending on the settings of two internal state variables, initially set for copy and not-evaluate. These state variables can be reset for the remainder of the expressions in the `DECLARE:` by means of the tags `DONTCOPY`, `EVAL@COMPILE`, etc.

The tags are:

EVAL@LOAD

DOEVAL@LOAD Evaluate the following forms when the file is loaded (unless overridden by `DONTEVAL@LOAD`).

DONTEVAL@LOAD Do not evaluate the following forms when the file is loaded.

EVAL@LOADWHEN This tag can be used to provide conditional evaluation. The value of the expression immediately following the tag determines whether or not to evaluate subsequent expressions when loading. ... `EVAL@LOADWHEN T` ... is equivalent to ... `EVAL@LOAD` ...

COPY

DOCOPY When compiling, copy the following forms into the compiled file.

DONTCOPY When compiling, do not copy the following forms into the compiled file.

Note: If the file manager commands following `DONTCOPY` include record declarations for datatypes, or records with initialization forms, it is necessary to include a `INITRECORDS` file manager command (see the prior section) outside of the `DONTCOPY` form so that the initialization information is copied. For example, if `FOO` was defined as a datatype,

```
(DECLARE: DONTCOPY (RECORDS FOO))  
(INITRECORDS FOO)
```

would copy the data type declaration for `FOO`, but would not copy the whole record declaration.

COPYWHEN When compiling, if the next form evaluates to non-NIL, copy the following forms into the compiled file.

EVAL@COMPILE

DOEVAL@COMPILE When compiling, evaluate the following forms.

DONTEVAL@COMPILE When compiling, do not evaluate the following forms.

EVAL@COMPILEWHEN When compiling, if the next form evaluates to non-NIL, evaluate the following forms.

FIRST For expressions that are to be copied to the compiled file, the tag **FIRST** can be used to specify that the following expressions in the **DECLARE:** are to appear at the front of the compiled file, before anything else except the **FILECREATED** expressions (see the Symbolic File Format section). For example, **(DECLARE: COPY FIRST (P (PRINT MESS1 T)) NOTFIRST (P (PRINT MESS2 T)))** will cause **(PRINT MESS1 T)** to appear first in the compiled file, followed by any functions, then **(PRINT MESS2 T)**.

NOTFIRST Reverses the effect of **FIRST**.

The value of **DECLARETAGSLST** is a list of all the tags used in **DECLARE:** expressions. If a tag not on this list appears in a **DECLARE:** file manager command, spelling correction is performed using **DECLARETAGSLST** as a spelling list.

Note that the function **LOADCOMP** (see the Loading Files section) provides a convenient way of obtaining information from the **DECLARE:** expressions in a file, without reading in the entire file. This information may be used for compiling other files.

(BLOCKS BLOCK . . . BLOCK)

[File Manager Command]

For each **BLOCK** , writes a **DECLARE:** expression which the block compile functions interpret as a block declaration. See Chapter 18.

Exporting Definitions

When building a large system in Interlisp, it is often the case that there are record definitions, macros and the like that are needed by several different system files when running, analyzing and compiling the source code of the system, but which are not needed for running the compiled code. By using the **DECLARE:** file manager command with tag **DONTCOPY** (see the prior section), these definitions can be kept out of the compiled files, and hence out of the system constructed by loading the compiled files into Interlisp. This saves loading time, space in the resulting system, and whatever other overhead might be incurred by keeping those definitions around, e.g., burden on the record package to consider more possibilities in translating record accesses, or conflicts between system record fields and user record fields.

However, if the implementor wants to debug or compile code in the resulting system, the definitions are needed. And even if the definitions *had* been copied to the compiled files, a similar problem arises if one wants to work on system code in a regular Interlisp environment where none of the system files had been loaded. One could mandate that any definition needed by more than one file in the system should reside on a distinguished file of definitions, to be loaded into any environment where the system files are worked on. Unfortunately, this would keep the definitions away from where they logically belong. The **EXPORT** mechanism is designed to solve this problem.

To use the mechanism, the implementor identifies any definitions needed by files other than the one in which the definitions reside, and wraps the corresponding file manager commands in the **EXPORT**

INTERLISP-D REFERENCE MANUAL

file manager command. Thereafter, GATHEREXPORTS can be used to make a single file containing all the exports.

(**EXPORT** *COM* . . . *COM*) [File Manager Command]

This command is used for "exporting" definitions. Like *COM*, each of the commands *COM* . . . *COM* is interpreted as a file manager command. The commands are also flagged in the file as being "exported" commands, for use with GATHEREXPORTS.

(**GATHEREXPORTS** *FROMFILES* *TOFILE* *FLG*) [Function]

FROMFILES is a list of files containing **EXPORT** commands. **GATHEREXPORTS** extracts all the exported commands from those files and produces a loadable file *TOFILE* containing them. If *FLG* = *EVAL*, the expressions are evaluated as they are gathered; i.e., the exports are effectively loaded into the current environment as well as being written to *TOFILE*.

(**IMPORTFILE** *FILE* *RETURNFLG*) [Function]

If *RETURNFLG* is *NIL*, this loads any exported definitions from *FILE* into the current environment. If *RETURNFLG* is *T*, this returns a list of the exported definitions (evaluable expressions) without actually evaluating them.

(**CHECKIMPORTS** *FILES* *NOASKFLG*) [Function]

Checks each of the files in *FILES* to see if any exists in a version newer than the one from which the exports in memory were taken (**GATHEREXPORTS** and **IMPORTFILE** note the creation dates of the files involved), or if any file in the list has not had its exports loaded at all. If there are any such files, you are asked for permission to **IMPORTFILE** each such file. If *NOASKFLG* is non-*NIL*, **IMPORTFILE** is performed without asking.

For example, suppose file *FOO* contains records *R1*, *R2*, and *R3*, macros *BAR* and *BAZ*, and constants *CON1* and *CON2*. If the definitions of *R1*, *R2*, *BAR*, and *BAZ* are needed by files other than *FOO*, then the file commands for *FOO* might contain the command

```
(DECLARE: EVAL@COMPILE DONTCOPY
  (EXPORT (RECORDS R1 R2)
    (MACROS BAR BAZ) )
  (RECORDS R3)
  (CONSTANTS BAZ) )
```

None of the commands inside this **DECLARE:** would appear on *FOO*'s compiled file, but (**GATHEREXPORTS** ' (*FOO*) 'MYEXPORTS) would copy the record definitions for *R1* and *R2* and the macro definitions for *BAR* and *BAZ* to the file *MYEXPORTS*.

FileVars

In each of the file manager commands described above, if the symbol * follows the command type, the form following the *, i.e., *CADDR* of the command, is evaluated and its value used in executing the command, e.g., (**FNS** * (APPEND *FNS1* *FNS2*)). When this form is a symbol, e.g. (**FNS** *

FOOFNS), we say that the variable is a "filevar". Note that (COMS * FORM) provides a way of *computing* what should be done by MAKEFILE.

Example:

```
← (SETQ FOOFNS '(FOO1 FOO2 FOO3))
  (FOO1 FOO2 FOO3)

← (SETQ FOOCOMS
  '((FNS * FOOFNS)
    (VARS FIE)
    (PROP MACRO FOO1 FOO2)
    (P (MOVD 'FOO1 'FIE1))))

← (MAKEFILE 'FOO)
```

would create a file FOO containing:

```
(FILECREATED "time and date the file was made" . "other
information")
(PRETTYCOMPRINT FOOCOMS)
(RPAQQ FOOCOMS ((FNS * FOOFNS) ...))
(RPAQQ FOOFNS (FOO1 FOO2 FOO3))
(DEFINEQ "definitions of FOO1, FOO2, and FOO3")
(RPAQQ FIE "value of FIE")
(PUTPROPS FOO1 MACRO PROPVALUE)
(PUTPROPS FOO2 MACRO PROPVALUE)
(MOVD (QUOTE FOO1) (QUOTE FIE1))
STOP
```

For the PROP and IFPROP commands (see the Litatom Properties section), the * follows the property name instead of the command, e.g., (PROP MACRO * FOOMACROS). Also, in the form (* * comment ...), the word comment is not treated as a filevar.

Defining New File Manager Commands

A file manager command is defined by specifying the values of certain properties. You can specify the various attributes of a file manager command for a new command, or respecify them for an existing command. The following properties are used:

MACRO

[File Manager Command Property]

Defines how to dump the file manager command. Used by MAKEFILE. Value is a pair (ARGS . COMS). The "arguments" to the file manager command are substituted for ARGS throughout COMS, and the result treated as a list of file manager commands. For example, following (FILEPKGCOM 'FOO 'MACRO '((X Y) . COMS)), the file manager command (FOO A B) will cause A to be substituted for X and B for Y throughout COMS, and then COMS treated as a list of commands.

The substitution is carried out by SUBPAIR (see Chapter 3), so that the "argument list" for the macro can also be atomic. For example, if (X . COMS) was used instead of ((X Y)

INTERLISP-D REFERENCE MANUAL

. *COMS*), then the command (FOO A B) would cause (A B) to be substituted for X throughout *COMS*.

Filevars are evaluated *before* substitution. For example, if the symbol * follows *NAME* in the command, CADDR of the command is evaluated substituting in *COMS*.

ADD

[File Manager Command Property]

Specifies how (if possible) to add an instance of an object of a particular type to a given file manager command. Used by ADDTOFILE. Value is *FN*, a function of three arguments, *COM*, a file manager command CAR of which is EQ to *COMMANDNAME*, *NAME*, a typed object, and *TYPE*, its type. *FN* should return T if it (undoably) adds *NAME* to *COM*, NIL if not. If no ADD property is specified, then the default is (1) if (CAR *COM*) = *TYPE* and (CADR *COM*) = *, and (CADDR *COM*) is a filevar (i.e. a literal atom), add *NAME* to the value of the filevar, or (2) if (CAR *COM*) = *TYPE* and (CADR *COM*) is not *, add *NAME* to (CDR *COM*).

Actually, the function is given a fourth argument, *NEAR*, which if non-NIL, means the function should try to add the item after *NEAR*. See discussion of ADDTOFILES?, in the Storing Files section.

DELETE

[File Manager Command Property]

Specifies how (if possible) to delete an instance of an object of a particular type from a given file manager command. Used by DELFROMFILES. Value is *FN*, a function of three arguments, *COM*, *NAME*, and *TYPE*, same as for ADD. *FN* should return T if it (undoably) deletes *NAME* from *COM*, NIL if not. If no DELETE property is specified, then the default is either (CAR *COM*) = *TYPE* and (CADR *COM*) = *, and (CADDR *COM*) is a filevar (i.e. a literal atom), and *NAME* is contained in the value of the filevar, then remove *NAME* from the filevar, or if (CAR *COM*) = *TYPE* and (CADR *COM*) is not *, and *NAME* is contained in (CDR *COM*), then remove *NAME* from (CDR *COM*).

If *FN* returns the value of ALL, it means that the command is now "empty", and can be deleted entirely from the command list.

CONTENTS

[File Manager Command Property]

CONTAIN

[File Manager Command Property]

Determines whether an instance of an object of a given type is contained in a given file manager command. Used by WHEREIS and INFILECOMS?. Value is *FN*, a function of three arguments, *COM*, a file manager command CAR of which is EQ to *COMMANDNAME*, *NAME*, and *TYPE*. The interpretation of *NAME* is as follows: if *NAME* is NIL, *FN* should return a list of elements of type *TYPE* contained in *COM*. If *NAME* is T, *FN* should return T if there are any elements of type *TYPE* in *COM*. If *NAME* is an atom other than T or NIL, return T if *NAME* of type *TYPE* is contained in *COM*. Finally, if *NAME* is a list, return a list of those elements of type *TYPE* contained in *COM* that are also contained in *NAME*.

It is sufficient for the CONTENTS function to simply return the list of items of type *TYPE* in command *COM*, i.e. it can in fact ignore the *NAME* argument. The *NAME* argument is supplied mainly for those situations where producing the entire list of items involves

significantly more computation or creates more storage than simply determining whether a particular item (or any item) of type *TYPE* is contained in the command.

If a *CONTENTS* property is specified and the corresponding function application returns *NIL* and *(CAR COM) = TYPE*, then the operation indicated by *NAME* is performed on the value of *(CADDR COM)*, if *(CADR COM) = **, otherwise on *(CDR COM)*. In other words, by specifying a *CONTENTS* property that returns *NIL*, e.g. the function *NILL*, you specify that a file manager command of name *FOO* produces objects of file manager type *FOO* and only objects of type *FOO*.

If the *CONTENTS* property is not provided, the command is simply expanded according to its *MACRO* definition, and each command on the resulting command list is then interrogated.

If *COMMANDNAME* is a file manager command that is used frequently, its expansion by the various parts of the system that need to interrogate files can result in a large number of *CONSES* and garbage collections. By informing the file manager as to what this command actually does and does not produce via the *CONTENTS* property, this expansion is avoided. For example, suppose you have a file manager command called *GRAMMARS* which dumps various property lists but no functions. The file manager could ignore this command when seeking information about *FNS*.

The function *FILEPKGCOM* is used to define new file manager commands, or to change the properties of existing commands. It is possible to redefine the attributes of system file manager commands, such as *FNS* or *PROPS*, and to cause unpredictable results.

(FILEPKGCOM COMMANDNAME PROP VAL ... PROP VAL) [NoSpread Function]

Nospread function for defining new file manager commands, or changing properties of existing file manager commands. *PROP* is one of the property names described above; *VAL* is the value to be given that property of the file manager command *COMMANDNAME*. Returns *COMMANDNAME*.

(FILEPKGCOM COMMANDNAME PROP) returns the value of the property *PROP*, without changing it.

(FILEPKGCOM COMMANDNAME) returns a property list of all of the defined properties of *COMMANDNAME*, using the property names as keys.

Specifying *TYPE* as the symbol *COM* can be used to define one file manager command as a synonym of another. For example, *(FILEPKGCOM 'INITVARIABLES 'COM 'INITVARS)* defines *INITVARIABLES* as a synonym for the file manager command *INITVARS*.

Functions for Manipulating File Command Lists

The following functions may be used to manipulate filecoms. The argument *COMS* does *not* have to correspond to the filecoms for some file. For example, *COMS* can be the list of commands generated as a result of expanding a user-defined file manager command.

The following functions will accept a file manager command as a valid value for their *TYPE* argument, even if it does not have a corresponding file manager type. User-defined file manager commands are expanded as necessary.

(**INFILECOMS?** *NAME TYPE COMS*) [Function]

COMS is a list of file manager commands, or a variable whose value is a list of file manager commands. *TYPE* is a file manager type. **INFILECOMS?** returns T if *NAME* of type *TYPE* is "contained" in *COMS*.

If *NAME* = NIL, **INFILECOMS?** returns a list of all elements of type *TYPE*.

If *NAME* = T, **INFILECOMS?** returns T if there are *any* elements of type *TYPE* in *COMS*.

(**ADDTOFILE** *NAME TYPE FILE NEAR LISTNAME*) [Function]

Adds *NAME* of type *TYPE* to the file manager commands for *FILE*. If *NEAR* is given and it is the name of an item of type *TYPE* already on *FILE*, then *NAME* is added to the command that dumps *NEAR*. If *LISTNAME* is given and is the name of a list of items of *TYPE* items on *FILE*, then *NAME* is added to that list. Uses **ADDTOCOMS** and **MAKENEWCOM**. Returns *FILE*. **ADDTOFILE** is undoable.

(**DELFROMFILES** *NAME TYPE FILES*) [Function]

Deletes all instances of *NAME* of type *TYPE* from the filecoms for each of the files on *FILES*. If *FILES* is a non-NIL symbol, (**LIST** *FILES*) is used. *FILES* = NIL defaults to **FILELST**. Returns a list of files from which *NAME* was actually removed. Uses **DELFROMCOMS**. **DELFROMFILES** is undoable.

Deleting a function will also remove the function from any **BLOCKS** declarations in the filecoms.

(**ADDTOCOMS** *COMS NAME TYPE NEAR LISTNAME*) [Function]

Adds *NAME* as a *TYPE* to *COMS*, a list of file manager commands or a variable whose value is a list of file manager commands. Returns NIL if **ADDTOCOMS** was unable to find a command appropriate for adding *NAME* to *COMS*. *NEAR* and *LISTNAME* are described in the discussion of **ADDTOFILE**. **ADDTOCOMS** is undoable.

The exact algorithm for adding commands depends the particular command itself. See discussion of the **ADD** property, in the description of **FILEPKGCOM**.

ADDTOCOMS will not attempt to add an item to any command which is inside of a DECLARE: unless you specified a specific name via the LISTNAME or NEAR option of ADDTOFILES?.

(**DELFROMCOMS** COMS NAME TYPE) [Function]

Deletes NAME as a TYPE from COMS. Returns NIL if DELFROMCOMS was unable to modify COMS to delete NAME. DELFROMCOMS is undoable.

(**MAKENEWCOM** NAME TYPE) [Function]

Returns a file manager command for dumping NAME of type TYPE. Uses the procedure described in the discussion of NEWCOM, in the Defining New File Manager Types section.

(**MOVETOFILE** TOFILE NAME TYPE FROMFILE) [Function]

Moves the definition of NAME as a TYPE from FROMFILE to TOFILE by modifying the file commands in the appropriate way (with DELFROMFILES and ADDTOFILE).

Note that if FROMFILE is specified, the definition will be retrieved from that file, even if there is another definition currently in your environment.

(**FILECOMSLST** FILE TYPE) [Function]

Returns a list of all objects of type TYPE in FILE.

(**FILEFNSLST** FILE) [Function]

Same as (FILECOMSLST FILE 'FNS).

(**FILECOMS** FILE TYPE) [Function]

Returns (PACK* FILE (OR TYPE 'COMS)). Note that (FILECOMS 'FOO) returns the symbol FOOCOMS, not the value of FOOCOMS.

(**SMASHFILECOMS** FILE) [Function]

Maps down (FILECOMSLST FILE 'FILEVARS) and sets to NOBIND all filevars (see the FileVars section), i.e., any variable used in a command of the form (COMMAND * VARIABLE). Also sets (FILECOMS FILE) to NOBIND. Returns FILE.

Symbolic File Format

The file manager manipulates symbolic files in a particular format. This format is defined so that the information in the file is easily readable when the file is listed, as well as being easily manipulated by the file manager functions. In general, there is no reason for you to manually change the contents of a symbolic file. However, to allow you to extend the file manager, this section describes some of the functions used to write symbolic files, and other matters related to their format.

INTERLISP-D REFERENCE MANUAL

(**PRETTYDEF** *PRTTYFNS PRTTYFILE PRTTYCOMS REPRINTFNS SOURCEFILE CHANGES*) [Function]

Writes a symbolic file in PRETTYPRINT format for loading, using *FILERDTBL* as its read table. **PRETTYDEF** returns the name of the symbolic file that was created.

PRETTYDEF operates under a **RESETLST** (see Chapter 14), so if an error occurs, or a Control-D is typed, all files that **PRETTYDEF** has opened will be closed, the (partially complete) file being written will be deleted, and any undoable operations executed will be undone. The **RESETLST** also means that any **RESETSAVES** executed in the file manager commands will also be protected.

PRTTYFNS is an optional list of function names. It is equivalent to including (*FNS * PRTTYFNS*) in the file manager commands in *PRTTYCOMS*. *PRTTYFNS* is an anachronism from when **PRETTYDEF** did not use a list of file manager commands, and should be specified as *NIL*.

PRTTYFILE is the name of the file on which the output is to be written. *PRTTYFILE* has to be a symbol. If *PRTTYFILE* = *NIL*, the primary output file is used. *PRTTYFILE* is opened if not already open, and it becomes the primary output file. *PRTTYFILE* is closed at end of **PRETTYDEF**, and the primary output file is restored.

PRTTYCOMS is a list of file manager commands interpreted as described in the File Manager Commands section. If *PRTTYCOMS* is atomic, its top level value is used and an *RPAQQ* is written which will set that atom to the list of commands when the file is subsequently loaded. A **PRETTYCOMPRINT** expression (see below) will also be written which informs you of the named atom or list of commands when the file is subsequently loaded. In addition, if any of the functions in the file are *nlambda* functions, **PRETTYDEF** will automatically print a **DECLARE:** expression suitable for informing the compiler about these functions, in case you recompile the file without having first loaded the *nlambda* functions (see Chapter 18).

REPRINTFNS and *SOURCEFILE* are for use in conjunction with remaking a file (see the Remaking a Symbolic File section). *REPRINTFNS* can be a list of functions to be prettyprinted, or *EXPRS*, meaning prettyprint all functions with *EXPR* definitions, or *ALL* meaning prettyprint all functions either defined as *EXPRS*, or with *EXPR* properties. Note that doing a remake with *REPRINTFNS* = *NIL* makes sense if there have been changes in the file, but not to any of the functions, e.g., changes to variables or property lists. *SOURCEFILE* is the name of the file from which to copy the definitions for those functions that are *not* going to be prettyprinted, i.e., those not specified by *REPRINTFNS*. *SOURCEFILE* = *T* means to use most recent version (i.e., highest number) of *PRTTYFILE*, the second argument to **PRETTYDEF**. If *SOURCEFILE* cannot be found, **PRETTYDEF** prints the message "FILE NOT FOUND, SO IT WILL BE WRITTEN ANEW", and proceeds as it does when *REPRINTFNS* and *SOURCEFILE* are both *NIL*.

PRETTYDEF calls **PRETTYPRINT** with its second argument *PRETTYDEFLG* = *T*, so whenever **PRETTYPRINT** starts a new function, it prints (on the terminal) the name of that function if more than 30 seconds (real time) have elapsed since the last time it printed the name of a function.

Note that normally if PRETTYPRINT is given a symbol which is not defined as a function but is known to be on one of the files noticed by the file manager, PRETTYPRINT will load in the definition (using LOADFNS) and print it. This is not done when PRETTYPRINT is called from PRETTYDEF.

In Medley the SYSPRETTYFLG is ignored in the Interlisp exec.

(**PRINTFNS** *X*) [Function]

X is a list of functions. PRINTFNS prettyprints a DEFINEQ expression that defines the functions to the primary output stream using the primary read table. Used by PRETTYDEF to implement the FNS file manager command.

(**PRINTDATE** *FILE CHANGES*) [Function]

Prints the FILECREATED expression at beginning of PRETTYDEF files. *CHANGES* used by the file manager.

(**FILECREATED** *X*) [NLambda NoSpread Function]

Prints a message (using LISPXPRINT) followed by the time and date the file was made, which is (CAR *X*). The message is the value of PRETTYHEADER, initially "FILE CREATED". If PRETTYHEADER = NIL, nothing is printed. (CDR *X*) contains information about the file, e.g., full name, address of file map, list of changed items, etc. FILECREATED also stores the time and date the file was made on the property list of the file under the property FILEDATES and performs other initialization for the file manager.

(**PRETTYCOMPRINT** *X*) [NLambda Function]

Prints *X* (unevaluated) using LISPXPRINT, unless PRETTYHEADER = NIL.

PRETTYHEADER [Variable]

Value is the message printed by FILECREATED. PRETTYHEADER is initially "FILE CREATED". If PRETTYHEADER = NIL, neither FILECREATED nor PRETTYCOMPRINT will print anything. Thus, setting PRETTYHEADER to NIL will result in "silent loads". PRETTYHEADER is reset to NIL during greeting (see Chapter 12).

(**FILECHANGES** *FILE TYPE*) [Function]

Returns a list of the changed objects of file manager type *TYPE* from the FILECREATED expression of *FILE*. If *TYPE* = NIL, returns an alist of all of the changes, with the file manager types as the CARs of the elements..

(**FILEDATE** *FILE*) [Function]

Returns the file date contained in the FILECREATED expression of *FILE*.

(**LISPSOURCEFILEP** *FILE*) [Function]

Returns a non-NIL value if *FILE* is in file manager format and has a file map, NIL otherwise.

INTERLISP-D REFERENCE MANUAL

Copyright Notices

The system has a facility for automatically printing a copyright notice near the front of files, right after the FILECREATED expression, specifying the years it was edited and the copyright owner. The format of the copyright notice is:

```
(* Copyright (c) 1981 by Foo Bars Corporation)
```

Once a file has a copyright notice then every version will have a new copyright notice inserted into the file without your intervention. (The copyright information necessary to keep the copyright up to date is stored at the end of the file.).

Any year the file has been edited is considered a "copyright year" and therefore kept with the copyright information. For example, if a file has been edited in 1981, 1982, and 1984, then the copyright notice would look like:

```
(* Copyright (c) 1981,1982,1984 by Foo Bars Corporation)
```

When a file is made, if it has no copyright information, the system will ask you to specify the copyright owner (if COPYRIGHTFLG = T). You may specify one of the names from COPYRIGHTOWNERS, or give one of the following responses:

- Type a left-square-bracket. The system will then prompt for an arbitrary string which will be used as the owner-string
- Type a right-square-bracket, which specifies that you really do not want a copyright notice.
- Type "NONE" which specifies that this file should never have a copyright notice.

For example, if COPYRIGHTOWNERS has the value

```
((BBN "Bolt Beranek and Newman Inc.")  
(XEROX "Xerox Corporation"))
```

then for a new file FOO the following interaction will take place:

```
Do you want to Copyright FOO? Yes  
Copyright owner: (user typed ?)  
one of:  
BBN - Bolt Beranek and Newman Inc.  
XEROX - Xerox Corporation  
NONE - no copyright ever for this file  
[ - new copyright owner -- type one line of text  
] - no copyright notice for this file now  
  
Copyright owner: BBN
```

Then "Foo Bars Corporation" in the above copyright notice example would have been "Bolt Beranek and Newman Inc."

The following variables control the operation of the copyright facility:

COPYRIGHTFLG [Variable]

The value of COPYRIGHTFLG determines whether copyright information is maintained in files. Its value is interpreted as follows:

- NIL** The system will preserve old copyright information, but will not ask you about copyrighting new files. This is the default value of COPYRIGHTFLG.
- T** When a file is made, if it has no copyright information, the system will ask you to specify the copyright owner.
- NEVER** The system will neither prompt for new copyright information nor preserve old copyright information.
- DEFAULT** The value of DEFAULTCOPYRIGHTOWNER (below) is used for putting copyright information in files that don't have any other copyright. The prompt "Copyright owner for file xx:" will still be printed, but the default will be filled in immediately.

COPYRIGHTOWNERS [Variable]

COPYRIGHTOWNERS is a list of entries of the form (*KEY OWNERSTRING*), where *KEY* is used as a response to ASKUSER and *OWNERSTRING* is a string which is the full identification of the owner.

DEFAULTCOPYRIGHTOWNER [Variable]

If you do not respond in DWIMWAIT seconds to the copyright query, the value of DEFAULTCOPYRIGHTOWNER is used.

Functions Used Within Source Files

The following functions are normally only used within symbolic files, to set variable values, property values, etc. Most of these have special behavior depending on file manager variables.

(**RPAQ** *VAR VALUE*) [NLambda Function]

An nlambda function like SETQ that sets the top level binding of *VAR* (unevaluated) to *VALUE*.

INTERLISP-D REFERENCE MANUAL

(**RPAQQ** *VAR VALUE*) [NLambda Function]

An nlambda function like **SETQQ** that sets the top level binding of *VAR* (unevaluated) to *VALUE* (unevaluated).

(**RPAQ?** *VAR VALUE*) [NLambda Function]

Similar to **RPAQ**, except that it does nothing if *VAR* already has a top level value other than **NOBIND**. Returns *VALUE* if *VAR* is reset, otherwise **NIL**.

RPAQ, **RPAQQ**, and **RPAQ?** generate errors if *X* is not a symbol. All are affected by the value of **DFNFLG** (see Chapter 10). If **DFNFLG** = **ALLPROP** (and the value of *VAR* is other than **NOBIND**), instead of setting *X*, the corresponding value is stored on the property list of *VAR* under the property **VALUE**. All are undoable.

(**ADDTOVAR** *VAR X X ... X*) [NLambda NoSpread Function]

Each *X* that is not a member of the value of *VAR* is added to it, i.e. after **ADDTOVAR** completes, the value of *VAR* will be (**UNION** (**LIST** *X X ... X*) *VAR*). **ADDTOVAR** is used by **PRETTYDEF** for implementing the **ADDVARS** command. It performs some file manager related operations, i.e. "notices" that *VAR* has been changed. Returns the atom *VAR* (not the value of *VAR*).

(**APPENDTOVAR** *VAR X X ... X*) [NLambda NoSpread Function]

Similar to **ADDTOVAR**, except that the values are added to the end of the list, rather than at the beginning.

(**PUTPROPS** *ATM PROP VAL ... PROP VAL*) [NLambda NoSpread Function]

Nlambda nospread version of **PUTPROP** (none of the arguments are evaluated). For $i = 1 \dots N$, puts property *PROP*, value *VAL_i*, on the property list of *ATM*. Performs some file manager related operations, i.e., "notices" that the corresponding properties have been changed.

(**SAVEPUT** *ATM PROP VAL*) [Function]

Same as **PUTPROP**, but marks the corresponding property value as having been changed (used by the file manager).

File Maps

A file map is a data structure which contains a symbolic 'map' of the contents of a file. Currently, this consists of the begin and end byte address (see **GETFILEPTR**, in Chapter 25) for each **DEFINEQ** expression in the file, the begin and end address for each function definition within the **DEFINEQ**, and the begin and end address for each compiled function.

MAKEFILE, **PRETTYDEF**, **LOADFNS**, **RECOMPILE**, and numerous other system functions depend heavily on the file map for efficient operation. For example, the file map enables **LOADFNS** to load

selected function definitions simply by setting the file pointer to the corresponding address using `SETFILEPTR`, and then performing a single `READ`. Similarly, the file map is heavily used by the "remake" option of `MAKEFILE` (see the *Remaking a Symbolic File* section): those function definitions that have been changed since the previous version are prettyprinted; the rest are simply copied from the old file to the new one, resulting in a considerable speedup.

Whenever a file is written by `MAKEFILE`, a file map for the new file is built. Building the map in this case essentially comes for free, since it requires only reading the current file pointer before and after each definition is written or copied. However, building the map does require that `PRETTYPRINT` know that it is printing a `DEFINEQ` expression. For this reason, you should never print a `DEFINEQ` expression onto a file yourself, but should instead always use the `FNS` file manager command (see the *Functions and Macros* section).

The file map is stored on the property list of the root name of the file, under the property `FILEMAP`. In addition, `MAKEFILE` writes the file map on the file itself. For cosmetic reasons, the file map is written as the last expression in the file. However, the *address* of the file map in the file is (over)written into the `FILECREATED` expression that appears at the beginning of the file so that the file map can be rapidly accessed without having to scan the entire file. In most cases, `LOAD` and `LOADFNS` do not have to build the file map at all, since a file map will usually appear in the corresponding file, unless the file was written with `BUILDMAPFLG = NIL`, or was written outside of Interlisp.

Currently, file maps for *compiled* files are not written onto the files themselves. However, `LOAD` and `LOADFNS` will build maps for a compiled file when it is loaded, and store it on the property `FILEMAP`. Similarly, `LOADFNS` will obtain and use the file map for a compiled file, when available.

The use and creation of file maps is controlled by the following variables:

BUILDMAPFLG

[Variable]

Whenever a file is read by `LOAD` or `LOADFNS`, or written by `MAKEFILE`, a file map is automatically built unless `BUILDMAPFLG = NIL`. (`BUILDMAPFLG` is initially `T`.)

While building the map will not help the first reference to a file, it will help in future references. For example, if you perform `(LOADFROM 'FOO)` where `FOO` does not contain a file map, the `LOADFROM` will be (slightly) slower than if `FOO` did contain a file map, but subsequent calls to `LOADFNS` for this version of `FOO` will be able to use the map that was built as the result of the `LOADFROM`, since it will be stored on `FOO`'s `FILEMAP` property.

USEMAPFLG

[Variable]

If `USEMAPFLG = T` (the initial setting), the functions that use file maps will first check the `FILEMAP` property to see if a file map for this file was previously obtained or built. If not, the first expression on the file is checked to see if it is a `FILECREATED` expression that also contains the address of a file map. If the file map is not on the `FILEMAP` property or in the file, a file map will be built (unless `BUILDMAPFLG = NIL`).

INTERLISP-D REFERENCE MANUAL

If `USEMAPFLG = NIL`, the `FILEMAP` property and the file will not be checked for the file map. This allows you to recover in those cases where the file and its map for some reason do not agree. For example, if you use a text editor to change a symbolic file that contains a map (not recommended), inserting or deleting just one character will throw that map off. The functions which use file maps contain various integrity checks to enable them to detect that something is wrong, and to generate the error `FILEMAP DOES NOT AGREE WITH CONTENTS OF FILE`. In such cases, you can set `USEMAPFLG` to `NIL`, causing the map contained in the file to be ignored, and then reexecute the operation.

18. COMPILER

The compiler is contained in the standard Medley system. It may be used to compile functions defined in Medley, or to compile definitions stored in a file. The resulting compiled code may be stored as it is compiled, so as to be available for immediate use, or it may be written onto a file for subsequent loading.

The most common way to use the compiler is to use one of the file package functions, such as `MAKEFILE` (Chapter 17), which automatically updates source files, and produces compiled versions. However, it is also possible to compile individual functions defined in Medley, by directly calling the compiler using functions such as `COMPILE`. No matter how the compiler is called, the function `COMPSET` is called which asks you certain questions concerning the compilation. (`COMPSET` sets the free variables `LAPFLG`, `STRF`, `SVFLG`, `LCFIL` and `LSTFIL` which determine various modes of operation.) Those that can be answered "yes" or "no" can be answered with `YES`, `Y`, or `T` for "yes"; and `NO`, `N`, or `NIL` for "no". The questions are:

LISTING? This asks whether to generate a listing of the compiled code. The LAP and machine code are usually not of interest but can be helpful in debugging macros. Possible answers are:

- 1 Prints output of pass 1, the LAP macro code
- 2 Prints output of pass 2, the machine code
- YES** Prints output of both passes
- NO** Prints no listings

The variable `LAPFLG` is set to the answer.

FILE: This question (which only appears if the answer to **LISTING?** is affirmative) ask where the compiled code listing(s) should be written. Answering `T` will print the listings at the terminal. The variable `LSTFIL` is set to the answer.

REDEFINE? This question asks whether the functions compiled should be redefined to their compiled definitions. If this is answered `YES`, the compiled code is stored and the function definition changed, otherwise the function definition remains unchanged.

The compiler does *not* respect the value of `DFNFLG` (Chapter 10) when it redefines functions to their compiled definitions. Therefore, if you set `DFNFLG` to `PROP` to completely avoid inadvertently redefining something in your running system, you *must* not answer `YES` to this question.

The variable `STRF` is set to `T` (if this is answered `YES`) or `NIL`.

INTERLISP-D REFERENCE MANUAL

SAVE EXPRS? This question asks whether the original defining EXPRS of functions should be saved. If answered YES, then before redefining a function to its compiled definition, the EXPR definition is saved on the property list of the function name. Otherwise they are discarded.

It is very useful to save the EXPR definitions, just in case the compiled function needs to be changed. The editing functions will retrieve this saved definition if it exists, rather than reading from a source file.

The variable SVFLG is set to T (if this is answered YES) or NIL.

OUTPUT FILE? This question asks whether (and where) the compiled definitions should be written into a file for later loading. If you answer with the name of a file, that file will be used. If you answer Y or YES, you will be asked the name of the file. If the file named is already open, it will continue to be used. If you answer T or TTY:, the output will be typed on the teletype (not particularly useful). If you answer N, NO, or NIL, output will *not* be done.

The variable LCFIL is set to the name of the file.

To make answering these questions easier, there are four other possible answers to the LISTING? question, which specify common compiling modes:

- S** Same as last setting. Uses the same answers to compiler questions as given for the last compilation.
- F** Compile to **F**ile, without redefining functions.
- ST** **S**Tore new definitions, saving EXPR definitions.
- STF** **S**Tore new definitions; **F**orget EXPR definitions.

Implicit in these answers are the answers to the questions on disposition of compiled code and EXPR definitions, so the questions REDEFINE? and SAVE EXPRS? would not be asked if these answers were given. OUTPUT FILE? would still be asked, however. For example:

```
←COMPILE((FACT FACT1 FACT2))
LISTING? ST
OUTPUT FILE? FACT.DCOM
(FACT COMPILING)
.
.
(FACT REDEFINED)
.
.
(FACT2 REDEFINED)
(FACT FACT1 FACT2)
←
```

This process caused the functions `FACT`, `FACT1`, and `FACT2` to be compiled, redefined, and the compiled definitions also written on the file `FACT.DCOM` for subsequent loading.

Compiler Printout

In Medley, for each function *FN* compiled, whether by `TCOMPL`, `RECOMPILE`, or `COMPILE`, the compiler prints:

```
(FN (ARG ... ARG) (uses: VAR ... VAR) (calls: FN ... FN))
```

The message is printed at the beginning of the second pass of the compilation of *FN*. (*ARG ... ARG*) is the list of arguments to *FN*; following *uses:* are the free variables referenced or set in *FN* (not including global variables); following *calls:* are the undefined functions called within *FN*.

If the compilation of *FN* causes the generation of one or more auxiliary functions, a compiler message will be printed for these functions before the message for *FN*, e.g.,

```
(FOOA0027 (X) (uses: XX))  
(FOO (A B))
```

When compiling a block, the compiler first prints (*BLKNAME BLKFN BLKFN ...*). Then the normal message is printed for the entire block. The names of the arguments to the block are generated by suffixing # and a number to the block name, e.g., (`FOOBLOCK (FOOBLOCK#0 FOOBLOCK#1) FREE-VARIABLES`). Then a message is printed for each *entry* to the block.

In addition to the above output, both `RECOMPILE` and `BRECOMPILE` print the name of each function that is being copied from the old compiled file to the new compiled file. The normal compiler message is printed for each function that is actually compiled.

The compiler prints out error messages when it encounters problems compiling a function. For example:

```
----- In BAZ:  
***** (BAZ - illegal RETURN)  
-----
```

The above error message indicates that an `illegal RETURN` compiler error occurred while trying to compile the function `BAZ`. Some compiler errors cause the compilation to terminate, producing nothing; however, there are other compiler errors which do not stop compilation. The compiler error messages are described in the last section of this chapter.

Compiler printout and error messages go to the file `COUTFILE`, initially `T`. `COUTFILE` can also be set to the name of a file opened for output, in which case all compiler printout will go to `COUTFILE`, i.e. the compiler will compile "silently." However, any error messages will be printed to both `COUTFILE` as well as `T`.

INTERLISP-D REFERENCE MANUAL

Global Variables

Variables that appear on the list GLOBALVARS, or have the property GLOBALVAR with value T, or are declared with the GLOBALVARS file package command, are called global variables. Such variables are always accessed through their top level value when they are used freely in a compiled function. In other words, a reference to the value of a global variable is equivalent to calling GETTOPVAL on the variable, regardless of whether or not it is bound in the current access chain. Similarly, (SETQ VARIABLE VALUE) will compile as (SETTOPVAL (QUOTE VARIABLE) VALUE).

All system parameters, unless otherwise specified, are declared as global variables. Thus, *rebinding* these variables in a deep bound system like Medley will not affect the behavior of the system: instead, the variables must be *reset* to their new values, and if they are to be restored to their original values, reset again. For example, you might write

```
(SETQ GLOBALVARIABLE NEWVALUE)
FORM
(SETQ GLOBALVARIABLE OLDVALUE)
```

In this case, if an error occurred during the evaluation of FORM, or a Control-D was typed, the global variable would not be restored to its original value. The function RESETVAR provides a convenient way of resetting global variables in such a way that their values are restored even if an error occurred or Control-D is typed.

Note: The variables that a given function accesses as global variables can be determined by using the function CALLS.

Local Variables and Special Variables

In normal compiled and interpreted code, all variable bindings are accessible by lower level functions because the variable's name is associated with its value. We call such variables *special* variables, or specvars. As mentioned earlier, the block compiler normally does *not* associate names with variable values. Such unnamed variables are not accessible from outside the function which binds them and are therefore *local* to that function. We call such unnamed variables local variables, or localvars.

The time economies of local variables can be achieved without block compiling by use of declarations. Using local variables will increase the speed of compiled code; the price is the work of writing the necessary specvar declarations for those variables which need to be accessed from outside the block.

LOCALVARS and SPECVARS are variables that affect compilation. During regular compilation, SPECVARS is normally T, and LOCALVARS is NIL or a list. This configuration causes all variables bound in the functions being compiled to be treated as special *except* those that appear on LOCALVARS. During block compilation, LOCALVARS is normally T and SPECVARS is NIL or a list. All variables are then treated as local *except* those that appear on SPECVARS.

Declarations to set `LOCALVARS` and `SPECVARS` to other values, and therefore affect how variables are treated, may be used at several levels in the compilation process with varying scope.

1. The declarations may be included in the filecoms of a file, by using the `LOCALVARS` and `SPECVARS` file package commands. The scope of the declaration is then the entire file:

```
... (LOCALVARS . T) (SPECVARS X Y) ...
```

2. The declarations may be included in block declarations; the scope is then the block, e.g.,

```
(BLOCKS ((FOOBLOCK FOO FIE (SPECVARS . T) (LOCALVARS
X)))
```

3. The declarations may also appear in individual functions, or in `PROG`'s or `LAMBDA`'s within a function, using the `DECLARE` function. In this case, the scope of the declaration is the function or the `PROG` or `LAMBDA` in which it appears. `LOCALVARS` and `SPECVARS` declarations must appear immediately after the variable list in the function, `PROG`, or `LAMBDA`, but intervening comments are permitted. For example:

```
(DEFINEQ ((FOO
  (LAMBDA (X Y)
    (DECLARE (LOCALVARS Y))
    (PROG (X Y Z)
      (DECLARE (LOCALVARS X))
      ... ]
```

If the above function is compiled (non-block), the outer `X` will be special, the `X` bound in the `PROG` will be local, and both bindings of `Y` will be local.

Declarations for `LOCALVARS` and `SPECVARS` can be used in two ways: either to cause variables to be treated the same whether the function(s) are block compiled or compiled normally, or to affect one compilation mode while not affecting the default in the other mode. For example:

```
(LAMBDA (X Y)
  (DECLARE (SPECVARS . T))
  (PROG (Z) ... ]
```

will cause `X`, `Y`, and `Z` to be specvars for both block and normal compilation while

```
(LAMBDA (X Y)
  (DECLARE (SPECVARS X))
  ... ]
```

will make `X` a specvar when block compiling, but when regular compiling the declaration will have no effect, because the default value of specvars would be `T`, and therefore *both* `X` and `Y` will be specvars by default.

Although `LOCALVARS` and `SPECVARS` declarations have the same form as other components of block declarations such as `(LINKFNS . T)`, their operation is somewhat different because the two

INTERLISP-D REFERENCE MANUAL

variables are not independent. `(SPECVARS . T)` will cause `SPECVARS` to be set to `T`, and `LOCALVARS` to be set to `NIL`. `(SPECVARS V1 V2 ...)` will have *no* effect if the value of `SPECVARS` is `T`, but if it is a list (or `NIL`), `SPECVARS` will be set to the union of its prior value and `(V1 V2 ...)`. The operation of `LOCALVARS` is analogous. Thus, to affect both modes of compilation one of the two (`LOCALVARS` or `SPECVARS`) must be declared `T` before specifying a list for the other.

Note: The variables that a given function binds as local variables or accesses as special variables can be determined by using the function `CALLS`.

Note: `LOCALVARS` and `SPECVARS` declarations affect the compilation of local variables within a function, but the arguments to functions are always accessible as `specvars`. This can be changed by redefining the following function:

(DASSEM.SAVELOCALVARS FN) [Function]

This function is called by the compiler to determine whether argument information for *FN* should be written on the compiled file for *FN*. If it returns `NIL`, the argument information is *not* saved, and the function is stored with arguments `U`, `V`, `W`, etc instead of the originals.

Initially, `DASSEM.SAVELOCALVARS` is defined to return `T`. `(MOVD 'NIL 'DASSEM.SAVELOCALVARS)` causes the compiler to retain no local variable or argument names. Alternatively, `DASSEM.SAVELOCALVARS` could be redefined as a more complex predicate, to allow finer discrimination.

Constants

Interlisp allows the expression of constructions which are intended to be description of their constant values. The following functions are used to define constant values. The function `SELECTC` provides a mechanism for comparing a value to a number of constants.

(CONSTANT X) [Function]

This function enables you to define that the expression *X* should be treated as a "constant" value. When `CONSTANT` is interpreted, *X* is evaluated each time it is encountered. If the `CONSTANT` form is compiled, however, the expression will be evaluated only once.

If the value of *X* has a readable print name, then it will be evaluated at compile-time, and the value will be saved as a literal in the compiled function's definition, as if `(QUOTE VALUE-OF-EXPRESSION)` had appeared instead of `(CONSTANT EXPRESSION)`.

If the value of *X* does not have a readable print name, then the expression *X* itself will be saved with the function, and it will be evaluated when the function is first loaded. The value will then be stored in the function's literals, and will be retrieved on future references.

If a program needed a list of 30 `NIL`s, you could specify `(CONSTANT (to 30 collect NIL))` instead of `(QUOTE (NIL NIL ...))`. The former is more concise and displays the important parameter much more directly than the latter.

CONSTANT can also be used to denote values that cannot be quoted directly, such as (CONSTANT (PACK NIL)), (CONSTANT (ARRAY 10)). It is also useful to parameterize quantities that are constant at run time but may differ at compile time, e.g., (CONSTANT BITSPERWORD) in a program is exactly equivalent to 36, if the variable BITSPERWORD is bound to 36 when the CONSTANT expression is evaluated at compile time.

Whereas the function CONSTANT attempts to evaluate the expression as soon as possible (compile-time, load-time, or first-run-time), other options are available, using the following two function:

(LOADTIMECONSTANT *X*) [Function]

Similar to CONSTANT, except that the evaluation of *X* is deferred until the compiled code for the containing function is loaded in. For example, (LOADTIMECONSTANT (DATE)) will return the date the code was loaded. If LOADTIMECONSTANT is interpreted, it merely returns the value of *X*.

(DEFERREDCONSTANT *X*) [Function]

Similar to CONSTANT, except that the evaluation of *X* is always deferred until the compiled function is first run. This is useful when the storage for the constant is excessive so that it shouldn't be allocated until (unless) the function is actually invoked. If DEFERREDCONSTANT is interpreted, it merely returns the value of *X*.

(CONSTANTS *VAR VAR ... VAR*) [NLambda NoSpread Function]

Defines *VAR* , ... *VAR* (unevaluated) to be compile-time constants. Whenever the compiler encounters a (free) reference to one of these constants, it will compile the form (CONSTANT *VAR*) instead.

If *VAR* is a list of the form (*VAR FORM*) , a free reference to the variable will compile as (CONSTANT *FORM*) .

The compiler prints a warning if user code attempts to bind a variable previously declared as a constant.

Constants can be saved using the CONSTANTS file package command.

Compiling Function Calls

When compiling the call to a function, the compiler must know the type of the function, to determine how the arguments should be prepared (evaluated/unevaluated, spread/nospread). There are three separate cases: lambda, nlambda spread, and nlambda nospread functions.

To determine which of these three cases is appropriate, the compiler will first look for a definition among the functions in the file that is being compiled. The function can be defined anywhere in any of the files given as arguments to BCOMPL, TCOMPL, BRECOMPILE or RECOMPILE. If the function is

INTERLISP-D REFERENCE MANUAL

not contained in the file, the compiler will look for other information in the variables `NLAMA`, `NLAML`, and `LAMS`, which can be set by you:

NLAMA [Variable]

(For `NLAMBda` Atoms) A list of functions to be treated as `nlambda` nospread functions by the compiler.

NLAML [Variable]

(For `NLAMBda` List) A list of functions to be treated as `nlambda` spread functions by the compiler.

LAMS [Variable]

A list of functions to be treated as `lambda` functions by the compiler. Note that including functions on `LAMS` is only necessary to override in-core `nlambda` definitions, since in the absence of other information, the compiler assumes the function is a `lambda`.

If the function is not contained in a file, or on the lists `NLAMA`, `NLAML`, or `LAMS`, the compiler will look for a current definition in the Interlisp system, and use its type. If there is no current definition, next `COMPILEUSERFN` is called:

COMPILEUSERFN [Variable]

When compiling a function call, if the function type cannot be found by looking in files, the variables `NLAMA`, `NLAML`, or `LAMS`, or at a current definition, then if the value of `COMPILEUSERFN` is not `NIL`, the compiler calls (the value of) `COMPILEUSERFN` giving it as arguments `CDR` of the form and the form itself, i.e., the compiler does `(APPLY* COMPILEUSERFN (CDR FORM) FORM)`. If a non-`NIL` value is returned, it is compiled instead of `FORM`. If `NIL` is returned, the compiler compiles the original expression as a call to a `lambda` spread that is not yet defined.

`COMPILEUSERFN` is only called when the compiler encounters a *list* `CAR` of which is not the name of a defined function. You can instruct the compiler about how to compile other data types via `COMPILETYPEELST`.

`CLISP` uses `COMPILEUSERFN` to tell the compiler how to compile iterative statements, `IF-THEN-ELSE` statements, and pattern match constructs.

If the compiler cannot determine the function type by any of the means above, it assumes that the function is a `lambda` function, and its arguments are to be evaluated.

If there are `nlambda` functions called from the functions being compiled, and they are only defined in a separate file, they must be included on `NLAMA` or `NLAML`, or the compiler will incorrectly assume that their arguments are to be evaluated, and compile the calling function correspondingly. This is only necessary if the compiler does not "know" about the function. If the function is defined at compile time, or is handled via a macro, or is contained in the same group of files as the functions that call it, the compiler will automatically handle calls to that function correctly.

FUNCTION and Functional Arguments

Compiling the function `FUNCTION` may involve creating and compiling a separate "auxiliary function", which will be called at run time. An auxiliary function is named by attaching a `GENSYM` to the end of the name of the function in which they appear, e.g., `FOOA0003`. For example, suppose `FOO` is defined as `(LAMBDA (X) ... (FOO1 X (FUNCTION ...)) ...)` and compiled. When `FOO` is run, `FOO1` will be called with two arguments, `X`, and `FOOA000N` and `FOO1` will call `FOOA000N` each time it uses its functional argument.

Compiling `FUNCTION` will *not* create an auxiliary function if it is a functional argument to a function that compiles open, such as most of the mapping functions (`MAPCAR`, `MAPLIST`, etc.). A considerable savings in time could be achieved by making `FOO1` compile open via a computed macro, e.g.

```
(PUTPROP 'FOO1 'MACRO
  '(Z (LIST (SUBST (CADADR Z)
                  (QUOTE FN)
                  DEF)
          (CAR Z))))
```

`DEF` is the definition of `FOO1` as a function of just its first argument, and `FN` is the name used for its functional argument in its definition. In this case, `(FOO1 X (FUNCTION ...))` would compile as an expression, containing the argument to `FUNCTION` as an open `LAMBDA` expression. Thus you save not only the function call to `FOO1`, but also each of the function calls to its functional argument. For example, if `FOO1` operates on a list of length ten, eleven function calls will be saved. Of course, this savings in time costs space, and you must decide which is more important.

Open Functions

When a function is called from a compiled function, a system routine is invoked that sets up the parameter and control push lists as necessary for variable bindings and return information. If the amount of time spent *inside* the function is small, this function calling time will be a significant percentage of the total time required to use the function. Therefore, many "small" functions, e.g., `CAR`, `CDR`, `EQ`, `NOT`, `CONS` are always compiled "open", i.e., they do not result in a function call. Other larger functions such as `PROG`, `SELECTQ`, `MAPC`, etc. are compiled open because they are frequently used. You can make other functions compile open via `MACRO` definitions. You can also affect the compiled code via `COMPILEUSERFN` and `COMPILETYPELIST`.

COMPILETYPELIST

Most of the compiler's mechanism deals with how to handle forms (lists) and variables (symbols). You can affect the compiler's behaviour with respect to lists and literal atoms in a number of ways, e.g. macros, declarations, `COMPILEUSERFN`, etc. `COMPILETYPELIST` allows you to tell the compiler

INTERLISP-D REFERENCE MANUAL

what to do when it encounters a data type *other* than a list or an atom. It is the facility in the compiler that corresponds to DEFEVAL for the interpreter.

COMPILETYPELST

[Variable]

A list of elements of the form (*TYPENAME* . *FUNCTION*). Whenever the compiler encounters a datum that is not a list and not an atom (or a number) in a context where the datum is being evaluated, the type name of the datum is looked up on COMPILETYPELST. If an entry appears CAR of which is equal to the type name, CDR of that entry is applied to the datum. If the value returned by this application is *not* EQ to the datum, then that value is compiled instead. If the value is EQ to the datum, or if there is no entry on COMPILETYPELST for this type name, the compiler simply compiles the datum as (QUOTE *DATUM*) .

Compiling CLISP

Since the compiler does not know about CLISP, in order to compile functions containing CLISP constructs, the definitions must first be DWIMIFYed. You can automate this process in several ways:

1. If the variable DWIMIFYCOMPFLG is T, the compiler will always DWIMIFY expressions before compiling them. DWIMIFYCOMPFLG is initially NIL.
2. If a file has the property FILETYPE with value CLISP on its property list, TCOMPL, BCOMPL, RECOMPILE, and BRECOMPILE will operate as though DWIMIFYCOMPFLG is T and DWIMIFY all expressions before compiling.
3. If the function definition has a local CLISP declaration, including a null declaration, i.e., just (CLISP:), the definition will be automatically DWIMIFYed before compiling.

Note: COMPILEUSERFN is defined to call DWIMIFY on iterative statements, IF-THEN statements, and fetch, replace, and match expressions, i.e., any CLISP construct which can be recognized by its CAR of form. Thus, if the only CLISP constructs in a function appear inside of iterative statements, IF statements, etc., the function does not have to be dwimified before compiling.

If DWIMIFY is ever unsuccessful in processing a CLISP expression, it will print the error message UNABLE TO DWIMIFY followed by the expression, and go into a break unless DWIMESSGAG = T. In this case, the expression is just compiled as is, i.e. as though CLISP had not been enabled. You can exit the break in one of these ways:

1. Type OK to the break, which will cause the compiler to try again, e.g. you could define some missing records while in the break, and then continue
2. Type ↑, which will cause the compiler to simply compile the expression as is, i.e. as though CLISP had not been enabled in the first place
3. Return an expression to be compiled in its place by using the RETURN break command.

Note: TCOMPL, BCOMPL, RECOMPILE, and BRECOMPILE all scan the entire file before doing any compiling, and take note of the names of all functions that are defined in the file as well as the names of all variables that are set by adding them to NOFIXFNSLST and NOFIXVARSLST, respectively. Thus, if a function is not currently defined, but *is* defined in the file being compiled, when DWIMIFY is called before compiling, it will not attempt to interpret the function name as CLISP when it appears as CAR of a form. DWIMIFY also takes into account variables that have been declared to be LOCALVARS, or SPECVARS, either via block declarations or DECLARE expressions in the function being compiled, and does not attempt spelling correction on these variables. The declaration USEDFREE may also be used to declare variables simply used freely in a function. These variables will also be left alone by DWIMIFY. Finally, NOSPELLFLG is reset to T when compiling functions from a file (as opposed to from their in-core definition) so as to suppress spelling correction.

Compiler Functions

Normally, the compiler is invoked through file package commands that keep track of the state of functions, and manage a set of files, such as MAKEFILE. However, it is also possible to explicitly call the compiler using one of a number of functions. Functions may be compiled from in-core definitions (via COMPILE), or from definitions in files (TCOMPL), or from a combination of in-core and file definitions (RECOMPILE).

TCOMPL and RECOMPILE produce "compiled" files. Compiled files usually have the same name as the symbolic file they were made from, suffixed with DCOM (the compiled file extension is stored as the value of the variable COMPILE.EXT). The file name is constructed from the name field only, e.g., (TCOMPL '<BOBROW>FOO.TEM;3) produces FOO.DCOM on the connected directory. The version number will be the standard default.

A "compiled file" contains the same expressions as the original symbolic file, except for the following:

1. A special FILECREATED expression appears at the front of the file which contains information used by the file package, and which causes the message COMPILED ON DATE to be printed when the file is loaded (the actual string printed is the value of COMPILEHEADER).
2. Every DEFINEQ in the symbolic file is replaced by the corresponding compiled definitions in the compiled file.
3. Expressions following a DONTCOPY tag inside of a DECLARE: that appears in the symbolic file are not copied to the compiled file.

The compiled definitions appear at the front of the compiled file, i.e., before the other expressions in the symbolic file, *regardless of where they appear in the symbolic file*. The only exceptions are expressions

INTERLISP-D REFERENCE MANUAL

that follow a `FIRST` tag inside of a `DECLARE`:. This "compiled" file can be loaded into any Interlisp system with `LOAD`.

Note: When a function is compiled from its in-core definition (as opposed to being compiled from a definition in a file), and the function has been modified by `BREAK`, `TRACE`, `BREAKIN`, or `ADVISE`, it is first restored to its original state, and a message is printed out, e.g., `FOO UNBROKEN`. If the function is not defined by an `expr` definition, the value of the function's `EXPR` property is used for the compilation, if there is one. If there is no `EXPR` property, and the compilation is being performed by `RECOMPILE`, the definition of the function is obtained from the file (using `LOADFNS`). Otherwise, the compiler prints `(FN NOT COMPILEABLE)`, and goes on to the next function.

(COMPILE *X* *FLG*) [Function]

X is a list of functions (if atomic, `(LIST X)` is used). `COMPILE` first asks the standard compiler questions, and then compiles each function on *X*, using its in-core definition. Returns *X*.

If compiled definitions are being written to a file, the file is closed unless `FLG = T`.

(COMPILE1 *FN* *DEF*) [Function]

Compiles *DEF*, redefining *FN* if `STRF = T` (`STRF` is one of the variables set by `COMPSET`). `COMPILE1` is used by `COMPILE`, `TCOMPL`, and `RECOMPILE`.

If `DWIMIFYCOMPFLG` is `T`, or *DEF* contains a `CLISP` declaration, *DEF* is dwimified before compiling.

(TCOMPL *FILES*) [Function]

`TCOMPL` is used to "compile files"; given a symbolic `LOAD` file (e.g., one created by `MAKEFILE`), it produces a "compiled file". *FILES* is a list of symbolic files to be compiled (if atomic, `(LIST FILES)` is used). `TCOMPL` asks the standard compiler questions, except for `"OUTPUT FILE: "`. The output from the compilation of each symbolic file is written on a file of the same name suffixed with `DCOM`, e.g., `(TCOMPL ' (SYM1 SYM2) )` produces two files, `SYM1.DCOM` and `SYM2.DCOM`.

`TCOMPL` processes the files one at a time, reading in the entire file. For each `FILECREATED` expression, the list of functions that were marked as changed by the file package is noted, and the `FILECREATED` expression is written onto the output file. For each `DEFINEQ` expression, `TCOMPL` adds any `nlambda` functions defined in the `DEFINEQ` to `NLAMA` or `NLAML`, and adds `lambda` functions to `LAMS`, so that calls to these functions will be compiled correctly. `NLAMA`, `NLAML`, and `LAMS` are rebound to their top level values (using `RESETVAR`) by all of the compiling functions, so that any additions to these lists while inside of these functions will not propagate outside. Expressions beginning with `DECLARE` are processed specially. All other expressions are collected to be subsequently written onto the output file.

After processing the file in this fashion, TCOMPL compiles each function, except for those functions which appear on the list DONTCOMPILEFNS (initially NIL), and writes the compiled definition onto the output file. TCOMPL then writes onto the output file the other expressions found in the symbolic file. DONTCOMPILEFNS might be used for functions that compile open, since their definitions would be superfluous when operating with the compiled file. Note that DONTCOMPILEFNS can be set via block declarations.

Note: If the rootname of a file has the property FILETYPE with value CLISP, or value a list containing CLISP, TCOMPL rebinds DWIMIFYCOMPFLG to T while compiling the functions on *FILE*, so the compiler will DWIMIFY all expressions before compiling them.

TCOMPL returns a list of the names of the output files. All files are properly terminated and closed. If the compilation of any file is aborted via an error or Control-D, all files are properly closed, and the (partially complete) compiled file is deleted.

(**RECOMPILE** *PFILE CFILE FNS*)

[Function]

The purpose of RECOMPILE is to allow you to update a compiled file without recompiling every function in the file. RECOMPILE does this by using the results of a previous compilation. It produces a compiled file similar to one that would have been produced by TCOMPL, but at a considerable savings in time by only compiling selected functions, and copying the compiled definitions for the remainder of the functions in the file from an earlier TCOMPL or RECOMPILE file.

PFILE is the name of the **P**retty file (source file) to be compiled; *CFILE* is the name of the **C**ompiled file containing compiled definitions that may be copied. *FNS* indicates which functions in *PFILE* are to be recompiled, e.g., have been changed or defined for the first time since *CFILE* was made. Note that *PFILE*, not *FNS*, drives RECOMPILE.

RECOMPILE asks the standard compiler questions, except for "OUTPUT FILE:". As with TCOMPL, the output automatically goes to *PFILE*.DCOM. RECOMPILE processes *PFILE* the same as does TCOMPL except that DEFINEQ expressions are not actually read into core. Instead, RECOMPILE uses the filemap to obtain a list of the functions contained in *PFILE*. The filemap enables RECOMPILE to skip over the DEFINEQs in the file by simply resetting the file pointer, so that in most cases the scan of the symbolic file is very fast (the only processing required is the reading of the non-DEFINEQs and the processing of the DECLARE: expressions as with TCOMPL). A map is built if the symbolic file does not already contain one, for example if it was written in an earlier system, or with BUILDMAPFLG = NIL.

After this initial scan of *PFILE*, RECOMPILE then processes the functions defined in the file. For each function in *PFILE*, RECOMPILE determines whether or not the function is to be (re)compiled. Functions that are members of DONTCOMPILEFNS are simply ignored. Otherwise, a function is recompiled if :

1. *FNS* is a list and the function is a member of that list
2. *FNS* = T or EXPRS and the function is defined by an expr definition

INTERLISP-D REFERENCE MANUAL

3. *FNS* = *CHANGES* and the function is marked as having been changed in the *FILECREATED* expression in *PFILE*
4. *FNS* = *ALL*

If a function is not to be recompiled, *RECOMPILE* obtains its compiled definition from *CFILE*, and copies it (and all generated subfunctions) to the output file, *PFILE.DCOM*. If the function does not appear on *CFILE*, *RECOMPILE* simply recompiles it. Finally, after processing all functions, *RECOMPILE* writes out all other expressions that were collected in the prescan of *PFILE*.

Note: If *FNS* = *ALL*, *CFILE* is superfluous, and does not have to be specified. This option may be used to compile a symbolic file that has never been compiled before, but which has already been loaded (since using *TCOMPL* would require reading the file in a second time).

If *CFILE* = *NIL*, *PFILE.DCOM* (the old version of the output file) is used for copying *from*. If both *FNS* and *CFILE* are *NIL*, *FNS* is set to the value of *RECOMPILEDEFAULT*, which is initially *CHANGES*. Thus you can perform his edits, dump the file, and then simply (*RECOMPILE 'FILE*) to update the compiled file.

The value of *RECOMPILE* is the file name of the new compiled file, *PFILE.DCOM*. If *RECOMPILE* is aborted due to an error or Control-D, the new (partially complete) compiled file will be closed and deleted.

RECOMPILE is designed to allow you to conveniently and *efficiently* update a compiled file, even when the corresponding symbolic file has not been (completely) loaded. For example, you can perform a *LOADFROM* to "notice" a symbolic file, edit the functions he wants to change (the editor will automatically load those functions not already loaded), call *MAKEFILE* to update the symbolic file (*MAKEFILE* will copy the unchanged functions from the old symbolic file), and then perform (*RECOMPILE PFILE*).

Note: Since *PRETTYDEF* automatically outputs a suitable *DECLARE* expression to indicate which functions in the file (if any) are defined as *NLAMBDAS*, calls to these functions will be handled correctly, even though the *NLAMBDA* functions themselves may never be loaded, or even looked at, by *RECOMPILE*.

Block Compiling

In Interlisp-10, block compiling provides a way of compiling several functions into a single block. Function calls between the component functions of the block are very fast. Thus, compiling a block consisting of just a single recursive function may yield great savings if the function calls itself many times. The output of a block compilation is a single, usually large, function. Calls from within the block to functions outside of the block look like regular function calls. A block can be entered via several different functions, called entries. These must be specified when the block is compiled.

In Medley, block compiling is handled somewhat differently; block compiling provides a mechanism for hiding function names internal to a block, but it does not provide a performance improvement. Block compiling in Medley works by automatically renaming the block functions with special names, and calling these functions with the normal function-calling mechanisms. Specifically, a function *FN* is renamed to `\BLOCK-NAME/FN`. For example, function `FOO` in block `BAR` is renamed to `\BAR/FOO`. Note that it is possible with this scheme to break functions internal to a block.

Block Declarations

Block compiling a file frequently involves giving the compiler a lot of information about the nature and structure of the compilation, e.g., block functions, entries, specvars, etc. To help with this, there is the `BLOCKS` file package command, which has the form:

```
(BLOCKS BLOCK ... BLOCK )
```

where each *BLOCK* is a block declaration. The `BLOCKS` command outputs a `DECLARE:` expression, which is noticed by `BCOMPL` and `BRECOMPILE`. `BCOMPL` and `BRECOMPILE` are sensitive to these declarations and take the appropriate action.

Note: Masterscope includes a facility for checking the block declarations of a file or files for various anomalous conditions, e.g. functions in block declarations which aren't on the file(s), functions in `ENTRIES` not in the block, variables that may not need to be `SPECVARS` because they are not used freely below the places they are bound, etc.

A block declaration is a list of the form:

```
(BLKNAME BLKFN ... BLKFN
  (VAR . VALUE ) ... (VAR . VALUE ))
```

BLKNAME is the name of a block. *BLKFN ... BLKFN* are the functions in the block and correspond to *BLKFNS* in the call to `BLOCKCOMPILE`. The `(VAR . VALUE )` expressions indicate the settings for variables affecting the compilation of that block. If *VALUE* is atomic, then *VAR* is set to *VALUE* , otherwise *VAR* is set to the UNION of *VALUE* and the current value of the variable *VAR* . Also, expressions of the form `(VAR * FORM)` will cause *FORM* to be evaluated and the resulting list used as described above (e.g. `(GLOBALVARS * MYGLOBALVARS)`).

For example, consider the block declaration below. The block name is `EDITBLOCK`, it includes a number of functions (`EDITL0`, `EDITL1`, ... `EDITH`), and it sets the variables `ENTRIES`, `SPECVARS`, `RETFNS`, and `GLOBALVARS`.

```
(EDITBLOCK
  EDITL0 EDITL1 UNDOEDITL EDITCOM EDITCOMA
  EDITMAC EDITCOMS EDIT]UNDO UNDOEDITCOM EDITH
  (ENTRIES EDITL0 ## UNDOEDITL)
  (SPECVARS L COM LCFLG #1 #2 #3 LISPXBUFFS)
  (RETFNS EDITL0)
```

INTERLISP-D REFERENCE MANUAL

(GLOBALVARS EDITCOMSA EDITCOMSL EDITOPS)

Whenever BCOMPL or BRECOMPILE encounter a block declaration, they rebind RETFNS, SPECVARS, GLOBALVARS, BLKLIBRARY, and DONTCOMPILEFNS to their top level values, bind BLKAPPLYFNS and ENTRIES to NIL, and bind BLKNAME to the first element of the declaration. They then scan the rest of the declaration, setting these variables as described above. When the declaration is exhausted, the block compiler is called and given BLKNAME, the list of block functions, and ENTRIES.

If a function appears in a block declaration, but is not defined in one of the files, then if it has an in-core definition, this definition is used and a message printed NOT ON FILE, COMPILING IN CORE DEFINITION. Otherwise, the message NOT COMPILEABLE, is printed and the block declaration processed as though the function were not on it, i.e. calls to the function will be compiled as external function calls.

Since all compiler variables are rebound for each block declaration, the declaration only has to set those variables it wants *changed*. Furthermore, setting a variable in one declaration has no effect on the variable's value for another declaration.

After finishing all blocks, BCOMPL and BRECOMPILE treat any functions in the file that did not appear in a block declaration in the same way as do TCOMPL and RECOMPILE. If you wish a function compiled separately as well as in a block, or if you wish to compile some functions (not blockcompile), with some compiler variables changed, you can use a special pseudo-block declaration of the form

(NIL BLKFN ... BLKFN (VAR . VALUE) ... (VAR . VALUE))

which means that BLKFN ... BLKFN should be compiled after first setting VAR ... VAR as described above.

The following variables control other aspects of compiling a block:

RETFNS [Variable]

Value is a list of internal block functions whose names must appear on the stack, e.g., if the function is to be returned from RETFROM, RETTO, RETEVAL, etc. Usually, internal calls between functions in a block are not put on the stack.

BLKAPPLYFNS [Variable]

Value is a list of internal block functions called by other functions in the same block using BLKAPPLY or BLKAPPLY* for efficiency reasons.

Normally, a call to APPLY from inside a block would be the same as a call to any other function outside of the block. If the first argument to APPLY turned out to be one of the entries to the block, the block would have to be reentered. BLKAPPLYFNS enables a program to compute the name of a function in the block to be called next, without the overhead of leaving the block and reentering it. This is done by including on the list BLKAPPLYFNS those functions which will be called in this fashion, and by using

BLKAPPLY in place of APPLY, and BLKAPPLY* in place of APPLY*. If BLKAPPLY or BLKAPPLY* is given a function not on BLKAPPLYFNS, the effect is the same as a call to APPLY or APPLY* and no error is generated. Note however, that BLKAPPLYFNS must be set at *compile* time, not run time, and furthermore, that all functions on BLKAPPLYFNS must be in the block, or an error is generated (at compile time), NOT ON BLKFNS.

BLKAPPLYFNS

[Variable]

Value is a list of functions that are considered to be in the "block library" of functions that should automatically be included in the block if they are called within the block.

Compiling a function open via a macro provides a way of eliminating a function call. For block compiling, the same effect can be achieved by including the function in the block. A further advantage is that the code for this function will appear only once in the block, whereas when a function is compiled open, its code appears at each place where it is called.

The block library feature provides a convenient way of including functions in a block. It is just a convenience since you can always achieve the same effect by specifying the function(s) in question as one of the block functions, provided it has an expr definition at compile time. The block library feature simply eliminates the burden of supplying this definition.

To use the block library feature, place the names of the functions of interest on the list BLKLIBRARY, and their expr definitions on the property list of the functions under the property BLKLIBRARYDEF. When the block compiler compiles a form, it first checks to see if the function being called is one of the block functions. If not, and the function is on BLKLIBRARY, its definition is obtained from the property value of BLKLIBRARYDEF, and it is automatically included as part of the block.

Block Compiling Functions

There are three user level functions for block compiling, BLOCKCOMPILE, BCOMPL, and BRECOMPILE, corresponding to COMPILE, TCOMPL, and RECOMPILE. Note that all of the remarks on macros, globalvars, compiler messages, etc., all apply equally for block compiling. Using block declarations, you can intermix in a single file functions compiled normally and block compiled functions.

(**BLOCKCOMPILE** *BLKNAME BLKFNS ENTRIES FLG*)

[Function]

BLKNAME is the name of a block, *BLKFNS* is a list of the functions comprising the block, and *ENTRIES* a list of entries to the block.

Each of the entries must also be on *BLKFNS* or an error is generated, NOT ON BLKFNS. If only one entry is specified, the block name can also be one of the *BLKFNS*, e.g., (BLOCKCOMPILE 'FOO ' (FOO FIE FUM) ' (FOO)). However, if more than one entry is specified, an error will be generated, CAN'T BE BOTH AN ENTRY AND THE BLOCK NAME.

If *ENTRIES* is NIL, (LIST *BLKNAME*) is used, e.g., (BLOCKCOMPILE 'COUNT ' (COUNT COUNT1))

INTERLISP-D REFERENCE MANUAL

If *BLKFNS* is *NIL*, (*LIST BLKNAME*) is used, e.g., (*BLOCKCOMPILE 'EQUAL*)

BLOCKCOMPILE asks the standard compiler questions, and then begins compiling. As with *COMPILE*, if the compiled code is being written to a file, the file is closed unless *FLG* = *T*. The value of *BLOCKCOMPILE* is a list of the entries, or if *ENTRIES* = *NIL*, the value is *BLKNAME*.

The output of a call to *BLOCKCOMPILE* is one function definition for *BLKNAME*, plus definitions for each of the functions on *ENTRIES* if any. These entry functions are very short functions which immediately call *BLKNAME*.

(**BCOMPL** *FILES CFILE*)

[Function]

FILES is a list of symbolic files (if atomic, (*LIST FILES*) is used). *BCOMPL* differs from *TCOMPL* in that it compiles all of the files at once, instead of one at a time, in order to permit one block to contain functions in several files. (If you have several files to be *BCOMPL*ed separately, you must make several calls to *BCOMPL*.) Output is to *CFILE* if given, otherwise to a file whose name is (*CAR FILES*) suffixed with *DCOM*. For example, (*BCOMPL ' (EDIT WEDIT)*) produces one file, *EDIT.DCOM*.

BCOMPL asks the standard compiler questions, except for "OUTPUT FILE:", then processes each file exactly the same as *TCOMPL*. *BCOMPL* next processes the block declarations as described above. Finally, it compiles those functions not mentioned in one of the block declarations, and then writes out all other expressions.

If any of the files have property *FILETYPE* with value *CLISP*, or a list containing *CLISP*, then *DWIMIFYCOMPFLG* is rebound to *T* for all of the files.

The value of *BCOMPL* is the output file (the new compiled file). If the compilation is aborted due to an error or Control-D, all files are closed and the (partially complete) output file is deleted.

It is permissible to *TCOMPL* files set up for *BCOMPL*; the block declarations will simply have no effect. Similarly, you can *BCOMPL* a file that does not contain any block declarations and the result will be the same as having *TCOMPL*ed it.

(**BRECOMPILE** *FILES CFILE FNS* —)

[Function]

BRECOMPILE plays the same role for *BCOMPL* that *RECOMPILE* plays for *TCOMPL*. Its purpose is to allow you to update a compiled file without requiring an entire *BCOMPL*.

FILES is a list of symbolic files (if atomic, (*LIST FILES*) is used). *CFILE* is the compiled file produced by *BCOMPL* or a previous *BRECOMPILE* that contains compiled definitions that may be copied. The interpretation of *FNS* is the same as with *RECOMPILE*.

BRECOMPILE asks the standard compiler questions, except for "OUTPUT FILE:". As with *BCOMPL*, output automatically goes to *FILE.DCOM*, where *FILE* is the first file in *FILES*.

BRECOMPILE processes each file the same as *RECOMPILE*, then processes each block declaration. If any of the functions in the block are to be recompiled, the entire block must be (is) recompiled. Otherwise, the block is copied from *CFILE* as with *RECOMPILE*. For

pseudo-block declarations of the form `(NIL FN . . .)`, all variable assignments are made, but only those functions indicated by *FNS* are recompiled.

After completing the block declarations, BRECOMPILE processes all functions that do not appear in a block declaration, recompiling those dictated by *FNS*, and copying the compiled definitions of the remaining from *CFILE*.

Finally, BRECOMPILE writes onto the output file the "other expressions" collected in the initial scan of *FILES*.

The value of BRECOMPILE is the output file (the new compiled file). If the compilation is aborted due to an error or Control-D, all files are closed and the (partially complete) output file is deleted.

If *CFILE* = NIL, the old version of *FILE.DCOM* is used, as with RECOMPILE. In addition, if *FNS* and *CFILE* are both NIL, *FNS* is set to the value of RECOMPILEDEFAULT, initially CHANGES.

Compiler Error Messages

Messages describing errors in the function being compiled are also printed on the terminal. These messages are always preceded by *****. Unless otherwise indicated below, the compilation will continue.

`(FN NOT ON FILE, COMPILING IN CORE DEFINITION)`

From calls to BCOMPL and BRECOMPILE.

`(FN NOT COMPILEABLE)`

An EXPR definition for *FN* could not be found. In this case, no code is produced for *FN*, and the compiler proceeds to the next function to be compiled, if any.

`(FN NOT FOUND)`

Occurs when RECOMPILE or BRECOMPILE try to copy the compiled definition of *FN* from *CFILE*, and cannot find it. In this case, no code is copied and the compiler proceeds to the next function to be compiled, if any.

`(FN NOT ON BLKFNS)`

FN was specified as an entry to a block, or else was on BLKAPPLYFNS, but did not appear on the *BLKFNS*. In this case, no code is produced for the entire block and the compiler proceeds to the next function to be compiled, if any.

INTERLISP-D REFERENCE MANUAL

(*FN* CAN'T BE BOTH AN ENTRY AND THE BLOCK NAME)

In this case, no code is produced for the entire block and the compiler proceeds to the next function to be compiled, if any.

(*BLKNAME* - USED BLKAPPLY WHEN NOT APPLICABLE)

BLKAPPLY is used in the block *BLKNAME*, but there are no BLKAPPLYFNS or ENTRIES declared for the block.

(*VAR* SHOULD BE A SPECVAR - USED FREELY BY *FN*)

While compiling a block, the compiler has already generated code to bind *VAR* as a LOCALVAR, but now discovers that *FN* uses *VAR* freely. *VAR* should be declared a SPECVAR and the block recompiled.

((*** --) COMMENT USED FOR VALUE)

A comment appears in a context where its value is being used, e.g. (LIST X (*** --) Y). The compiled function will run, but the value at the point where the comment was used is undefined.

((*FORM*) - NON-ATOMIC CAR OF FORM)

If you intended to treat the value of *FORM* as a function, you should use APPLY* (Chapter 10). *FORM* is compiled as if APPLY* had been used.

((SETQ *VAR* *EXPR* --) BAD SETQ)

SETQ of more than two arguments.

(*FN* - USED AS ARG TO NUMBER FN?)

The value of a predicate, such as GREATERP or EQ, is used as an argument to a function that expects numbers, such as IPLUS.

(*FN* - NO LONGER INTERPRETED AS FUNCTIONAL ARGUMENT)

The compiler has assumed *FN* is the name of a function. If you intended to treat the *value* of *FN* as a function, APPLY* (Chapter 10) should be used. This message is printed when *FN* is not defined, and is also a local variable of the function being compiled.

(*FN* - ILLEGAL RETURN)

RETURN encountered when not in PROG.

(*TG* - ILLEGAL GO)

GO encountered when not in a PROG.

(*TG* - MULTIPLY DEFINED TAG)

TG is a PROG label that is defined more than once in a single PROG. The second definition is ignored.

(*TG* - UNDEFINED TAG)

TG is a PROG label that is referenced but not defined in a PROG.

(*VAR* - NOT A BINDABLE VARIABLE)

VAR is NIL, T, or else not a literal atom.

(*VAR VAL* -- BAD PROG BINDING)

Occurs when there is a prog binding of the form (*VAR VAL* ... *VAL*).

(*TG* - MULTIPLY DEFINED TAG, LAP)

TG is a label that was encountered twice during the second pass of the compilation. If this error occurs with no indication of a multiply defined tag during pass one, the tag is in a LAP macro.

(*TG* - UNDEFINED TAG, LAP)

TG is a label that is referenced during the second pass of compilation and is not defined. LAP treats *TG* as though it were a COREVAL, and continues the compilation.

(*TG* - MULTIPLY DEFINED TAG, ASSEMBLE)

TG is a label that is defined more than once in an assemble form.

(*TG* - UNDEFINED TAG, ASSEMBLE)

TG is a label that is referenced but not defined in an assemble form.

(*OP* - OPCODE? - ASSEMBLE)

OP appears as CAR of an assemble statement, and is illegal.

(NO BINARY CODE GENERATED OR LOADED FOR *FN*)

A previous error condition was sufficiently serious that binary code for *FN* cannot be loaded without causing an error.

19. DWIM

A surprisingly large percentage of the errors made by Interlisp users are of the type that could be corrected by another Lisp programmer without any information about the purpose of the program or expression in question, e.g., misspellings, certain kinds of parentheses errors, etc. To correct these types of errors we have implemented in Medley a DWIM facility, short for Do-What-I-Mean. DWIM is called automatically whenever an error occurs in the evaluation of an Interlisp expression. (Currently, DWIM only operates on unbound atoms and undefined function errors.) DWIM then proceeds to try to correct the mistake using the current context of computation plus information about what you had previously been doing (and what mistakes you had been making) as guides to the remedy of the error. If DWIM is able to make the correction, the computation continues as though no error had occurred. Otherwise, the procedure is the same as though DWIM had not intervened: a break occurs, or an unwind to the last ERRORSET (see Chapter 14). The following protocol illustrates the operation of DWIM.

For example, suppose you define the factorial function (FACT N) as follows:

```
←DEFINEQ((FACT (LAMBDA (N) (COND
  ((ZEROP NO 1) ((T (ITIMS N (FACCT 9SUB1 N]
  (FACT)
  ←
```

Note that the definition of FACT contains several mistakes: ITIMES and FACT have been misspelled; the 0 in NO was intended to be a right parenthesis, but the Shift key was not pressed; similarly, the 9 in 9SUB1 was intended to be a left parenthesis; and finally, there is an extra left parenthesis in front of the T that begins the final clause in the conditional.

```
←PRETTYPRNT((FACCT]
  =PRETTYPRINT
  =FACT

  (FACT
    [LAMBDA (N)
      (COND
        ((ZEROP NO 1)
          ((T (ITIMS N (FACCT 9SUB1 N])
          (FACT)
  ←
```

After defining FACT, you want to look at its definition using PRETTYPRINT, which you unfortunately misspell. Since there is no function PRETTYPRNT in the system, an undefined function error occurs, and DWIM is called. DWIM invokes its spelling corrector, which searches a list of functions frequently used (by *this* user) for the best possible match. Finding one that is extremely close, DWIM proceeds on the assumption that PRETTYPRNT meant PRETTYPRINT, notifies you of this, and calls PRETTYPRINT.

INTERLISP-D REFERENCE MANUAL

DWIM

At this point, PRETTYPRINT would normally print (FACCT NOT PRINTABLE) and exit, since FACCT has no definition. Note that this is *not* an Interlisp error condition, so that DWIM would not be called as described above. However, it is obviously not what you *meant*.

This sort of mistake is corrected by having PRETTYPRINT itself explicitly invoke the spelling corrector portion of DWIM whenever given a function with no EXPR definition. Thus, with the aid of DWIM PRETTYPRINT is able to determine that you want to see the definition of the function FACT, and proceeds accordingly.

```

←FACT(3)
  NO [IN FACT] -> N ) ? YES
  [IN FACT] (COND -- ((T --))) ->
              (COND -- (T --))
  ITIMS [IN FACT] -> ITIMES
  FACCT [IN FACT] -> FACT
  9SUB1 [IN FACT] -> ( SUB1 ? YES
6

←PP FACT
  (FACT
    [LAMBDA (N)
      (COND
        ((ZEROP N)
         1)
        (T (ITIMES N (FACT (SUB1 N))
         FACT
         ←

```

You now call FACT. During its execution, five errors occur, and DWIM is called five times. At each point, the error is corrected, a message is printed describing the action taken, and the computation is allowed to continue as if no error had occurred. Following the last correction, 6 is printed, the value of (FACT 3). Finally, you prettyprint the new, now correct, definition of FACT.

In this particular example, you were operating in TRUSTING mode, which gives DWIM carte blanche for most corrections. You can also operate in CAUTIOUS mode, in which case DWIM will inform you of intended corrections before they are made, and allow you to approve or disapprove of them. If DWIM was operating in CAUTIOUS mode in the example above, it would proceed as follows:

```

←FACT(3)
  NO [IN FACT] -> N ) ? YES
  U.D.F. T [IN FACT] FIX? YES
  [IN FACT] (COND -- ((T --))) ->
              (COND -- (T --))
  ITIMS [IN FACT] -> ITIMES ? ...YES
  FACCT [IN FACT] -> FACT ? ...YES
  9SUB1 [IN FACT] -> ( SUB1 ? NO
  U.B.A.
  (9SUB1 BROKEN)
  :

```

For most corrections, if you do not respond in a specified interval of time, DWIM automatically proceeds with the correction, so that you need intervene only when you do not approve. In the example, you responded to the first, second, and fifth questions; DWIM responded for you on the third and fourth.

DWIM uses ASKUSER for its interactions with you (see Chapter 26). Whenever an interaction is about to take place and you have typed ahead, ASKUSER types several bells to warn you to stop typing, then clears and saves the input buffers, restoring them after the interaction is complete. Thus if you typed ahead before a DWIM interaction, DWIM will not confuse your type-ahead with the answer to its question, nor will your type-ahead be lost. The bells are printed by the function PRINTBELLS, which can be advised or redefined for specialized applications, e.g. to flash the screen for a display terminal.

A great deal of effort has gone into making DWIM "smart", and experience with a large number of users indicates that DWIM works very well; DWIM seldom fails to correct an error you feel it should have, and almost never mistakenly corrects an error. However, it is important to note that even when DWIM is wrong, no harm is done: since an error had occurred, you would have had to intervene anyway if DWIM took no action. Thus, if DWIM mistakenly corrects an error, you simply interrupt or abort the computation, reverse the DWIM change using UNDO (see Chapter 13), and make the correction you would have had to make without DWIM. An exception is if DWIM's correction mistakenly caused a destructive computation to be initiated, and information was lost before you could interrupt. We have not yet had such an incident occur.

(DWIM X)

[Function]

Used to enable/disable DWIM. If X is the symbol C, DWIM is enabled in CAUTIOUS mode, so that DWIM will ask you before making corrections. If X is T, DWIM is enabled in TRUSTING mode, so DWIM will make most corrections automatically. If X is NIL, DWIM is disabled. Medley initially has DWIM enabled in CAUTIOUS mode.

DWIM returns CAUTIOUS, TRUSTING or NIL, depending to what mode it has just been put into.

For corrections to expressions typed in for immediate execution (typed into LISPX, Chapter 13), DWIM always acts as though it were in TRUSTING mode, i.e., no approval necessary. For certain types of corrections, e.g., run-on spelling corrections, 9-0 errors, etc., DWIM always acts like it was in CAUTIOUS mode, and asks for approval. In either case, DWIM always informs you of its action as described below.

Spelling Correction Protocol

One type of error that DWIM can correct is the misspelling of a function or a variable name. When an unbound symbol or undefined function error occurs, DWIM tries to correct the spelling of the bad symbol. If a symbol is found whose spelling is "close" to the offender, DWIM proceeds as follows:

INTERLISP-D REFERENCE MANUAL

DWIM

If the correction occurs in the typed-in expression, DWIM prints *=CORRECT-SPELLING* and continues evaluating the expression. For example:

```
← (SETQ FOO (IPLUSS 1 2))  
      =IPLUS  
      3
```

If the correction does not occur in type-in, DWIM prints

```
BAD-SPELLING [IN FUNCTION-NAME] -> CORRECT-SPELLING
```

The appearance of *->* is to call attention to the fact that the user's function will be or has been changed.

Then, if DWIM is in *TRUSTING* mode, it prints a carriage return, makes the correction, and continues the computation. If DWIM is in *CAUTIOUS* mode, it prints a few spaces and *?* and then wait for approval. The user then has six options:

1. Type *Y*. DWIM types *es*, and proceeds with the correction.
2. Type *N*. DWIM types *o*, and does not make the correction.
3. Type *↑*. DWIM does not make the correction, and furthermore guarantees that the error will not cause a break.
4. Type Control-E. For error correction, this has the same effect as typing *N*.
5. Do nothing. In this case DWIM waits for *DWIMWAIT* seconds, and if you have not responded, DWIM will type *. . .* followed by the default answer.

The default on spelling corrections is determined by the value of the variable *FIXSPELLDEFAULT*, whose top level value is initially *Y*.

6. Type space or carriage-return. In this case DWIM will wait indefinitely. This option is intended for those cases where you want to think about your answer, and want to insure that DWIM does not get "impatient" and answer for you.

The procedure for spelling correction on other than Interlisp errors is analogous. If the correction is being handled as type-in, DWIM prints *=* followed by the correct spelling, and returns it to the function that called DWIM. Otherwise, DWIM prints the incorrect spelling, followed by the correct spelling. Then, if DWIM is in *TRUSTING* mode, DWIM prints a carriage-return and returns the correct spelling. Otherwise, DWIM prints a few spaces and a *?* and waits for approval. You can then respond with *Y*, *N*, Control-E, space, carriage return, or do nothing as described above.

The spelling corrector itself is not *ERRORSET* protected like the DWIM error correction routines. Therefore, typing *N* and typing Control-E may have different effects when the spelling corrector is called directly. The former simply instructs the spelling corrector to return *NIL*, and lets the calling

function decide what to do next; the latter causes an error which unwinds to the last `ERRORSET`, however far back that may be.

Parentheses Errors Protocol

When an unbound symbol or undefined error occurs, and the offending symbol contains 9 or 0, DWIM tries to correct errors caused by typing 9 for left parenthesis and 0 for right parenthesis. In these cases, the interaction with you is similar to that for spelling correction. If the error occurs in type-in, DWIM types `=CORRECTION`, and continues evaluating the expression. For example:

```
←(SETQ FOO 9IPLUS 1 2]
    = ( IPLUS
      3
```

If the correction does not occur in type-in, DWIM prints

```
BAD-ATOM [IN FUNCTION-NAME] -> CORRECTION ?
```

and then waits for approval. You then have the same six options as for spelling correction, except the waiting time is `3*DWIMWAIT` seconds. If you type Y, DWIM operates as if it were in `TRUSTING` mode, i.e., it makes the correction and prints its message.

Actually, DWIM uses the value of the variables `LPARKEY` and `RPARKEY` to determine the corresponding lower case character for left and right parentheses. `LPARKEY` and `RPARKEY` are initially 9 and 0 respectively, but they can be reset for other keyboard layouts, e.g., on some terminals left parenthesis is over 8, and right parenthesis is over 9.

Undefined Function T Errors

When an undefined function error occurs, and the offending function is T, DWIM tries to correct certain types of parentheses errors involving a T clause in a conditional. DWIM recognizes errors of the following forms:

<code>(COND --) (T --)</code>	The T clause appears outside and immediately following the COND.
<code>(COND -- (-- & (T --)))</code>	The T clause appears inside a previous clause.
<code>(COND -- ((T --)))</code>	The T clause has an extra pair of parentheses around it.

For undefined function errors that are not one of these three types, DWIM takes no corrective action at all, and the error will occur.

INTERLISP-D REFERENCE MANUAL

DWIM

If the error occurs in type-in, DWIM simply types `T FIXED` and makes the correction. Otherwise if DWIM is in `TRUSTING` mode, DWIM makes the correction and prints the message:

```
[IN FUNCTION-NAME] {BAD-COND} ->
                      {CORRECTED-COND}
```

If DWIM is in `CAUTIOUS` mode, DWIM prints

```
UNDEFINED FUNCTION T
[IN FUNCTION-NAME]   FIX?
```

and waits for approval. You then have the same options as for spelling corrections and parenthesis errors. If you type `Y` or default, DWIM makes the correction and prints its message.

Having made the correction, DWIM must then decide how to proceed with the computation. In the first case, `(COND --) (T --)`, DWIM cannot know whether the `T` clause would have been executed if it had been inside of the `COND`. Therefore DWIM asks you `CONTINUE WITH T CLAUSE` (with a default of `YES`). If you type `N`, DWIM continues with the form after the `COND`, i.e., the form that originally followed the `T` clause.

In the second case, `(COND -- (-- & (T --)))`, DWIM has a different problem. After moving the `T` clause to its proper place, DWIM must return as the value of `&` as the value of the `COND`. Since this value is no longer around, DWIM asks you `OK TO REEVALUATE` and then prints the expression corresponding to `&`. If you type `Y`, or default, DWIM continues by reevaluating `&`, otherwise DWIM aborts, and a `U.D.F. T` error will then occur (even though the `COND` has in fact been fixed). If DWIM can determine for itself that the form can safely be reevaluated, it does not consult you before reevaluating. DWIM can do this if the form is atomic, or `CAR` of the form is a member of the list `OKREEVALST`, and each of the arguments can safely be reevaluated. For example, `(SETQ X (CONS (IPLUS Y Z) W))` is safe to reevaluate because `SETQ`, `CONS`, and `IPLUS` are all on `OKREEVALST`.

In the third case, `(COND -- ((T --)))`, there is no problem with continuation, so no further interaction is necessary.

DWIM Operation

Whenever the interpreter encounters an atomic form with no binding, or a non-atomic form `CAR` of which is not a function or function object, it calls the function `FAULTEVAL`. Similarly, when `APPLY` is given an undefined function, `FAULTAPPLY` is called. When DWIM is enabled, `FAULTEVAL` and `FAULTAPPLY` are redefined to first call the DWIM package, which tries to correct the error. If DWIM cannot decide how to fix the error, or you disapprove of DWIM's correction (by typing `N`), or you type `Control-E`, then `FAULTEVAL` and `FAULTAPPLY` cause an error or break. If you type `↑` to DWIM, DWIM exits by performing `(RETEVAL 'FAULTEVAL ' (ERROR!))`, so that an error will be generated at the position of the call to `FAULTEVAL`.

If DWIM can (and is allowed to) correct the error, it exits by performing RETEVAL of the corrected form, as of the position of the call to FAULTEVAL or FAULTAPPLY. Thus in the example at the beginning of the chapter, when DWIM determined that ITIMS was ITIMES misspelled, DWIM called RETEVAL with (ITIMES N (FACCT 9SUB1 N)). Since the interpreter uses the value returned by FAULTEVAL exactly as though it were the value of the erroneous form, the computation will thus proceed exactly as though no error had occurred.

In addition to continuing the computation, DWIM also repairs the cause of the error whenever possible; in the above example, DWIM also changed (with RPLACA) the expression (ITIMS N (FACCT 9SUB1 N)) that caused the error. Note that if your program had *computed* the form and called EVAL, it would not be possible to repair the cause of the error, although DWIM could correct the misspelling each time it occurred.

Error correction in DWIM is divided into three categories: unbound atoms, undefined CAR of form, and undefined function in APPLY. Assuming that the user approves DWIM's corrections, the action taken by DWIM for the various types of errors in each of these categories is summarized below.

DWIM Correction: Unbound Atoms

If DWIM is called as the result of an unbound atom error, it proceeds as follows:

1. If the first character of the unbound atom is ' , DWIM assumes that you (intentionally) typed 'ATOM for (QUOTE ATOM) and makes the appropriate change. No message is typed, and no approval is requested.

If the unbound atom is just ' itself, DWIM assumes you want the *next* expression quoted, e.g., (CONS X ' (A B C)) will be changed to (CONS X (QUOTE (A B C))). Again no message will be printed or approval asked. If no expression follows the ' , DWIM gives up.

Note: ' is normally defined as a read-macro character which converts 'FOO to (QUOTE FOO) on input, so DWIM will not see the ' in the case of expressions that are typed-in.

2. If CLISP (see Chapter 21) is enabled, and the atom is part of a CLISP construct, the CLISP transformation is performed and the result returned. For example, N-1 is transformed to (SUB1 N), and (... FOO_3 ...) is transformed into (... (SETQ FOO 3) ...).
3. If the atom contains an 9 (actually LPARKEY (see the DWIM Functions and Variables section below), DWIM assumes the 9 was intended to be a left parenthesis, and calls the editor to make appropriate repairs on the expression containing the atom. DWIM assumes that you did not notice the mistake, i.e., that the entire expression was affected by the missing left parenthesis. For example, if you type (SETQ X (LIST (CONS 9CAR Y) (CDR Z)) Y), the expression will be changed to (SETQ X (LIST (CONS (CAR Y) (CDR Z)) Y)). The 9 does not have to be the first character of the atom: DWIM will handle (CONS X9CAR Y) correctly.

INTERLISP-D REFERENCE MANUAL

DWIM

4. If the atom contains a 0 (actually RPARKEY, see the DWIM Functions and Variables section below), DWIM assumes the 0 was intended to be a right parenthesis and operates as in the case above.
5. If the atom begins with a 7, the 7 is treated as a '. For example, 7FOO becomes 'FOO, and then (QUOTE FOO).
6. The expressions on DWIMUSERFORMS (see the DWIMUSERFORMS section below) are evaluated in the order that they appear. If any of these expressions returns a non-NIL value, this value is treated as the form to be used to continue the computation, it is evaluated and its value is returned by DWIM.
7. If the unbound atom occurs in a function, DWIM attempts spelling correction using the LAMBDA and PROG variables of the function as the spelling list.
8. If the unbound atom occurred in a type-in to a break, DWIM attempts spelling correction using the LAMBDA and PROG variables of the broken function as the spelling list.
9. Otherwise, DWIM attempts spelling correction using SPELLINGS3 (see the Spelling Lists section below).
10. If all of the above fail, DWIM gives up.

Undefined CAR of Form

If DWIM is called as the result of an undefined CAR of form error, it proceeds as follows:

1. If CAR of the form is T, DWIM assumes a misplaced T clause and operates as described in the Undefined Function T Errors section above.
2. If CAR of the form is F/L, DWIM changes the "F/L" to "FUNCTION(LAMBDA". For example, (F/L (Y) (PRINT (CAR Y))) is changed to (FUNCTION (LAMBDA (Y) (PRINT (CAR Y))). No message is printed and no approval requested. If you omit the variable list, DWIM supplies (X), e.g., (F/L (PRINT (CAR X))) is changed to (FUNCTION (LAMBDA (X) (PRINT (CAR X))). DWIM determines that you have supplied the variable list when more than one expression follows F/L, CAR of the first expression is not the name of a function, and every element in the first expression is atomic. For example, DWIM will supply (X) when correcting (F/L (PRINT (CDR X)) (PRINT (CAR X))).
3. If CAR of the form is a CLISP word (IF, FOR, DO, FETCH, etc.), the indicated CLISP transformation is performed, and the result is returned as the corrected form. See Chapter 21.
4. If CAR of the form has a function definition, DWIM attempts spelling correction on CAR of the definition using as spelling list the value of LAMBDA\$PLST, initially (LAMBDA NLAMBDA).
5. If CAR of the form has an EXPR or CODE property, DWIM prints *CAR-OF-FORM UNSAVED*, performs an UNSAVEDEF, and continues. No approval is requested.

6. If CAR of the form has a FILEDEF property, the definition is loaded from a file (except when DWIMIFYing). If the value of the property is atomic, the entire file is to be loaded. If the value is a list, CAR is the name of the file and CDR the relevant functions, and LOADFNS will be used. For both cases, *LDFLG* will be SYSLOAD (see Chapter 17). DWIM uses FINDFILE (Chapter 24), so that the file can be on any of the directories on DIRECTORIES, initially (NIL NEWLISP LISP LISPUSERS). If the file is found, DWIM types SHALL I LOAD followed by the file name or list of functions. If you approve, DWIM loads the function(s) or file, and continues the computation.
7. If CLISP is enabled, and CAR of the form is part of a CLISP construct, the indicated transformation is performed, e.g., $(N \leftarrow N-1)$ becomes (SETQ N (SUB1 N)).
8. If CAR of the form contains an 9, DWIM assumes a left parenthesis was intended e.g., (CONS9CAR X).
9. If CAR of the form contains a 0, DWIM assumes a right parenthesis was intended.
10. If CAR of the form is a list, DWIM attempts spelling correction on CAAR of the form using LAMBDA SPLST as spelling list. If successful, DWIM returns the corrected expression itself.
11. The expressions on DWIMUSERFORMS are evaluated in the order they appear. If any returns a non-NIL value, this value is treated as the corrected form, it is evaluated, and DWIM returns its value.
12. Otherwise, DWIM attempts spelling correction using SPELLINGS2 as the spelling list (see the Spelling Lists section below). When DWIMIFYing, DWIM also attempts spelling correction on function names not defined but previously encountered, using NOFIXFNSLST as a spelling list (see Chapter 21).
13. If all of the above fail, DWIM gives up.

Undefined Function in APPLY

If DWIM is called as the result of an undefined function in APPLY error, it proceeds as follows:

1. If the function has a definition, DWIM attempts spelling correction on CAR of the definition using LAMBDA SPLST as spelling list.
2. If the function has an EXPR or CODE property, DWIM prints *FN UNSAVED*, performs an UNSAVEDEF and continues. No approval is requested.
3. If the function has a property FILEDEF, DWIM proceeds as in case 6 of undefined CAR of form.
4. If the error resulted from type-in, and CLISP is enabled, and the function name contains a CLISP operator, DWIM performs the indicated transformation, e.g., type $FOO \leftarrow (APPEND FIE FUM)$.
5. If the function name contains an 9, DWIM assumes a left parenthesis was intended, e.g., EDIT9FOO].

INTERLISP-D REFERENCE MANUAL

DWIM

6. If the "function" is a list, DWIM attempts spelling correction on CAR of the list using LAMBDA SPLST as spelling list.
7. The expressions on DWIMUSERFORMS are evaluated in the order they appear, and if any returns a non-NIL value, this value is treated as the function used to continue the computation, i.e., it will be applied to its arguments.
8. DWIM attempts spelling correction using SPELLINGS1 as the spelling list.
9. DWIM attempts spelling correction using SPELLINGS2 as the spelling list.
10. If all fail, DWIM gives up.

DWIMUSERFORMS

The variable DWIMUSERFORMS provides a convenient way of adding to the transformations that DWIM performs. For example, you might want to change atoms of the form \$X to (QA4LOOKUP X). Before attempting spelling correction, but after performing other transformations (F/L, 9, 0, CLISP, etc.), DWIM evaluates the expressions on DWIMUSERFORMS in the order they appear. If any expression returns a non-NIL value, this value is treated as the transformed form to be used. If DWIM was called from FAULTEVAL, this form is evaluated and the resulting value is returned as the value of FAULTEVAL. If DWIM is called from FAULTAPPLY, this form is treated as a function to be applied to FAULTARGS, and the resulting value is returned as the value of FAULTAPPLY. If all of the expressions on DWIMUSERFORMS return NIL, DWIM proceeds as though DWIMUSERFORMS = NIL, and attempts spelling correction. Note that DWIM simply takes the value and returns it; the expressions on DWIMUSERFORMS are responsible for making any modifications to the original expression. The expressions on DWIMUSERFORMS should make the transformation permanent, either by associating it with FAULTX via CLISPTRAN, or by destructively changing FAULTX.

In order for an expression on DWIMUSERFORMS to be able to be effective, it needs to know various things about the context of the error. Therefore, several of DWIM's internal variables have been made SPECVARS (see Chapter 18) and are therefore "visible" to DWIMUSERFORMS. Below are a list of those variables that may be useful.

FAULTX [Variable]

For unbound atom and undefined car of form errors, FAULTX is the atom or form. For undefined function in APPLY errors, FAULTX is the name of the function.

FAULTARGS [Variable]

For undefined function in APPLY errors, FAULTARGS is the list of arguments. FAULTARGS may be modified or reset by expressions on DWIMUSERFORMS.

FAULTAPPLYFLG [Variable]

Value is T for undefined function in APPLY errors; NIL otherwise. The value of FAULTAPPLYFLG *after* an expression on DWIMUSERFORMS returns a non-NIL value determines how the latter value is to be treated. Following an undefined function in APPLY error, if an expression on DWIMUSERFORMS sets FAULTAPPLYFLG to NIL, the value returned is treated as a form to be evaluated, rather than a function to be applied.

FAULTAPPLYFLG is necessary to distinguish between unbound atom and undefined function in APPLY errors, since FAULTARGS may be NIL and FAULTX atomic in both cases.

TAIL [Variable]

For unbound atom errors, TAIL is the tail of the expression CAR of which is the unbound atom. DWIMUSERFORMS expression can replace the atom by another expression by performing (/RPLACA TAIL EXPR)

PARENT [Variable]

For unbound atom errors, PARENT is the form in which the unbound atom appears. TAIL is a tail of PARENT.

TYPE-IN? [Variable]

True if the error occurred in type-in.

FAULTFN [Variable]

Name of the function in which error occurred. FAULTFN is TYPE-IN when the error occurred in type-in, and EVAL or APPLY when the error occurred under an explicit call to EVAL or APPLY.

DWIMIFYFLG [Variable]

True if the error was encountered while DWIMIFYing (as opposed to happening while running a program).

EXPR [Variable]

Definition of FAULTFN, or argument to EVAL, i.e., the superform in which the error occurs.

The initial value of DWIMUSERFORMS is ((DWIMLOADFNS?)). DWIMLOADFNS? is a function for automatically loading functions from files. If DWIMLOADFNSFLG is T (its initial value), and CAR of the form is the name of a function, and the function is contained on a file that has been noticed by the file package, the function is loaded, and the computation continues.

DWIM Functions and Variables

DWIMWAIT [Variable]

Value is the number of seconds that DWIM will wait before it assumes that you are not going to respond to a question and uses the default response `FIXSPELLDEFAULT`.

DWIM operates by dismissing for 250 milliseconds, then checking to see if anything has been typed. If not, it dismisses again, etc. until `DWIMWAIT` seconds have elapsed. Thus, there will be a delay of at most 1/4 second before DWIM responds to your answer.

FIXSPELLDEFAULT [Variable]

If approval is requested for a spelling correction, and you do not respond, defaults to value of `FIXSPELLDEFAULT`, initially Y. `FIXSPELLDEFAULT` is rebound to N when DWIMIFYing.

ADDSPELLFLG [Variable]

If NIL, suppresses calls to `ADDSPELL`. Initially T.

NOSPELLFLG [Variable]

If T, suppresses *all* spelling correction. If some other non-NIL value, suppresses spelling correction in programs but not type-in. `NOSPELLFLG` is initially NIL. It is rebound to T when compiling from a file.

RUNONFLG [Variable]

If NIL, suppresses run-on spelling corrections. Initially NIL.

DWIMLOADFNSFLG [Variable]

If T, tells DWIM that when it encounters a call to an undefined function contained on a file that has been noticed by the file package, to simply load the function. `DWIMLOADFNSFLG` is initially T (see above).

LPARKEY [Variable]

RPARKEY [Variable]

DWIM uses the value of the variables `LPARKEY` and `RPARKEY` (initially 9 and 0 respectively) to determine the corresponding lower case character for left and right parentheses. `LPARKEY` and `RPARKEY` can be reset for other keyboard layouts. For example, on some terminals left parenthesis is over 8, and right parenthesis is over 9.

OKREEVALST [Variable]

The value of `OKREEVALST` is a list of functions that DWIM can safely reevaluate. If a form is atomic, or CAR of the form is a member of `OKREEVALST`, and each of the arguments can safely be reevaluated, then the form can be safely reevaluated. For example, `(SETQ X (CONS (IPLUS Y Z) W))` is safe to reevaluate because `SETQ`, `CONS`, and `IPLUS` are all on `OKREEVALST`.

DWIMFLG [Variable]

DWIMFLG = NIL, all DWIM operations are disabled. (DWIM 'C) and (DWIM T) set DWIMFLG to T; (DWIM NIL) sets DWIMFLG to NIL.

APPROVEFLG [Variable]

APPROVEFLG = T if DWIM should ask the user for approval before making a correction that will modify the definition of one of his functions; NIL otherwise.

When DWIM is put into CAUTIOUS mode with (DWIM 'C), APPROVEFLG is set to T; for TRUSTING mode, APPROVEFLG is set to NIL.

LAMBDA SPLST [Variable]

DWIM uses the value of LAMBDA SPLST as the spelling list when correcting "bad" function definitions. Initially (LAMBDA NLAMBDA). You may wish to add to LAMBDA SPLST if you elect to define new "function types" via an appropriate DWIMUSERFORMS entry. For example, the QLAMBDA's of SRI's QLISP are handled in this way.

Spelling Correction

The spelling corrector is given as arguments a misspelled word (word means symbol), a spelling list (a list of words), and a number: *XWORD*, *SPLST*, and *REL* respectively. Its task is to find that word on *SPLST* which is closest to *XWORD*, in the sense described below. This word is called a *respelling* of *XWORD*. *REL* specifies the minimum "closeness" between *XWORD* and a respelling. If the spelling corrector cannot find a word on *SPLST* closer to *XWORD* than *REL*, or if it finds two or more words equally close, its value is NIL, otherwise its value is the respelling. The spelling corrector can also be given an optional functional argument, *FN*, to be used for selecting out a subset of *SPLST*, i.e., only those members of *SPLST* that satisfy *FN* will be considered as possible respellings.

The exact algorithm for computing the spelling metric is described later, but briefly "closeness" is inversely proportional to the number of disagreements between the two words, and directly proportional to the length of the longer word. For example, PRTTYPRNT is "closer" to PRETTYPRINT than CS is to CONS even though both pairs of words have the same number of disagreements. The spelling corrector operates by proceeding down *SPLST*, and computing the closeness between each word and *XWORD*, and keeping a list of those that are closest. Certain differences between words are not counted as disagreements, for example a single transposition, e.g., CONS to CNOS, or a doubled letter, e.g., CONS to CONSS, etc. In the event that the spelling corrector finds a word on *SPLST* with *no* disagreements, it will stop searching and return this word as the respelling. Otherwise, the spelling corrector continues through the entire spelling list. Then if it has found one and only one "closest" word, it returns this word as the respelling. For example, if *XWORD* is VONS, the spelling corrector will probably return CONS as the respelling. However, if *XWORD* is CONZ, the spelling corrector will not be able to return a respelling, since CONZ is equally close to both CONS and COND. If the spelling corrector finds an acceptable respelling, it interacts with you as described earlier.

INTERLISP-D REFERENCE MANUAL

DWIM

In the special case that the misspelled word contains one or more `$`s (escape), the spelling corrector searches for those words on *SPLST* that match *XWORD*, where a `$` can match any number of characters (including 0), e.g., `FOO$` matches `FOO1` and `FOO`, but not `NEWFOO`. `$FOO$` matches all three. Both completion and correction may be involved, e.g. `RPETTY$` will match `PRETTYPRINT`, with one mistake. The entire spelling list is always searched, and if more than one respelling is found, the spelling corrector prints `AMBIGUOUS`, and returns `NIL`. For example, `CON$` would be ambiguous if both `CONS` and `COND` were on the spelling list. If the spelling corrector finds one and only one respelling, it interacts with you as described earlier.

For both spelling correction and spelling completion, regardless of whether or not you approve of the spelling corrector's choice, the respelling is moved to the front of *SPLST*. Since many respellings are of the type with no disagreements, this procedure has the effect of considerably reducing the time required to correct the spelling of frequently misspelled words.

Synonyms

Spelling lists also provide a way of defining synonyms for a particular context. If a dotted pair appears on a spelling list (instead of just an atom), `CAR` is interpreted as the correct spelling of the misspelled word, and `CDR` as the antecedent for that word. If `CAR` is *identical* with the misspelled word, the antecedent is returned without any interaction or approval being necessary. If the misspelled word *corrects* to `CAR` of the dotted pair, the usual interaction and approval will take place, and then the antecedent, i.e., `CDR` of the dotted pair, is returned. For example, you could make `IFLG` synonymous with `CLISPIFTRANFLG` by adding `(IFLG . CLISPIFTRANFLG)` to `SPELLINGS3`, the spelling list for unbound atoms. Similarly, you could make `OTHERWISE` mean the same as `ELSEIF` by adding `(OTHERWISE . ELSEIF)` to `CLISPIFWORDSPLST`, or make `L` be synonymous with `LAMBDA` by adding `(L . LAMBDA)` to `LAMBDA SPLST`. You can also use `L` as a variable without confusion, since the association of `L` with `LAMBDA` occurs only in the appropriate context.

Spelling Lists

Any list of atoms can be used as a spelling list, e.g., `BROKENFNS`, `FILELST`, etc. Various system packages have their own spellings lists, e.g., `LISPXCOMS`, `CLISPFORWORDSPLST`, `EDITCOMSA`, etc. These are documented under their corresponding sections, and are also indexed under "spelling lists." In addition to these spelling lists, the system maintains, i.e., automatically adds to, and occasionally prunes, four lists used solely for spelling correction: `SPELLINGS1`, `SPELLINGS2`, `SPELLINGS3`, and `USERWORDS`. These spelling lists are maintained *only* when `ADDSPELLFLG` is non-`NIL`. `ADDSPELLFLG` is initially `T`.

SPELLINGS1

[Variable]

`SPELLINGS1` is a list of functions used for spelling correction when an input is typed in apply format, and the function is undefined, e.g., `EDITF(FOO)`. `SPELLINGS1` is initialized to contain `DEFINEQ`, `BREAK`, `MAKEFILE`, `EDITF`, `TCOMPL`, `LOAD`, etc. Whenever `LISPX` is given an input in apply format, i.e., a function and arguments, the name of the function is added to `SPELLINGS1` if the function has a definition.

For example, typing `CALLS (EDITF)` will cause `CALLS` to be added to `SPELLINGS1`. Thus if you typed `CALLS (EDITF)` and later typed `CALLLS (EDITV)`, since `SPELLINGS1` would then contain `CALLS`, `DWIM` would be successful in correcting `CALLLS` to `CALLS`.

SPELLINGS2

[Variable]

`SPELLINGS2` is a list of functions used for spelling correction for all other undefined functions. It is initialized to contain functions such as `ADD1`, `APPEND`, `COND`, `CONS`, `GO`, `LIST`, `NCONC`, `PRINT`, `PROG`, `RETURN`, `SETQ`, etc. Whenever `LISPX` is given a non-atomic form, the name of the function is added to `SPELLINGS2`. For example, typing `(RETFROM (STKPOS (QUOTE FOO) 2))` to a break would add `RETFROM` to `SPELLINGS2`. Function names are also added to `SPELLINGS2` by `DEFINE`, `DEFINEQ`, `LOAD` (when loading compiled code), `UNSAVEDEF`, `EDITF`, and `PRETTYPRINT`.

SPELLINGS3

[Variable]

`SPELLINGS3` is a list of words used for spelling correction on all unbound atoms. `SPELLINGS3` is initialized to `EDITMACROS`, `BREAKMACROS`, `BROKENFNS`, and `ADVISEDFNS`. Whenever `LISPX` is given an atom to evaluate, the name of the atom is added to `SPELLINGS3` if the atom has a value. Atoms are also added to `SPELLINGS3` whenever they are edited by `EDITV`, and whenever they are set via `RPAQ` or `RPAQQ`. For example, when a file is loaded, all of the variables set in the file are added to `SPELLINGS3`. Atoms are also added to `SPELLINGS3` when they are set by a `LISPX` input, e.g., typing `(SETQ FOO (REVERSE (SETQ FIE . . .)))` will add both `FOO` and `FIE` to `SPELLINGS3`.

USERWORDS

[Variable]

`USERWORDS` is a list containing both functions and variables that you have *referred* to, e.g., by breaking or editing. `USERWORDS` is used for spelling correction by `ARGLIST`, `UNSAVEDEF`, `PRETTYPRINT`, `BREAK`, `EDITF`, `ADVISE`, etc. `USERWORDS` is initially `NIL`. Function names are added to it by `DEFINE`, `DEFINEQ`, `LOAD`, (when loading compiled code, or loading `exprs` to property lists) `UNSAVEDEF`, `EDITF`, `EDITV`, `EDITP`, `PRETTYPRINT`, etc. Variable names are added to `USERWORDS` at the same time as they are added to `SPELLINGS3`. In addition, the variable `LASTWORD` is always set to the last word added to `USERWORDS`, i.e., the last function or variable referred to by the user, and the respelling of `NIL` is defined to be the value of `LASTWORD`. Thus, if you had just defined a function, you can then prettyprint it by typing `PP()`.

Each of the above four spelling lists are divided into two sections separated by a special marker (the value of the variable `SPELLSTR1`). The first section contains the "permanent" words; the second section contains the temporary words. New words are added to the corresponding spelling list at the front of its temporary section (except that functions added to `SPELLINGS1` or `SPELLINGS2` by `LISPX` are always added to the end of the permanent section. If the word is already in the temporary section, it is moved to the front of that section; if the word is in the permanent section, no action is taken. If the length of the temporary section then exceeds a specified number, the last (oldest) word in the temporary section is forgotten, i.e., deleted. This procedure prevents the spelling lists from becoming cluttered with unimportant words that are no longer being used, and thereby slowing down spelling

INTERLISP-D REFERENCE MANUAL

DWIM

correction time. Since the spelling corrector usually moves each word selected as a respelling to the front of its spelling list, the word is thereby moved into the permanent section. Thus once a word is misspelled and corrected, it is considered important and will never be forgotten.

The spelling correction algorithm will not alter a spelling list unless it contains the special marker (the value of `SPELLSTR1`). This provides a way to ensure that a spelling list will not be altered.

<code>#SPELLINGS1</code>	[Variable]
<code>#SPELLINGS2</code>	[Variable]
<code>#SPELLINGS3</code>	[Variable]
<code>#USERWORDS</code>	[Variable]

The maximum length of the temporary section for `SPELLINGS1`, `SPELLINGS2`, `SPELLINGS3` and `USERWORDS` is given by the value of `#SPELLINGS1`, `#SPELLINGS2`, `#SPELLINGS3`, and `#USERWORDS`, initialized to 30, 30, 30, and 60 respectively.

You can alter these values to modify the performance behavior of spelling correction.

Generators for Spelling Correction

For some applications, it is more convenient to *generate* candidates for a respelling one by one, rather than construct a complete list of all possible candidates, e.g., spelling correction involving a large directory of files, or a natural language data base. For these purposes, *SPLST* can be an array (of any size). The first element of this array is the generator function, which is called with the array itself as its argument. Thus the function can use the remainder of the array to store "state" information, e.g., the last position on a file, a pointer into a data structure, etc. The value returned by the function is the next candidate for respelling. If `NIL` is returned, the spelling "list" is considered to be exhausted, and the closest match is returned. If a candidate is found with no disagreements, it is returned immediately without waiting for the "list" to exhaust.

SPLST can also be a generator, i.e. the value of the function `GENERATOR` (Chapter 11). The generator *SPLST* will be started up whenever the spelling corrector needs the next candidate, and it should return candidates via the function `PRODUCE`. For example, the following could be used as a "spelling list" which effectively contains all functions in the system:

```
[GENERATOR
  (MAPATOMS (FUNCTION (LAMBDA (X) (if (GETD X) then (PRODUCE
X]
```

Spelling Corrector Algorithm

The basic philosophy of DWIM spelling correction is to count the number of disagreements between two words, and use this number divided by the length of the longer of the two words as a measure of their relative disagreement. One minus this number is then the relative agreement or closeness. For example, `CONS` and `CONX` differ only in their last character. Such substitution errors count as one disagreement, so that the two words are in 75% agreement. Most calls to the spelling corrector specify a relative agreement of 70, so that a single substitution error is permitted in words of four characters

or longer. However, spelling correction on shorter words is possible since certain types of differences such as single transpositions are not counted as disagreements. For example, AND and NAD have a relative agreement of 100. Calls to the spelling corrector from DWIM use the value of FIXSPELLREL, which is initially 70. Note that by setting FIXSPELLREL to 100, only spelling corrections with "zero" mistakes, will be considered, e.g., transpositions, double characters, etc.

The central function of the spelling corrector is CHOOZ. CHOOZ takes as arguments: a word, a minimum relative agreement, a spelling list, and an optional functional argument, *XWORD*, *REL*, *SPLST*, and *FN* respectively.

CHOOZ proceeds down *SPLST* examining each word. Words not satisfying *FN* (if *FN* is non-NIL), or those obviously too long or too short to be sufficiently close to *XWORD* are immediately rejected. For example, if *REL* = 70, and *XWORD* is 5 characters long, words longer than 7 characters will be rejected.

Special treatment is necessary for words shorter than *XWORD*, since doubled letters are not counted as disagreements. For example, CONSSSS and CONS have a relative agreement of 100. CHOOZ handles this by counting the number of doubled characters in *XWORD* before it begins scanning *SPLST*, and taking this into account when deciding whether to reject shorter words.

If *TWORD*, the current word on *SPLST*, is not rejected, CHOOZ computes the number of disagreements between it and *XWORD* by calling a subfunction, SKOR.

SKOR operates by scanning both words from left to right one character at a time. SKOR operates on the list of character codes for each word. This list is computed by CHOOZ before calling SKOR. Characters are considered to agree if they are the same characters or appear on the same key (i.e., a shift mistake). The variable SPELLCASEARRAY is a CASEARRAY which is used to determine equivalence classes for this purpose. It is initialized to equivalence lowercase and upper case letters, as well as the standard key transitions: for example, 1 with !, 3 with #, etc.

If the first character in *XWORD* and *TWORD* do *not* agree, SKOR checks to see if either character is the same as one previously encountered, and not accounted-for at that time. (In other words, transpositions are not handled by lookahead, but by *lookback*.) A displacement of two or fewer positions is counted as a transposition; a displacement by more than two positions is counted as a disagreement. In either case, both characters are now considered as accounted for and are discarded, and SKORing continues.

If the first character in *XWORD* and *TWORD* do not agree, and neither agree with previously unaccounted-for characters, and *TWORD* has more characters remaining than *XWORD*, SKOR removes and saves the first character of *TWORD*, and continues by comparing the rest of *TWORD* with *XWORD* as described above. If *TWORD* has the same or fewer characters remaining than *XWORD*, the procedure is the same except that the character is removed from *XWORD*. In this case, a special check is first made to see if that character is equal to the *previous* character in *XWORD*, or to the *next* character in *XWORD*, i.e., a double character typo, and if so, the character is considered accounted-for, and not counted as a disagreement. In this case, the "length" of *XWORD* is also decremented. Otherwise making *XWORD*

INTERLISP-D REFERENCE MANUAL

DWIM

sufficiently long by adding double characters would make it be arbitrarily close to *TWORD*, e.g., *XXXXXX* would correct to *PP*.

When *SKOR* has finished processing both *XWORD* and *TWORD* in this fashion, the value of *SKOR* is the number of unaccounted-for characters, plus the number of disagreements, plus the number of transpositions, with two qualifications:

1. If both *XWORD* and *TWORD* have a character unaccounted-for in the same position, the two characters are counted only once, i.e., substitution errors count as only one disagreement, not two
2. If there are no unaccounted-for characters and no disagreements, transpositions are not counted.

This permits spelling correction on very short words, such as edit commands, e.g., *XRT->XTR*. Transpositions are also not counted when *FASTYPEFLG* = *T*, for example, *IPULX* and *IPLUS* will be in 80% agreement with *FASTYPEFLG* = *T*, only 60% with *FASTYPEFLG* = *NIL*. The rationale behind this is that transpositions are much more common for fast typists, and should not be counted as disagreements, whereas more deliberate typists are not as likely to combine transpositions and other mistakes in a single word, and therefore can use more conservative metric. *FASTYPEFLG* is initially *NIL*.

Spelling Corrector Functions and Variables

(**ADDSPELL** *X SPLST N*)

[Function]

Adds *X* to one of the spelling lists as determined by the value of *SPLST*:

<i>NIL</i>	Adds <i>X</i> to <i>USERWORDS</i> and to <i>SPELLINGS2</i> . Used by <i>DEFINEQ</i> .
0	Adds <i>X</i> to <i>USERWORDS</i> . Used by <i>LOAD</i> when loading <i>EXPRS</i> to property lists.
1	Adds <i>X</i> to <i>SPELLINGS1</i> (at end of permanent section). Used by <i>LISPX</i> .
2	Adds <i>X</i> to <i>SPELLINGS2</i> (at end of permanent section). Used by <i>LISPX</i> .
3	Adds <i>X</i> to <i>USERWORDS</i> and <i>SPELLINGS3</i> .
a spelling list	If <i>SPLST</i> is a spelling list, <i>X</i> is added to it. In this case, <i>N</i> is the (optional) length of the temporary section. If <i>X</i> is already on the spelling list, and in its temporary section, <i>ADDSPELL</i> moves <i>X</i> to the front of that section.

ADDSPELL sets *LASTWORD* to *X* when *SPLST* = *NIL*, 0 or 3.

If *X* is not a symbol, *ADDSPELL* takes no action.

Note that the various systems calls to ADDSPELL, e.g., from DEFINE, EDITF, LOAD, etc., can all be suppressed by setting or binding ADDSPELLFLG to NIL (see the DWIM Functions and Variables section above).

(**MISSPELLED?** *XWORD REL SPLST FLG TAIL FN*) [Function]

If *XWORD* = NIL or \$ (<esc>), MISSPELLED? prints = followed by the value of LASTWORD, and returns this as the respelling, without asking for approval. Otherwise, MISSPELLED? checks to see if *XWORD* is really misspelled, i.e., if *FN* applied to *XWORD* is true, or *XWORD* is already contained on *SPLST*. In this case, MISSPELLED? simply returns *XWORD*. Otherwise MISSPELLED? computes and returns (FIXSPELL *XWORD REL SPLST FLG TAIL FN*).

(**FIXSPELL** *XWORD REL SPLST FLG TAIL FN TIEFLG DONTMOVETOPFLG*) [Function]

The value of FIXSPELL is either the respelling of or NIL. If for some reason itself is on , then FIXSPELL aborts and calls ERROR!. If there is a possibility that is spelled correctly, MISSPELLED? should be used instead of FIXSPELL. FIXSPELL performs all of the interactions described earlier, including requesting your approval if necessary.

If *XWORD* = NIL or \$ (escape), the respelling is the value of LASTWORD, and no approval is requested.

If *XWORD* contains lowercase characters, and the corresponding uppercase word is correct, i.e. on *SPLST* or satisfies *FN*, the uppercase word is returned and no interaction is performed. If FIXSPELL.UPPERCASE.QUIET is NIL (the default), a warning "=XX" is printed when coercing from "xx" to "XX". If FIXSPELL.UPPERCASE.QUIET is non-NIL, no warning is given.

If *REL* = NIL, defaults to the value of FIXSPELLREL (initially 70).

If *FLG* = NIL, the correction is handled in type-in mode, i.e., approval is never requested, and *XWORD* is not typed. If *FLG* = T, *XWORD* is typed (before the =) and approval is requested if APPROVEFLG = T. If *FLG* = NO-MESSAGE, the correction is returned with no further processing. In this case, a run-on correction will be returned as a dotted pair of the two parts of the word, and a synonym correction as a list of the form (*WORD1 WORD2*), where *WORD1* is (the corrected version of) *XWORD*, and *WORD2* is the synonym. The effect of the function CHOOZ can be obtained by calling FIXSPELL with *FLG* = NO-MESSAGE.

If *TAIL* is not NIL, and the correction is successful, CAR of *TAIL* is replaced by the respelling (using /RPLACA).

FIXSPELL will attempt to correct misspellings caused by running two words together, if the global variable RUNONFLG is non-NIL (default is NIL). In this case, approval is always requested. When a run-on error is corrected, CAR of *TAIL* is replaced by the two words, and the value of FIXSPELL is the first one. For example, if FIXSPELL is called to correct the edit command (MOVE TO AFTERCOND 3 2) with *TAIL* = (AFTERCOND 3 2), *TAIL* would be changed to (AFTER COND 2 3), and FIXSPELL would return AFTER (subject to your approval where necessary). If *TAIL* = T, FIXSPELL will also perform run-

INTERLISP-D REFERENCE MANUAL

DWIM

on corrections, returning a dotted pair of the two words in the event the correction is of this type.

If *TIEFLG* = *NIL* and a tie occurs, i.e., more than one word on *SPLST* is found with the same degree of "closeness", *FIXSPELL* returns *NIL*, i.e., no correction. If *TIEFLG* = *PICKONE* and a tie occurs, the first word is taken as the correct spelling. If *TIEFLG* = *LIST*, the value of *FIXSPELL* is a list of the respellings (even if there is only one), and *FIXSPELL* will not perform any interaction with you, nor modify *TAIL*, the idea being that the calling program will handle those tasks. Similarly, if *TIEFLG* = *EVERYTHING*, a list of all candidates whose degree of closeness is above *REL* will be returned, regardless of whether some are better than others. No interaction will be performed.

If *DONTMOVETOPFLG* = *T* and a correction occurs, it will *not* be moved to the front of the spelling list. Also, the spelling list will not be altered unless it contains the special marker used to separate the temporary and permanent parts of the system spelling lists (the value of *SPELLSTR1*).

(**FNCHECK** *FN NOERRORFLG SPELLFLG PROPFLG TAIL*)

[Function]

The task of **FNCHECK** is to check whether *FN* is the name of a function and if not, to correct its spelling. If *FN* is the name of a function or spelling correction is successful, **FNCHECK** adds the (corrected) name of the function to *USERWORDS* using **ADDSPELL**, and returns it as its value.

Since **FNCHECK** is called by many low level functions such as **ARGLIST**, **UNSAVEDEF**, etc., spelling correction only takes place when *DWIMFLG* = *T*, so that these functions can operate in a small Interlisp system which does not contain *DWIM*.

NOERRORFLG informs **FNCHECK** whether or not the calling function wants to handle the unsuccessful case: if *NOERRORFLG* is *T*, **FNCHECK** simply returns *NIL*, otherwise it prints *fn NOT A FUNCTION* and generates a non-breaking error.

If *FN* does not have a definition, but does have an *EXPR* property, then spelling correction is not attempted. Instead, if *PROPFLG* = *T*, *FN* is considered to be the name of a function, and is returned. If *PROPFLG* = *NIL*, *FN* is *not* considered to be the name of a function, and *NIL* is returned or an error generated, depending on the value of *NOERRORFLG*.

FNCHECK calls **MISSPELLED?** to perform spelling correction, so that if *FN* = *NIL*, the value of *LASTWORD* will be returned. *SPELLFLG* corresponds to **MISSPELLED?**'s fourth argument, *FLG*. If *SPELLFLG* = *T*, approval will be asked if *DWIM* was enabled in *CAUTIOUS* mode, i.e., if *APPROVEFLG* = *T*. *TAIL* corresponds to the fifth argument to **MISSPELLED?**.

FNCHECK is currently used by **ARGLIST**, **UNSAVEDEF**, **PRETTYPRINT**, **BREAK0**, **BREAKIN**, **ADVISE**, and **CALLS**. For example, **BREAK0** calls **FNCHECK** with *NOERRORFLG* = *T* since if **FNCHECK** cannot produce a function, **BREAK0** wants to define a dummy one. **CALLS** however calls **FNCHECK** with *NOERRORFLG* = *NIL*, since it cannot operate without a function.

Many other system functions call `MISSPELLED?` or `FIXSPELL` directly. For example, `BREAK1` calls `FIXSPELL` on unrecognized atomic inputs before attempting to evaluate them, using as a spelling list a list of all break commands. Similarly, `LISPX` calls `FIXSPELL` on atomic inputs using a list of all `LISPX` commands. When `UNBREAK` is given the name of a function that is not broken, it calls `FIXSPELL` with two different spelling lists, first with `BROKENFNS`, and if that fails, with `USERWORDS`. `MAKEFILE` calls `MISSPELLED?` using `FILELST` as a spelling list. Finally, `LOAD`, `BCOMPL`, `BRECOMPILE`, `TCOMPL`, and `RECOMPILE` all call `MISSPELLED?` if their input file(s) won't open.

20. CLISP

The syntax of Lisp is very simple. It can be described concisely, but it makes Lisp difficult to read and write without tools. Unlike many languages, there are no reserved words in Lisp such as IF, THEN, FOR, DO, etc., nor reserved characters like +, -, =, ←, etc. The only components of the language are atoms and delimiters. This eliminates the need for parsers and precedence rules, and makes Lisp programs easy to manipulate. For example, a Lisp interpreter can be written in one or two pages of Lisp code. This makes Lisp the most suitable programming language for writing programs that deal with other programs as data.

Human language is based on more complicated structures and relies more on special words to carry the meaning. The definition of the factorial function looks like this in Lisp:

```
(COND ((ZEROP N) 1) (T (TIMES N (FACTORIAL ((SUB1 N))))))
```

This definition is easy to read for a machine but difficult to read for a human. CLISP is designed to make Interlisp programs easier to read and write. CLISP does this by translating various operators, conditionals, and iterative statements to Interlisp. For example, factorial can be written in CLISP:

```
(IF N = 0 THEN 1 ELSE N*(FACTORIAL N-1))
```

CLISP will translate this expression to the form in the example above. The translation will take place when the form is read so there are no performance penalties.

You should view CLISP as a shorthand for producing Lisp programs. CLISP makes a program easy to read and sometimes more compact.

CLISP is implemented via the error correction machinery in Interlisp (see Chapter 20). Any expression that Interlisp thinks is well-formed will never be seen by CLISP. This means that interpreted programs that do not use CLISP constructs do not pay for its availability by slower execution time. In fact, the Interlisp interpreter does not know about CLISP at all. When the interpreter finds an error it calls an error routine which in turn invokes the Do-What-I-Mean (DWIM) analyzer. The DWIM analyzer knows how to deal with CLISP expressions. If the expression in question turns out to be a CLISP construct, the translated form is returned to the interpreter. In addition, the original CLISP expression is modified so that it *becomes* the correctly translated Interlisp form. In this way, the analysis and translation are done only once.

Integrating CLISP into Medley makes possible Do-What-I-Mean features for CLISP constructs as well as for pure Lisp expressions. For example, if you have defined a function named GET-PARENT, CLISP would know not to attempt to interpret the form (GET-PARENT) as an arithmetic infix operation. (Actually, CLISP would never get to see this form, since it does not contain any errors.) If you mistakenly write (GET-PRAENT), CLISP would know you meant (GET-PARENT), and not

INTERLISP-D REFERENCE MANUAL

(DIFFERENCE GET PRAENT), by using the information that PARENT is not the name of a variable, and that GET-PARENT is the name of a user function whose spelling is "very close" to that of GET-PRAENT. Similarly, by using information about the program's environment not readily available to a preprocessor, CLISP can successfully resolve the following sorts of ambiguities:

1. (LIST X*FACT N), where FACT is the name of a variable, means (LIST (X*FACT) N).
2. (LIST X*FACT N), where FACT is *not* the name of a variable but instead is the name of a function, means (LIST X*(FACT N)), i.e., N is FACT's argument.
3. (LIST X*FACT(N)), FACT the name of a function (and not the name of a variable), means (LIST X*(FACT N)).
4. Cases 1, 2 and 3 with FACT misspelled!

The first expression is correct both from the standpoint of CLISP syntax and semantics so the change would be made notification. In the other cases, you would be informed or consulted about what was taking place. For example, suppose you write the expression (LIST X*FCCT N). Assume also that there was both a function named FACT and a variable named FCT.

1. You will first be asked if FCCT is a misspelling of FCT. If you say YES, the expression will be interpreted as (LIST (X*FCT) N). If you say NO, you will be asked if FCCT was a misspelling of FACT, i.e., if you intended X*FCCT N to mean X*(FACT N).
2. If you say YES to this question, the indicated transformation will be performed. If you say NO, the system will ask if X*FCCT should be treated as CLISP, since FCCT is not the name of a (bound) variable.
3. If you say YES, the expression will be transformed, if NO, it will be left alone, i.e., as (LIST X*FCCT N). Note that we have not even considered the case where X*FCCT is itself a misspelling of a variable name, e.g., a variable named XFCT (as with GET-PRAENT). This sort of transformation will be considered after you said NO to X*FCCT N -> X*(FACT N).

The question of whether X*FCCT should be treated as CLISP is important because Interlisp users may have programs that employ identifiers containing CLISP operators. Thus, if CLISP encounters the expression A/B in a context where either A or B are not the names of variables, it will ask you if A/B is intended to be CLISP, in case you really do have a free variable named A/B.

Note: Through the discussion above, we speak of CLISP or DWIM asking you. Actually, if you typed in the expression in question for immediate execution, you are simply informed of the transformation, on the grounds that you would prefer an occasional misinterpretation rather than being continuously bothered, especially since you can always retype what you intended if a mistake occurs, and ask the programmer's assistant to UNDO the effects of the mistaken operations if necessary. For transformations on expressions in your programs, you can tell CLISP whether you wish to operate in CAUTIOUS or TRUSTING mode. In the former case (most typical) you will be asked to approve

CLISP

transformations, in the latter, CLISP will operate as it does on type-in, i.e., perform the transformation after informing you.

CLISP can also handle parentheses errors caused by typing 8 or 9 for (or). (On most terminals, 8 and 9 are the lowercase characters for (and), i.e., (and 8 appear on the same key, as do) and 9.) For example, if you write `N*8FACTORIAL N-1`, the parentheses error can be detected and fixed before the infix operator `*` is converted to the Interlisp function `TIMES`. CLISP is able to distinguish this situation from cases like `N*8*X` meaning `(TIMES N 8 X)`, or `N*8X`, where `8X` is the name of a variable, again by using information about the programming environment. In fact, by integrating CLISP with DWIM, CLISP has been made sufficiently tolerant of errors that almost everything can be misspelled! For example, CLISP can successfully translate the definition of `FACTORIAL`:

```
(IFF N = 0 THENN1 ESLE N*8FACTTORIALNN-1)
```

to the corresponding `COND`, while making five spelling corrections and fixing the parenthesis error. CLISP also contains a facility for converting from Interlisp back to CLISP, so that after running the above incorrect definition of `FACTORIAL`, you could "clispify" the now correct version to obtain `(IF N = 0 THEN 1 ELSE N*(FACTORIAL N-1))`.

This sort of robustness prevails throughout CLISP. For example, the iterative statement permits you to say things like:

```
(FOR OLD X FROM M TO N DO (PRINT X) WHILE (PRIMEP X))
```

However, you can also write `OLD (X←M)`, `(OLD X←M)`, `(OLD (X←M))`, permute the order of the operators, e.g., `(DO PRINT X TO N FOR OLD X←M WHILE PRIMEP X)`, omit either or both sets of parentheses, misspell any or all of the operators `FOR`, `OLD`, `FROM`, `TO`, `DO`, or `WHILE`, or leave out the word `DO` entirely! And, of course, you can also misspell `PRINT`, `PRIMEP`, `M` or `N`! In this example, the only thing you could not misspell is the first `X`, since it specifies the *name* of the variable of iteration. The other two instances of `X` could be misspelled.

CLISP is well integrated into Medley. For example, the above iterative statement translates into an equivalent Interlisp form using `PROG`, `COND`, `GO`, etc. When the interpreter subsequently encounters this CLISP expression, it automatically obtains and evaluates the translation. Similarly, the compiler "knows" to compile the translated form. However, if you `PRETTYPRINT` your program, `PRETTYPRINT` "knows" to print the original CLISP at the corresponding point in your function. Similarly, when you edit your program, the editor keeps the translation invisible to you. If you modify the CLISP, the translation is automatically discarded and recomputed the next time the expression is evaluated.

In short, CLISP is not a language at all, but rather a system. It plays a role analagous to that of the programmer's assistant (Chapter 13). Whereas the programmer's assistant is an invisible intermediary agent between your console requests and the Interlisp executive, CLISP sits between your programs and the Interlisp interpreter.

INTERLISP-D REFERENCE MANUAL

Only a small effort has been devoted to defining the core syntax of CLISP. Instead, most of the effort has been concentrated on providing a facility which "makes sense" out of the input expressions using context information as well as built-in and acquired information about user and system programs. It has been said that communication is based on the intention of the speaker to produce an effect in the recipient. CLISP operates under the assumption that what you say is *intended* to represent a meaningful operation, and therefore tries very hard to make sense out of it. The motivation behind CLISP is not to provide you with many different ways of saying the same thing, but to enable you to worry less about the *syntactic* aspects of your communication with the system. In other words, it gives you a new degree of freedom by permitting you to concentrate more on the problem at hand, rather than on translation into a formal and unambiguous language.

DWIM and CLISP are invoked on iterative statements because CAR of the iterative statement is not the name of a function, and hence generates an error. If you define a function by the same name as an i.s. operator, e.g., WHILE, TO, etc., the operator will no longer have the CLISP interpretation when it appears as CAR of a form, although it will continue to be treated as an i.s. operator if it appears in the interior of an i.s. To alert you, a warning message is printed, e.g., (WHILE DEFINED, THEREFORE DISABLED IN CLISP).

CLISP Interaction with User

Syntactically and semantically well formed CLISP transformations are always performed without informing you. Other CLISP transformations described in the previous section, e.g., misspellings of operands, infix operators, parentheses errors, unary minus - binary minus errors, all follow the same protocol as other DWIM transformations (Chapter 19). That is, if DWIM has been enabled in TRUSTING mode, or the transformation is in an expression you typed in for immediate execution, your approval is not requested, but you are informed. However, if the transformation involves a user program, and DWIM was enabled in CAUTIOUS mode, you will be asked to approve. If you say NO, the transformation is not performed. Thus, in the previous section, phrases such as "one of these (transformations) succeeds" and "the transformation LAST-ELL -> LAST-EL would be found" etc., all mean if you are in CAUTIOUS mode and the error is in a program, the corresponding transformation will be performed only if you approve (or defaults by not responding). If you say NO, the procedure followed is the same as though the transformation had not been found. For example, if A*B appears in the function FOO, and B is not bound (and no other transformations are found) you would be asked A*B [IN FOO] TREAT AS CLISP ? (The waiting time on such interactions is three times as long as for simple corrections, i.e., 3*DWIMWAIT).

In certain situations, DWIM asks for approval even if DWIM is enabled in TRUSTING mode. For example, you are always asked to approve a spelling correction that might also be interpreted as a CLISP transformation, as in LAST-ELL -> LAST-EL.

If you approved, A*B would be transformed to (ITIMES A B), which would then cause a U.B.A.B. error in the event that the program was being run (remember the entire discussion also applies to DWIMifying). If you said NO, A*B would be left alone.

CLISP

If the value of `CLISPHELPFLG` = `NIL` (initially `T`), you will not be asked to approve any CLISP transformation. Instead, in those situations where approval would be required, the effect is the same as though you had been asked and said `NO`.

CLISP Character Operators

CLISP recognizes a number of special characters operators, both prefix and infix, which are translated into common expressions. For example, the character `+` is recognized to represent addition, so CLISP translates the symbol `A+B` to the form `(IPLUS A B)`. Note that CLISP is invoked, and this translation is made, only if an error occurs, such as an unbound atom error or an undefined function error for the perfectly legitimate symbol `A+B`. Therefore you may choose not to use these facilities with no penalty, similar to other CLISP facilities.

You have a lot of flexibility in using CLISP character operators. A list can always be substituted for a symbol, and vice versa, without changing the interpretation of a phrase. For example, if the value of `(FOO X)` is `A`, and the value of `(FIE Y)` is `B`, then `(LIST (FOO X) (FIE Y))` has the same value as `(LIST A+B)`. Note that the first expression is a list of *four* elements: the atom `"LIST"`, the list `"(FOO X)"`, the atom `"+"`, and the list `"(FIE X)"`, whereas the second expression, `(LIST A+B)`, is a list of only *two* elements: the symbol `"LIST"` and the symbol `"A+B"`. Since `(LIST (FOO X) (FIE Y))` is indistinguishable from `(LIST (FOO X) + (FIE Y))` because spaces before or after parentheses have no effect on the Interlisp `READ` program, to be consistent, extra spaces have no effect on atomic operands either. In other words, CLISP will treat `(LIST A+ B)`, `(LIST A +B)`, and `(LIST A + B)` the same as `(LIST A+B)`.

Note: CLISP does not use its own special `READ` program because this would require you to explicitly identify CLISP expressions, instead of being able to intermix Interlisp and CLISP.

<code>+</code>	[CLISP Operator]
<code>-</code>	[CLISP Operator]
<code>*</code>	[CLISP Operator]
<code>/</code>	[CLISP Operator]
<code>↑</code>	[CLISP Operator]

CLISP recognizes `+`, `-`, `*`, `/`, and `↑` as the normal arithmetic infix operators. The `-` is also recognized as the prefix operator, unary minus. These are converted to `PLUS`, `DIFFERENCE` (or in the case of unary minus, `MINUS`), `TIMES`, `QUOTIENT`, and `EXPT`.

Normally, CLISP uses the "generic" arithmetic functions `PLUS`, `TIMES`, etc. CLISP contains a facility for declaring which type of arithmetic is to be used, either by making a global declaration, or by separate declarations about individual functions or variables.

The usual precedence rules apply (although you can easily change them), i.e., `*` has higher precedence than `+` so that `A+B*C` is the same as `A+(B*C)`, and both `*` and `/` are lower than `↑` so that `2*X↑2` is the same as `2*(X↑2)`. Operators of the same precedence group from left to right, e.g., `A/B/C` is equivalent to `(A/B)/C`. Minus is binary whenever possible, i.e., except when it is the first operator in a list, as in `(-A)` or `(-A)`, or when it

INTERLISP-D REFERENCE MANUAL

immediately follows another operator, as in $A * B$. Note that grouping with parentheses can always be used to override the normal precedence grouping, or when you are not sure how a particular expression will parse. The complete order of precedence for CLISP operators is given below.

Note that $+$ in front of a number will disappear when the number is read, e.g., $(FOO \ X \ +2)$ is indistinguishable from $(FOO \ X \ 2)$. This means that $(FOO \ X \ +2)$ will not be interpreted as CLISP, or be converted to $(FOO \ (IPLUS \ X \ 2))$. Similarly, $(FOO \ X \ -2)$ will not be interpreted the same as $(FOO \ X \ -2)$. To circumvent this, always type a space between the $+$ or $-$ and a number if an infix operator is intended, e.g., write $(FOO \ X \ + \ 2)$.

=	[CLISP Operator]
GT	[CLISP Operator]
LT	[CLISP Operator]
GE	[CLISP Operator]
LE	[CLISP Operator]

These are infix operators for "Equal", "Greater Than", "Less Than", "Greater Than or Equal", and "Less Than or Equal".

GT, LT, GE, and LE are all affected by the same declarations as $+$ and $*$, with the initial default to use GREATERP and LESSP.

Note that only single character operators, e.g., $+$, $\leftarrow$, $=$, etc., can appear in the *interior* of an atom. All other operators must be set off from identifiers with spaces. For example, XLTY will not be recognized as CLISP. In some cases, DWIM will be able to diagnose this situation as a run-on spelling error, in which case after the atom is split apart, CLISP will be able to perform the indicated transformation.

A number of Lisp functions, such as EQUAL, MEMBER, AND, OR, etc., can also be treated as CLISP infix operators. New infix operators can be easily added (see the CLISP Internal Conventions section below). Spelling correction on misspelled infix operators is performed using CLISPINFIXSPLST as a spelling list.

AND is higher than OR, and both AND and OR are lower than the other infix operators, so $(X \ OR \ Y \ AND \ Z)$ is the same as $(X \ OR \ (Y \ AND \ Z))$, and $(X \ AND \ Y \ EQUAL \ Z)$ is the same as $(X \ AND \ (Y \ EQUAL \ Z))$. All of the infix predicates have lower precedence than Interlisp forms, since it is far more common to apply a predicate to two forms, than to use a Boolean as an argument to a function. Therefore, $(FOO \ X \ GT \ FIE \ Y)$ is translated as $((FOO \ X) \ GT \ (FIE \ Y))$, rather than as $(FOO \ (X \ GT \ (FIE \ Y)))$. However, you can easily change this.

:	[CLISP Operator]

$X:N$ extracts the N th element of the list X . $FOO:3$ specifies the third element of FOO , or $(CADDR \ FOO)$. If N is less than zero, this indicates elements counting from the end of the list; i.e. $FOO:-1$ is the last element of FOO . $:$ operators can be nested, so $FOO:1:2$ means the second element of the first element of FOO , or $(CADAR \ FOO)$.

CLISP

The `:` operator can also be used for extracting substructures of records (see Chapter 8). Record operations are implemented by replacing expressions of the form `X:FOO` by `(fetch FOO of X)`. Both lower- and uppercase are acceptable.

`:` is also used to indicate operations in the pattern match facility (see Chapter 12). `X: (& 'A -- 'B)` translates to `(match X with (& 'A -- 'B))`

[CLISP Operator]

In combination with `:`, a period can be used to specify the "data path" for record operations. For example, if `FOO` is a field of the `BAR` record, `X:BAR.FOO` is translated into `(fetch (BAR FOO) of X)`. Subrecord fields can be specified with multiple periods: `X:BAR.FOO.BAZ` translates into `(fetch (BAR FOO BAZ) of X)`.

Note: If a record contains fields with periods in them, `CLISPIFY` will not translate a record operation into a form using periods to specify the data path. For example, `CLISPIFY` will NOT translate `(fetch A.B of X)` into `X:A.B`.

[CLISP Operator]

`X:N`, returns the *N*th tail of the list *X*. For example, `FOO:3` is `(CDDDR FOO)`, and `FOO:-1` is `(LAST FOO)`.

[CLISP Operator]

`←` is used to indicate assignment. For example, `X←Y` translates to `(SETQ X Y)`. If *X* does not have a value, and is not the name of one of the bound variables of the function in which it appears, spelling correction is attempted. However, since this may simply be a case of assigning an initial value to a new free variable, `DWIM` will always ask for approval before making the correction.

In conjunction with `:` and `::`, `←` can also be used to perform a more general type of assignment, involving structure modification. For example, `X:2←Y` means "make the second element of *X* be *Y*", in Interlisp terms `(RPLACA (CDR X) Y)`. Note that the *value* of this operation is the value of `RPLACA`, which is `(CDR X)`, rather than *Y*. Negative numbers can also be used, e.g., `X:-2_Y`, which translates to `(RPLACA (NLEFT X 2) Y)`.

You can indicate you want `/RPLACA` and `/RPLACD` used (undoable version of `RPLACA` and `RPLACD`, see Chapter 13), or `FRPLACA` and `FRPLACD` (fast versions of `RPLACA` and `RPLACD`, see Chapter 3), by means of CLISP declarations. The initial default is to use `RPLACA` and `RPLACD`.

`←` is also used to indicate assignment in record operations (`X:FOO←Y` translates to `(replace FOO of X with Y)`), and pattern match operations (Chapter 12).

`←` has different precedence on the left from on the right. On the left, `←` is a "tight" operator, i.e., high precedence, so that `A+B←C` is the same as `A+(B←C)`. On the right, `←` has broader scope so that `A←B+C` is the same as `A←(B+C)`.

INTERLISP-D REFERENCE MANUAL

On type-in, $\$ \leftarrow FORM$ (where $\$$ is the escape key) is equivalent to set the "last thing mentioned", i.e., is equivalent to $(SET\ LASTWORD\ FORM)$ (see Chapter 20). For example, immediately after examining the value of `LONGVARIABLENAME`, you could set it by typing $\$ \leftarrow$ followed by a form.

Note that an atom of the form $X \leftarrow Y$, appearing at the top level of a `PROG`, will not be recognized as an assignment statement because it will be interpreted as a `PROG` label by the Interlisp interpreter, and therefore will not cause an error, so `DWIM` and `CLISP` will never get to see it. Instead, one must write $(X \leftarrow Y)$.

$<$ $>$	[CLISP Operator] [CLISP Operator]

Angle brackets are used in CLISP to indicate list construction. The appearance of a " $<$ " corresponds to a "(" and indicates that a list is to be constructed containing all the elements up to the corresponding " $>$ ". For example, $<A\ B\ <C>>$ translates to $(LIST\ A\ B\ (LIST\ C))$. $!$ can be used to indicate that the next expression is to be inserted in the list as a *segment*, e.g., $<A\ B\ !\ C>$ translates to $(CONS\ A\ (CONS\ B\ C))$ and $<! \ A\ !\ B\ C>$ to $(APPEND\ A\ B\ (LIST\ C))$. $!!$ is used to indicate that the next expression is to be inserted as a segment, and furthermore, all list structure to its right in the angle brackets is to be physically attached to it, e.g., $<!! \ A\ B>$ translates to $(NCONC1\ A\ B)$, and $<!! \ A\ !B\ !C>$ to $(NCONC\ A\ (APPEND\ B\ C))$. Not $(NCONC\ (APPEND\ A\ B)\ C)$, which would have the same value, but would attach C to B , and not attach either to A . Note that $<$, $!$, $!!$, and $>$ need not be separate atoms, for example, $<A\ B\ !\ C>$ may be written equally well as $<\ A\ B\ !\ C\ >$. Also, arbitrary Interlisp or CLISP forms may be used within angle brackets. For example, one can write $<FOO \leftarrow (FIE\ X)\ !\ Y>$ which translates to $(CONS\ (SETQ\ FOO\ (FIE\ X))\ Y)$. `CLISPIFY` converts expressions in `CONS`, `LIST`, `APPEND`, `NCONC`, `NCONC1`, `/NCONC`, and `/NCONC1` into equivalent CLISP expressions using $<$, $>$, $!$, and $!!$.

Note: brackets differ from other CLISP operators. For example, $<A\ B\ 'C>$ translates to $(LIST\ A\ B\ (QUOTE\ C))$ even though following $'$, all operators are ignored for the rest of the identifier. (This is true only if a previous unmatched $<$ has been seen, e.g., $(PRINT\ 'A>B)$ will print the atom $A>B$.) Note however that $<A\ B\ ' \ C>\ D>$ is equivalent to $(LIST\ A\ B\ (QUOTE\ C>) \ D)$.

$'$	[CLISP Operator]

CLISP recognizes $'$ as a prefix operator. $'$ means `QUOTE` when it is the first character in an identifier, and is ignored when it is used in the interior of an identifier. Thus, $X = 'Y$ means $(EQ\ X\ (QUOTE\ Y))$, but $X = CAN'T$ means $(EQ\ X\ CAN'T)$, not $(EQ\ X\ CAN)$ followed by $(QUOTE\ T)$. This enables users to have variable and function names with $'$ in them (so long as the $'$ is not the first character).

Following $'$, all operators are ignored for the rest of the identifier, e.g., $'*A$ means $(QUOTE\ *A)$, and $'X=Y$ means $(QUOTE\ X=Y)$, not $(EQ\ (QUOTE\ X)\ Y)$. To write $(EQ$

CLISP

(QUOTE X) Y), one writes Y='X, or 'X =Y. This is one place where an extra space does make a difference.

On type-in, '\$ (escape) is equivalent to (QUOTE VALUE-OF-LASTWORD) (see Chapter 19). For example, after calling PRETTYPRINT on LONGFUNCTION, you could move its definition to FOO by typing (MOVD '\$ 'FOO).

Note that this is not (MOVD \$ 'FOO), which would be equivalent to (MOVD LONGFUNCTION 'FOO), and would (probably) cause a U.B.A. LONGFUNCTION error, nor (MOVD (\$ FOO), which would actually move the definition of \$ to FOO, since DWIM and the spelling corrector would never be invoked.

~

[CLISP Operator]

CLISP recognizes ~ as a prefix operator meaning NOT. ~ can negate a form, as in ~(ASSOC X Y), or ~X, or negate an infix operator, e.g., (A ~GT B) is the same as (A LEQ B). Note that ~A = B means (EQ (NOT A) B).

When ~ negates an operator, e.g., ~=:, ~LT, the two operators are treated as a single operator whose precedence is that of the second operator. When ~ negates a function, e.g., (~FOO X Y), it negates the whole form, i.e., (~ (FOO X Y)).

Order of Precedence of CLISP Operators:

,
:
← (left precedence)
- (unary), ~
↑
*, /
+, - (binary)
← (right precedence)
=

Interlisp forms

LT, GT, EQUAL, MEMBER, etc.
AND
OR
IF, THEN, ELSEIF, ELSE
iterative statement operators

Declarations

CLISP declarations are used to affect the choice of Interlisp function used as the translation of a particular operator. For example, A+B can be translated as either (PLUS A B), (FPLUS A B), or (IPLUS A B), depending on the declaration in effect. Similarly X:1←Y can mean (RPLACA X Y),

INTERLISP-D REFERENCE MANUAL

(FRPLACA X Y), or (/RPLACA X Y), and <!! A B> either (NCONC1 A B) or (/NCONC1 A B). Note that the choice of function on all CLISP transformations are affected by the CLISP declaration in effect, i.e., iterative statements, pattern matches, record operations, as well as infix and prefix operators.

(**CLISPDEC** *DECLST*)

[Function]

Puts into effect the declarations in *DECLST*. CLISPDEC performs spelling corrections on words not recognized as declarations. CLISPDEC is undoable.

You can make (changes) a global declaration by calling CLISPDEC with *DECLST* a list of declarations, e.g., (CLISPDEC '(FLOATING UNDOABLE)). Changing a global declaration does not affect the speed of subsequent CLISP transformations, since all CLISP transformations are table driven (i.e., property list), and global declarations are accomplished by making the appropriate internal changes to CLISP at the time of the declaration. If a function employs *local* declarations (described below), there will be a slight loss in efficiency owing to the fact that for each CLISP transformation, the declaration list must be searched for possibly relevant declarations.

Declarations are implemented in the order that they are given, so that later declarations override earlier ones. For example, the declaration FAST specifies that FRPLACA, FRPLACD, FMEMB, and FLAST be used in place of RPLACA, RPLACD, MEMB, and LAST; the declaration RPLACA specifies that RPLACA be used. Therefore, the declarations (FAST RPLACA RPLACD) will cause FMEMB, FLAST, RPLACA, and RPLACD to be used.

The initial global declaration is MIXED and STANDARD.

The table below gives the declarations available in CLISP, and the Interlisp functions they indicate:

Declaration:	Interlisp Functions to be used:
MIXED	PLUS, MINUS, DIFFERENCE, TIMES, QUOTIENT, LESSP, GREATERP
INTEGER or FIXED	IPLUS, IMINUS, IDIFFERENCE, ITIMES, IQUOTIENT, ILESSP, IGREATERP
FLOATING	FPLUS, FMINUS, FDIFFERENCE, FTIMES, FQUOTIENT, LESSP, FGREATERP
FAST	FRPLACA, FRPLACD, FMEMB, FLAST, FASSOC
UNDOABLE	/RPLACA, /RPLACD, /NCONC, /NCONC1, /MAPCONC, /MAPCON
STANDARD	RPLACA, RPLACD, MEMB, LAST, ASSOC, NCONC, NCONC1, MAPCONC, MAPCON
RPLACA, RPLACD, /RPLACA, etc.	corresponding function

CLISP

You can also make local declarations affecting a selected function or functions by inserting an expression of the form `(CLISP: . DECLARATIONS)` immediately following the argument list, i.e., as `CADDR` of the definition. Such local declarations take precedence over global declarations. Declarations affecting selected variables can be indicated by lists, where the first element is the name of a variable, and the rest of the list the declarations for that variable. For example, `(CLISP: FLOATING (X INTEGER))` specifies that in this function integer arithmetic be used for computations involving `X`, and floating arithmetic for all other computations, where "involving" means where the variable itself is an operand. For example, with the declaration `(FLOATING (X INTEGER))` in effect, `(FOO X)+(FIE X)` would translate to `FPLUS`, i.e., use floating arithmetic, even though `X` appears somewhere inside of the operands, whereas `X+(FIE X)` would translate to `IPLUS`. If there are declarations involving *both* operands, e.g., `X+Y`, with `(X FLOATING) (Y INTEGER)`, whichever appears first in the declaration list will be used.

You can also make local record declarations by inserting a record declaration, e.g., `(RECORD --)`, `(ARRAYRECORD --)`, etc., in the local declaration list. In addition, a local declaration of the form `(RECORDS A B C)` is equivalent to having copies of the global declarations `A`, `B`, and `C` in the local declaration. Local record declarations override global record declarations for the function in which they appear. Local declarations can also be used to override the global setting of certain DWIM/CLISP parameters effective only for transformations within that function, by including in the local declaration an expression of the form `(VARIABLE = VALUE)`, e.g., `(PATVARDEFAULT = QUOTE)`.

The `CLISP:` expression is converted to a comment of a special form recognized by CLISP. Whenever a CLISP transformation that is affected by declarations is about to be performed in a function, this comment will be searched for a relevant declaration, and if one is found, the corresponding function will be used. Otherwise, if none are found, the global declaration(s) currently in effect will be used.

Local declarations are effective in the order that they are given, so that later declarations can be used to override earlier ones, e.g., `(CLISP: FAST RPLACA RPLACD)` specifies that `FMEMB`, `FLAST`, `RPLACA`, and `RPLACD` be used. An exception to this is that declarations for specific variables take precedence of general, function-wide declarations, regardless of the order of appearance, as in `(CLISP: (X INTEGER) FLOATING)`.

`CLISPIFY` also checks the declarations in effect before selecting an infix operator to ensure that the corresponding CLISP construct would in fact translate back to this form. For example, if a `FLOATING` declaration is in effect, `CLISPIFY` will convert `(FPLUS X Y)` to `X+Y`, but leave `(IPLUS X Y)` as is. If `(FPLUS X Y)` is `CLISPIFY`ed while a `FLOATING` declaration is under effect, and then the declaration is changed to `INTEGER`, when `X+Y` is translated back to Interlisp, it will become `(IPLUS X Y)`.

CLISP Operation

CLISP is a part of the basic Medley system. Without any special preparations, you can include CLISP constructs in programs, or type them in directly for evaluation (in `EVAL` or `APPLY` format), then, when

INTERLISP-D REFERENCE MANUAL

the "error" occurs, and DWIM is called, it will destructively transform the CLISP to the equivalent Interlisp expression and evaluate the Interlisp expression. CLISP transformations, like all DWIM corrections, are undoable. User approval is not requested, and no message is printed. This entire discussion also applies to CLISP transformation initiated by calls to DWIM from DWIMIFY.

However, if a CLISP construct contains an error, an appropriate diagnostic is generated, and the form is left unchanged. For example, if you write `(LIST X+Y*)`, the error diagnostic `MISSING OPERAND AT X+Y* IN (LIST X+Y*)` would be generated. Similarly, if you write `(LAST+EL X)`, CLISP knows that `((IPLUS LAST EL) X)` is not a valid Interlisp expression, so the error diagnostic `MISSING OPERATOR IN (LAST+EL X)` is generated. (For example, you might have meant to say `(LAST+EL*X)`.) If `LAST+EL` were the name of a defined function, CLISP would never see this form.

Since the bad CLISP transformation might not be CLISP at all, for example, it might be a misspelling of a user function or variable, DWIM holds all CLISP error messages until after trying other corrections. If one of these succeeds, the CLISP message is discarded. Otherwise, if all fail, the message is printed (but no change is made). For example, suppose you type `(R/PLACA X Y)`. CLISP generates a diagnostic, since `((IQUOTIENT R PLACA) X Y)` is obviously not right. However, since `R/PLACA` spelling corrects to `/RPLACA`, this diagnostic is never printed.

Note: CLISP error messages are not printed on type-in. For example, typing `X+*Y` will just produce a `U.B.A. X+*Y` message.

If a CLISP infix construct is well formed from a syntactic standpoint, but one or both of its operands are atomic and not bound, it is possible that either the operand is misspelled, e.g., you wrote `X+YY` for `X+Y`, or that a CLISP transformation operation was not intended at all, but that the entire expression is a misspelling. For the purpose of DWIMIFYing, "not bound" means no top level value, not on list of bound variables built up by DWIMIFY during its analysis of the expression, and not on `NOFIXVARSLST`, i.e., not previously seen.

For example, if you have a variable named `LAST-EL`, and write `(LIST LAST-ELL)`. Therefore, CLISP computes, but does not actually perform, the indicated infix transformation. DWIM then continues, and if it is able to make another correction, does so, and ignores the CLISP interpretation. For example, with `LAST-ELL`, the transformation `LAST-ELL -> LAST-EL` would be found.

If no other transformation is found, and DWIM is about to interpret a construct as CLISP for which one of the operands is not bound, DWIM will ask you whether CLISP was intended, in this case by printing `LAST-ELL TREAT AS CLISP ?`.

Note: If more than one infix operator was involved in the CLISP construct, e.g., `X+Y+Z`, or the operation was an assignment to a variable already noticed, or `TREATASCLISPFLG` is `T` (initially `NIL`), you will simply be informed of the correction, e.g., `X+Y+Z TREATED AS CLISP`. Otherwise, even if DWIM was enabled in `TRUSTING` mode, you will be asked to approve the correction.

The same sort of procedure is followed with 8 and 9 errors. For example, suppose you write `FOO8*X` where `FOO8` is not bound. The CLISP transformation is noted, and DWIM proceeds. It next asks you

CLISP

to approve `FOO8*X -> FOO ( *X`. For example, this would make sense if you have (or plan to define) a function named `*X`. If you refuse, you are asked whether `FOO8*X` is to be treated as CLISP. Similarly, if `FOO8` were the name of a variable, and you write `FOOO8*X`, you will first be asked to approve `FOOO8*X -> FOOO ( XX`, and if you refuse, then be offered the `FOOO8 -> FOO8` correction. The 8-9 transformation is tried before spelling correction since it is empirically more likely that an unbound atom or undefined function containing an 8 or a 9 is a parenthesis error, rather than a spelling error.

CLISP also contains provision for correcting misspellings of infix operators (other than single characters), IF words, and i.s. operators. This is implemented in such a way that the user who does not misspell them is not penalized. For example, if you write `IF N = 0 THEN 1 ELSSE N*(FACT N-1)` CLISP does *not* operate by checking each word to see if it is a misspelling of IF, THEN, ELSE, or ELSEIF, since this would seriously degrade CLISP's performance on *all* IF statements. Instead, CLISP assumes that all of the IF words are spelled correctly, and transforms the expression to `(COND ((ZEROP N) 1 ELSSE N*(FACT N-1)))`. Later, after DWIM cannot find any other interpretation for ELSSE, and using the fact that this atom originally appeared in an IF statement, DWIM attempts spelling correction, using `(IF THEN ELSE ELSEIF)` for a spelling list. When this is successful, DWIM "fails" all the way back to the original IF statement, changes ELSSE to ELSE, and starts over. Misspellings of AND, OR, LT, GT, etc. are handled similarly.

CLISP also contains many Do-What-I-Mean features besides spelling corrections. For example, the form `(LIST +X Y)` would generate a MISSING OPERATOR error. However, `(LIST -X Y)` makes sense, if the minus is unary, so DWIM offers this interpretation to you. Another common error, especially for new users, is to write `(LIST X*FOO(Y))` or `(LIST X*FOO Y)`, where FOO is the name of a function, instead of `(LIST X*(FOO Y))`. Therefore, whenever an operand that is not bound is also the name of a function (or corrects to one), the above interpretations are offered.

CLISP Translations

The translation of CLISP character operators and the CLISP word IF are handled by *replacing* the CLISP expression with the corresponding Interlisp expression, and discarding the original CLISP. This is done because (1) the CLISP expression is easily recomputable (by CLISPIFY) and (2) the Interlisp expressions are simple and straightforward. Another reason for discarding the original CLISP is that it may contain errors that were corrected in the course of translation (e.g., `FOO←FOOO:1, N*8FOO X`), etc.). If the original CLISP were retained, either you would have to go back and fix these errors by hand, thereby negating the advantage of having DWIM perform these corrections, or else DWIM would have to keep correcting these errors over and over.

Note that CLISPIFY is sufficiently fast that it is practical for you to configure your Interlisp system so that all expressions are automatically CLISPIFYed immediately before they are presented to you. For example, you can define an edit macro to use in place of P which calls CLISPIFY on the current expression before printing it. Similarly, you can inform PRETTYPRINT to call CLISPIFY on each expression before printing it, etc.

INTERLISP-D REFERENCE MANUAL

Where (1) or (2) are not the case, e.g., with iterative statements, pattern matches, record expressions, etc. the original CLISP is retained (or a slightly modified version thereof), and the translation is stored elsewhere (by the function `CLISPTRAN`, in the Miscellaneous Functions and Variables), usually in the hash array `CLISPARRAY`. The interpreter automatically checks this array when given a form `CAR` of which is not a function. Similarly, the compiler performs a `GETHASH` when given a form it does not recognize to see if it has a translation, which is then compiled instead of the form. Whenever you *change* a CLISP expression by editing it, the editor automatically deletes its translation (if one exists), so that the next time it is evaluated or `DWIMIFIED`, the expression will be retranslated (if the value of `CLISPTRANFLG` is `T`, `DWIMIFY` will also (re)translate any expressions which have translations stored remotely, see the `CLISPIFY` section). The function `PPT` and the edit commands `PPT` and `CLISP:` are available for examining translations (see the Miscellaneous Functions and Variables section).

You can also indicate that you want the original CLISP retained by embedding it in an expression of the form `(CLISP . CLISP-EXPRESSION)`, e.g., `(CLISP X:5:3)` or `(CLISP <A B C ! D>)`. In such cases, the translation will be stored remotely as described above. Furthermore, such expressions will be treated as CLISP even if infix and prefix transformations have been disabled by setting `CLISPFLG` to `NIL` (see the Miscellaneous Functions and Variables section). In other words, you can instruct the system to interpret as CLISP infix or prefix constructs only those expressions that are specifically flagged as such. You can also include CLISP declarations by writing `(CLISP DECLARATIONS . FORM)`, e.g., `(CLISP (CLISP: FLOATING) ...)`. These declarations will be used in place of any CLISP declarations in the function definition. This feature provides a way of including CLISP declarations in macro definitions.

Note: CLISP translations can also be used to supply an interpretation for function objects, as well as forms, either for function objects that are used openly, i.e., appearing as `CAR` of form, function objects that are explicitly `APPLIED`, as with arguments to mapping functions, or function objects contained in function definition cells. In all cases, if `CAR` of the object is not `LAMBDA` or `NLAMBDA`, the interpreter and compiler will check `CLISPARRAY`.

DWIMIFY

`DWIMIFY` is effectively a preprocessor for CLISP. `DWIMIFY` operates by scanning an expression as though it were being interpreted, and for each form that would generate an error, calling `DWIM` to "fix" it. `DWIMIFY` performs *all* DWIM transformations, not just CLISP transformations, so it does spelling correction, fixes 8-9 errors, handles `F/L`, etc. Thus you will see the same messages, and be asked for approval in the same situations, as you would if the expression were actually run. If `DWIM` is unable to make a correction, no message is printed, the form is left as it was, and the analysis proceeds.

`DWIMIFY` knows exactly how the interpreter works. It knows the syntax of `PROGs`, `SELECTQs`, `LAMBDA` expressions, `SETQs`, et al. It knows how variables are bound, and that the argument of `NLAMBDA`s are not evaluated (you can inform `DWIMIFY` of a function or macro's nonstandard binding or evaluation by giving it a suitable `INFO` property, see below). In the course of its analysis of a

CLISP

particular expression, DWIMIFY builds a list of the bound variables from the LAMBDA expressions and PROGS that it encounters. It uses this list for spelling corrections. DWIMIFY also knows not to try to "correct" variables that are on this list since they would be bound if the expression were actually being run. However, note that DWIMIFY cannot, a priori, know about variables that are used freely but would be bound in a higher function if the expression were evaluated in its normal context. Therefore, DWIMIFY will try to "correct" these variables. Similarly, DWIMIFY will attempt to correct forms for which CAR is undefined, even when the form is not in error from your standpoint, but the corresponding function has simply not yet been defined.

Note: DWIMIFY rebinds FIXSPELLDEFAULT to N, so that if you are not at the terminal when DWIMIFYing (or compiling), spelling corrections will not be performed.

DWIMIFY will also inform you when it encounters an expression with too *many* arguments (unless DWIMCHECK#ARGSFLG = NIL), because such an occurrence, although does not cause an error in the Interlisp interpreter, nevertheless is frequently symptomatic of a parenthesis error. For example, if you wrote (CONS (QUOTE FOO X)) instead of (CONS (QUOTE FOO) X), DWIMIFY will print:

```
POSSIBLE PARENTHESIS ERROR IN
(QUOTE FOO X)
TOO MANY ARGUMENTS (MORE THAN 1)
```

DWIMIFY will also check to see if a PROG label contains a clisp character (unless DWIMCHECKPROGLABELSFLG = NIL, or the label is a member of NOFIXVARSLST), and if so, will alert you by printing the message SUSPICIOUS PROG LABEL, followed by the label. The PROG label will *not* be treated as CLISP.

Note that in most cases, an attempt to transform a form that is already as you intended will have no effect (because there will be nothing to which that form could reasonably be transformed). However, in order to avoid needless calls to DWIM or to avoid possible confusion, you can inform DWIMIFY *not* to attempt corrections or transformations on certain functions or variables by adding them to the list NOFIXFNSLST or NOFIXVARSLST respectively. Note that you could achieve the same effect by simply setting the corresponding variables, and giving the functions dummy definitions.

DWIMIFY will never attempt corrections on global variables, i.e., variables that are a member of the list GLOBALVARS, or have the property GLOBALVAR with value T, on their property list. Similarly, DWIMIFY will not attempt to correct variables declared to be SPECVARS in block declarations or via DECLARE expressions in the function body. You can also declare variables that are simply used freely in a function by using the USEDFREE declaration.

DWIMIFY and DWIMIFYFNS (used to DWIMIFY several functions) maintain two internal lists of those functions and variables for which corrections were unsuccessfully attempted. These lists are initialized to the values of NOFIXFNSLST and NOFIXVARSLST. Once an attempt is made to fix a particular function or variable, and the attempt fails, the function or variable is added to the corresponding list, so that on subsequent occurrences (within this call to DWIMIFY or DWIMIFYFNS), no attempt at correction is made. For example, if FOO calls FIE several times, and FIE is undefined at the time FOO is DWIMIFYed, DWIMIFY will not bother with FIE after the first occurrence. In other words, once DWIMIFY "notices" a function or variable, it no longer attempts to correct it. DWIMIFY

INTERLISP-D REFERENCE MANUAL

and DWIMIFYFNS also "notice" free variables that are set in the expression being processed. Moreover, once DWIMIFY "notifies" such functions or variables, it subsequently treats them the same as though they were actually defined or set.

Note that these internal lists are local to each call to DWIMIFY and DWIMIFYFNS, so that if a function containing FOOO, a misspelled call to FOO, is DWIMIFYed before FOO is defined or mentioned, if the function is DWIMIFYed again after FOO has been defined, the correction will be made.

You can undo selected transformations performed by DWIMIFY, as described in Chapter 13.

(DWIMIFY *X QUIETFLG L*) [Function]

Performs all DWIM and CLISP corrections and transformations on *X* that would be performed if *X* were run, and prints the result unless *QUIETFLG* = T.

If *X* is an atom and *L* is NIL, *X* is treated as the name of a function, and its entire definition is DWIMIFYed. If *X* is a list or *L* is not NIL, *X* is the expression to be DWIMIFYed. If *L* is not NIL, it is the edit push-down list leading to *X*, and is used for determining context, i.e., what bound variables would be in effect when *X* was evaluated, whether *X* is a form or sequence of forms, e.g., a COND clause, etc.

If *X* is an iterative statement and *L* is NIL, DWIMIFY will also print the translation, i.e., what is stored in the hash array.

(DWIMIFYFNS *FN . . . FN*) [NLambda NoSpread Function]

DWIMIFYs each of the functions given. If only one argument is given, it is evaluated. If its value is a list, the functions on this list are DWIMIFYed. If only one argument is given, it is atomic, its value is not a list, and it is the name of a known file, DWIMIFYFNS will operate on (FILEFNSLST *FN*), e.g. (DWIMIFYFNS FOO.LSP) will DWIMIFY every function in the file FOO.LSP.

Every 30 seconds, DWIMIFYFNS prints the name of the function it is processing, a la PRETTYPRINT.

Value is a list of the functions DWIMIFYed.

DWIMINMACROSFLG [Variable]

Controls how DWIMIFY treats the arguments in a "call" to a macro, i.e., where the CAR of the form is undefined, but has a macro definition. If DWIMINMACROSFLG is T, then macros are treated as LAMBDA functions, i.e., the arguments are assumed to be evaluated, which means that DWIMIFY will descend into the argument list. If DWIMINMACROSFLG is NIL, macros are treated as NLAMBDA functions. DWIMINMACROSFLG is initially T.

INFO [Property Name]

Used to inform DWIMIFY of nonstandard behavior of particular forms with respect to evaluation, binding of arguments, etc. The INFO property of a symbol is a single atom or list of atoms chosen from among the following:

CLISP

- EVAL** Informs DWIMIFY (and CLISP and Masterscope) that an nlambda function *does* evaluate its arguments. Can also be placed on a macro name to override the behavior of DWIMINMACROSFLG = NIL.
- NOEVAL** Informs DWIMIFY that a macro does *not* evaluate all of its arguments, even when DWIMINMACROSFLG = T.
- BINDS** Placed on the INFO property of a function or the CAR of a special form to inform DWIMIFY that the function or form binds variables. In this case, DWIMIFY assumes that CADDR of the form is the variable list, i.e., a list of symbols, or lists of the form (VAL VALUE). LAMBDA, NLAMBDA, PROG, and RESETVARS are handled in this fashion.
- LABELS** Informs CLISPIFY that the form interprets top-level symbols as labels, so that CLISPIFY will never introduce an atom (by packing) at the top level of the expression. PROG is handled in this fashion.

NOFIXFNSLST [Variable]

List of functions that DWIMIFY will not try to correct.

NOFIXVARSLST [Variable]

List of variables that DWIMIFY will not try to correct.

NOSPELLFLG [Variable]

If T, DWIMIFY will not perform any spelling corrections. Initially NIL. NOSPELLFLG is reset to T when compiling functions whose definitions are obtained from a file, as opposed to being in core.

CLISPHELPPFLG [Variable]

If NIL, DWIMIFY will not ask you for approval of any CLISP transformations. Instead, in those situations where approval would be required, the effect is the same as though you had been asked and said NO. Initially T.

DWIMIFYCOMPFLG [Variable]

If T, DWIMIFY is called before compiling an expression. Initially NIL.

DWIMCHECK#ARGSFLG [Variable]

If T, causes DWIMIFY to check for too many arguments in a form. Initially T.

DWIMCHECKPROGLABELSFLG [Variable]

If T, causes DWIMIFY to check whether a PROG label contains a CLISP character. Initially T.

INTERLISP-D REFERENCE MANUAL

DWIMESSGAG

[Variable]

If T, suppresses all DWIMIFY error messages. Initially NIL.

CLISPRETRANFLG

[Variable]

If T, informs DWIMIFY to (re)translate all expressions which have remote translations in the CLISP hash array. Initially NIL.

CLISPIFY

CLISPIFY converts Interlisp expressions to CLISP. Note that the expression given to CLISPIFY need *not* have originally been input as CLISP, i.e., CLISPIFY can be used on functions that were written before CLISP was even implemented. CLISPIFY is cognizant of declaration rules as well as all of the precedence rules. For example, CLISPIFY will convert (IPLUS A (ITIMES B C)) into A+B*C, but (ITIMES A (IPLUS B C)) into A*(B+C). CLISPIFY handles such cases by first DWIMIFYing the expression. CLISPIFY also knows how to handle expressions consisting of a mixture of Interlisp and CLISP, e.g., (IPLUS A B*C) is converted to A+B*C, but (ITIMES A B+C) to (A*(B+C)). CLISPIFY converts calls to the six basic mapping functions, MAP, MAPC, MAPCAR, MAPLIST, MAPCONC, and MAPCON, into equivalent iterative statements. It also converts certain easily recognizable internal PROG loops to the corresponding iterative statements. CLISPIFY can convert all iterative statements input in CLISP back to CLISP, regardless of how complicated the translation was, because the original CLISP is saved.

CLISPIFY is not destructive to the original Interlisp expression, i.e., CLISPIFY produces a new expression without changing the original. The new expression may however contain some "pieces" of the original, since CLISPIFY attempts to minimize the number of CONSES by not copying structure whenever possible.

CLISPIFY will not convert expressions appearing as arguments to NLAMBDA functions, except for those functions whose INFO property is or contains the atom EVAL. CLISPIFY also contains built in information enabling it to process special forms such as PROG, SELECTQ, etc. If the INFO property is or contains the atom LABELS, CLISPIFY will never create an atom (by packing) at the top level of the expression. PROG is handled in this fashion.

Note: Disabling a CLISP operator with CLDISABLE (see the Miscellaneous Functions and Variables section) will also disable the corresponding CLISPIFY transformation. Thus, if ← is "turned off", A←B will not transform to (SETQ A B), nor vice versa.

(CLISPIFY X EDITCHAIN)

[Function]

Clispifies X. If X is an atom and EDITCHAIN is NIL, X is treated as the name of a function, and its definition (or EXPR property) is clispified. After CLISPIFY has finished, X is redefined (using /PUTD) with its new CLISP definition. The value of CLISPIFY is X. If X

CLISP

is atomic and not the name of a function, spelling correction is attempted. If this fails, an error is generated.

If *X* is a list, or *EDITCHAIN* is not NIL, *X* itself is the expression to be clispified. If *EDITCHAIN* is not NIL, it is the edit push-down list leading to *X* and is used to determine context as with *DWIMIFY*, as well as to obtain the local declarations, if any. The value of *CLISPIFY* is the clispified version of *X*.

(CLISPIFYFNS FN ... FN) [NLambda NoSpread Function]

Like *DWIMIFYFNS* except calls *CLISPIFY* instead of *DWIMIFY*.

CL:FLG [Variable]

Affects *CLISPIFY*'s handling of forms beginning with *CAR*, *CDR*, ... *CDDDDR*, as well as pattern match and record expressions. If *CL:FLG* is NIL, these are not transformed into the equivalent *:* expressions. This will prevent *CLISPIFY* from constructing any expression employing a *:* infix operator, e.g., (*CADR X*) will not be transformed to *X:2*. If *CL:FLG* is T, *CLISPIFY* will convert to *:* notation only when the argument is atomic or a simple list (a function name and one atomic argument). If *CL:FLG* is ALL, *CLISPIFY* will convert to *:* expressions whenever possible.

CL:FLG is initially T.

CLREMPARSFLG [Variable]

If T, *CLISPIFY* will remove parentheses in certain cases from simple forms, where "simple" means a function name and one or two atomic arguments. For example, (*COND* ((*ATOM X*) --)) will *CLISPIFY* to (*IF ATOM X THEN --*). However, if *CLREMPARSFLG* is set to NIL, *CLISPIFY* will produce (*IF (ATOM X) THEN --*). Regardless of the flag setting, the expression can be input in either form.

CLREMPARSFLG is initially NIL.

CLISPIFYPACKFLG [Variable]

CLISPIFYPACKFLG affects the treatment of infix operators with atomic operands. If *CLISPIFYPACKFLG* is T, *CLISPIFY* will pack these into single atoms, e.g., (*IPLUS A (ITIMES B C)*) becomes *A+B*C*. If *CLISPIFYPACKFLG* is NIL, no packing is done, e.g., the above becomes *A + B * C*.

CLISPIFYPACKFLG is initially T.

CLISPIFYUSERFN [Variable]

If T, causes the function *CLISPIFYUSERFN*, which should be a function of one argument, to be called on each form (list) not otherwise recognized by *CLISPIFY*. If a non-NIL value is returned, it is treated as the clispified form. Initially NIL

Note that *CLISPIFYUSERFN* must be both set and defined to use this feature.

INTERLISP-D REFERENCE MANUAL

FUNNYATOMLST

[Variable]

Suppose you have variables named A, B, and A*B. If CLISPIFY were to convert (ITIMES A B) to A*B, A*B would not translate back correctly to (ITIMES A B), since it would be the name of a variable, and therefore would not cause an error. You can prevent this from happening by adding A*B to the list FUNNYATOMLST. Then, (ITIMES A B) would CLISPIFY to A * B.

Note that A*B's appearance on FUNNYATOMLST would *not* enable DWIM and CLISP to decode A*B+C as (IPLUS A*B C); FUNNYATOMLST is used only by CLISPIFY. Thus, if an identifier contains a CLISP character, it should always be separated (with spaces) from other operators. For example, if X* is a variable, you should write (SETQ X* FORM) in CLISP as X* ←FORM, not X*←FORM. In general, it is best to avoid use of identifiers containing CLISP character operators as much as possible.

Miscellaneous Functions and Variables

CLISPFLG

[Variable]

If CLISPFLG = NIL, disables all CLISP infix or prefix transformations (but does not affect IF/THEN/ELSE statements, or iterative statements).

If CLISPFLG = TYPE-IN, CLISP transformations are performed only on expressions that are typed in for evaluation, i.e., not on user programs.

If CLISPFLG = T, CLISP transformations are performed on all expressions.

The initial value for CLISPFLG is T. CLISPIFYing anything will cause CLISPFLG to be set to T.

CLISPCHARS

[Variable]

A list of the operators that can appear in the interior of an atom. Currently (+ - * / ↑ ~ ' = ← : < > +- ~= @ !).

CLISPCHARRAY

[Variable]

A bit table of the characters on CLISPCHARS used for calls to STRPOS (Chapter 4). CLISPCHARRAY is initialized by performing (SETQ CLISPCHARRAY (MAKEBITTABLE CLISPCHARS)).

CLISPFIXSPLST

[Variable]

A list of infix operators used for spelling correction.

CLISPARRAY

[Variable]

Hash array used for storing CLISP translations. CLISPARRAY is checked by FAULTEVAL and FAULTAPPLY on erroneous forms before calling DWIM, and by the compiler.

CLISP

(**CLEARCLISPARRAY** *NAME* --) [Function]

Macro and CLISP expansions are cached in CLISPARRAY, the systems CLISP hash array. When anything changes that would invalidate an expansion, it needs to be removed from the cache. CLEARCLISPARRAY does this for you. The system does this automatically whenever you define/redefine a CLISP or macro form. If you have changed something that a CLISP word or a macro depends on the system will not be able to detect this, so you will have to invalidate the cache by calling CLEARCLISPARRAY. You can clear the whole cache by calling (CLRHASH CLISPARRAY).

(**CLISPTRAN** *X TRAN*) [Function]

Gives *X* the translation *TRAN* by storing (key *X*, value *TRAN*) in the hash array CLISPARRAY. CLISPTRAN is called for all CLISP translations, via a non-linked, external function call, so it can be advised.

(**CLISPDEC** *DECLST*) [Function]

Puts into effect the declarations in *DECLST*. CLISPDEC performs spelling corrections on words not recognized as declarations. CLISPDEC is undoable.

(**CLDISABLE** *OP*) [Function]

Disables the CLISP operator *OP*. For example, (CLDISABLE '-) makes - be just another character. CLDISABLE can be used on all CLISP operators, e.g., infix operators, prefix operators, iterative statement operators, etc. CLDISABLE is undoable.

Note: Simply removing a character operator from CLISPCHARS will prevent it from being treated as a CLISP operator when it appears as part of an atom, but it will continue to be an operator when it appears as a separate atom, e.g. (FOO + X) vs FOO+X.

CLISPIFTRANFLG [Variable]

Affects handling of translations of IF-THEN-ELSE statements (see Chapter 9). If T, the translations are stored elsewhere, and the (modified) CLISP retained. If NIL, the corresponding COND expression replaces the CLISP. Initially T.

CLISPIFYPRETTYFLG [Variable]

If non-NIL, causes PRETTYPRINT (and therefore PP and MAKEFILE) to CLISPIFY selected function definitions before printing them according to the following interpretations of CLISPIFYPRETTYFLG:

- ALL** Clispify all functions.
- T** or **EXPRS** Clispify all functions currently defined as EXPRS.
- CHANGES** Clispify all functions marked as having been changed.
- a list Clispify all functions in that list.

INTERLISP-D REFERENCE MANUAL

CLISPIFYPRETTYFLG is (temporarily) reset to T when MAKEFILE is called with the option CLISPIFY, and reset to CHANGES when the file being dumped has the property FILETYPE value CLISP. CLISPIFYPRETTYFLG is initially NIL.

Note: If CLISPIFYPRETTYFLG is non-NIL, and the only transformation performed by DWIM are well formed CLISP transformations, i.e., no spelling corrections, the function will *not* be marked as changed, since it would only have to be re-clispified and re-prettyprinted when the file was written out.

(PPT X) [NLambda NoSpread Function]

Both a function and an edit macro for prettyprinting translations. It performs a PP after first resetting PRETTYTRANFLG to T, thereby causing any translations to be printed instead of the corresponding CLISP.

CLISP: [Editor Command]

Edit macro that obtains the translation of the correct expression, if any, from CLISPARRAY, and calls EDITE on it.

CL [Editor Command]

Edit macro. Replaces current expression with CLISPIFYed current expression. Current expression can be an element or tail.

DW [Editor Command]

Edit macro. DWIMIFYs current expression, which can be an element (atom or list) or tail.

Both CL and DW can be called when the current expression is either an element or a tail and will work properly. Both consult the declarations in the function being edited, if any, and both are undoable.

(LOWERCASE FLG) [Function]

If FLG = T, LOWERCASE makes the necessary internal modifications so that CLISPIFY will use lower case versions of AND, OR, IF, THEN, ELSE, ELSEIF, and all i.s. operators. This produces more readable output. Note that you can always type in *either* upper or lower case (or a combination), regardless of the action of LOWERCASE. If FLG = NIL, CLISPIFY will use uppercase versions of AND, OR, et al. The value of LOWERCASE is its previous "setting". LOWERCASE is undoable. The initial setting for LOWERCASE is T.

CLISP Internal Conventions

CLISP is almost entirely table driven by the property lists of the corresponding infix or prefix operators. For example, much of the information used for translating the + infix operator is stored on the property list of the symbol "+". Thus it is relatively easy to add new infix or prefix operators or change old ones, simply by adding or changing selected property values. (There is some built in

CLISP

information for handling minus, `:`, `'`, and `~`, i.e., you could not yourself add such "special" operators, although you can disable or redefine them.)

Global declarations operate by changing the `LISPFN` and `CLISPINFIX` properties of the appropriate operators.

CLISPTYPE

[Property Name]

The property value of the property `CLISPTYPE` is the precedence number of the operator: higher values have higher precedence, i.e., are tighter. Note that the actual value is unimportant, only the value relative to other operators. For example, `CLISPTYPE` for `:`, `↑`, and `*` are 14, 6, and 4 respectively. Operators with the same precedence group left to right, e.g., `/` also has precedence 4, so `A/B*C` is `(A/B)*C`.

An operator can have a different left and right precedence by making the value of `CLISPTYPE` be a dotted pair of two numbers, e.g., `CLISPTYPE` of `←` is `(8 . -12)`. In this case, `CAR` is the left precedence, and `CDR` the right, i.e., `CAR` is used when comparing with operators on the *left*, and `CDR` with operators on the *right*. For example, `A*B←C+D` is parsed as `A*(B←(C+D))` because the left precedence of `←` is 8, which is higher than that of `*`, which is 4. The right precedence of `←` is -12, which is lower than that of `+`, which is 2.

If the `CLISPTYPE` property for any operator is removed, the corresponding `CLISP` transformation is disabled, as well as the inverse `CLISPIFY` transformation.

UNARYOP

[Property Name]

The value of property `UNARYOP` must be `T` for unary operators or brackets. The operand is always on the right, i.e., unary operators or brackets are always prefix operators.

BROADSCOPE

[Property Name]

The value of property `BROADSCOPE` is `T` if the operator has lower precedence than Interlisp forms, e.g., `LT`, `EQUAL`, `AND`, etc. For example, `(FOO X AND Y)` parses as `((FOO X) AND Y)`. If the `BROADSCOPE` property were removed from the property list of `AND`, `(FOO X AND Y)` would parse as `(FOO (X AND Y))`.

LISPFN

[Property Name]

The value of the property `LISPFN` is the name of the function to which the infix operator translates. For example, the value of `LISPFN` for `↑` is `EXPT`, for `'` `QUOTE`, etc. If the value of the property `LISPFN` is `NIL`, the infix operator itself is also the function, e.g., `AND`, `OR`, `EQUAL`.

SETFN

[Property Name]

If `FOO` has a `SETFN` property `FIE`, then `(FOO --)←X` translates to `(FIE -- X)`. For example, if you make `ELT` be an infix operator, e.g. `#`, by putting appropriate `CLISPTYPE` and `LISPFN` properties on the property list of `#` then you can also make `#` followed by `←` translate to `SETA`, e.g., `X#N←Y` to `(SETA X N Y)`, by putting `SETA` on the property list of

INTERLISP-D REFERENCE MANUAL

ELT under the property SETFN. Putting the list (ELT) on the property list of SETA under property SETFN will enable SETA forms to CLISPIFY back to ELT's.

CLISPINFIX [Property Name]

The value of this property is the CLISP infix to be used in CLISPIFYing. This property is stored on the property list of the corresponding Interlisp function, e.g., the value of property CLISPINFIX for EXPT is $\uparrow$, for QUOTE is ' etc.

CLISPWORD [Property Name]

Appears on the property list of clisp operators which can appear as CAR of a form, such as FETCH, REPLACE, IF, iterative statement operators, etc. Value of property is of the form (KEYWORD . NAME), where NAME is the lowercase version of the operator, and KEYWORD is its type, e.g. FORWARD, IFWORD, RECORDWORD, etc.

KEYWORD can also be the name of a function. When the atom appears as CAR of a form, the function is applied to the form and the result taken as the correct form. In this case, the function should either physically change the form, or call CLISPTRAN to store the translation.

As an example, to make & be an infix character operator meaning OR, you could do the following:

```
← (PUTPROP '&' 'CLISPTYPE (GETPROP 'OR 'CLISPTYPE))
← (PUTPROP '&' 'LISPFN 'OR)
← (PUTPROP '&' 'BROADSCOPE T)
← (PUTPROP 'OR 'CLISPINFIX '&')
← (SETQ CLISPCHARS (CONS '&' CLISPCHARS))
← (SETQ CLISPCHARRAY (MAKEBITTABLE CLISPCHARS))
```

21. PERFORMANCE ISSUES

This chapter describes a number of areas that often contribute to performance problems in Medley programs. Many performance problems can be improved by optimizing the use of storage, since allocating and reclaiming large amounts of storage is expensive. Another tactic that can sometimes yield performance improvements is to change the use of variable bindings on the stack to reduce variable lookup time. There are a number of tools that can be used to determine which parts of a computation cause performance bottlenecks.

Storage Allocation and Garbage Collection

As an Medley application program runs, it creates data structures (allocated out of free storage space), manipulates them, and then discards them. If there were no way to reclaim this space, over time the Medley memory would fill up, and the computation would come to a halt. Actually, long before this could happen the system would probably become intolerably slow, due to “data fragmentation,” which occurs when the data currently in use are spread over many virtual memory pages, so that most of the computer time must be spent swapping disk pages into physical memory. The problem of fragmentation will occur in any situation where the virtual memory is significantly larger than the real physical memory. To reduce swapping, you want to keep the “working set” (the set of pages containing actively referenced data) as small as possible.

You can write programs that don’t generate much “garbage” data, or which recycle data, but such programs tend to be complex and hard to debug. Spending effort writing such programs defeats the whole point of using a system with automatic storage allocation. An important part of any Lisp implementation is the “garbage collector” that finds discarded data and reclaims its space.

There are several well-known approaches to garbage collection. One method is the traditional mark-and-sweep, which identifies “garbage” data by marking all accessible data structures, and then sweeping through the data spaces to find all unmarked objects (i.e., not referenced by any other object). This method is guaranteed to reclaim all garbage, but it takes time proportional to the number of allocated objects, which may be very large. Also, the time that a mark-and-sweep garbage collection takes is independent of the amount of garbage collected; it is possible to sweep through the whole virtual memory, and only recover a small amount of garbage.

For interactive applications, it is not acceptable to have long interruptions in a computation for garbage collect. Medley solves this problem by using a reference-counting garbage collector. With this scheme, there is a table containing counts of how many times each object is referenced. This table is updated as pointers are created and discarded, incurring a small overhead distributed over the computation as a whole. (Note: References from the stack are not counted, but are handled separately at “sweep” time; thus the vast majority of data manipulations do not cause updates to this table.) At opportune moments, the garbage collector scans this table, and reclaims all objects that are no longer accessible (have a reference count of zero). The pause while objects are reclaimed is only the time for scanning the reference count tables (small) plus time proportional to the amount of garbage that has to

INTERLISP-D REFERENCE MANUAL

be collected (typically less than a second). "Opportune" times occur when a certain number of cells have been allocated or when the system has been waiting for you to type something for long enough. The frequency of garbage collection is controlled by the functions and variables described below. For the best system performance, it is desirable to adjust these parameters for frequent, short garbage collections, which will not interrupt interactive applications for very long, and which will have the added benefit of reducing data fragmentation, keeping the working set small.

One problem with the Medley garbage collector is that not all garbage is guaranteed to be collected. Circular data structures, which point to themselves directly or indirectly, are never reclaimed, since their reference counts are always at least one. With time, this unreclaimable garbage may increase the working set to unacceptable levels. Some users have worked with the same Medley virtual memory for a very long time, but it is a good idea to occasionally save all of your functions in files, reinitialize Medley, and rebuild your system. Many users end their working day by issuing a command to rebuild their system and then leaving the machine to perform this task in their absence. If the system seems to be spending too much time swapping (an indication of fragmented working set), this procedure is definitely recommended.

Another limitation of the reference-counting garbage collector is that the table in which reference counts are maintained is of fixed size. For typical Lisp objects that are pointed to from exactly one place (e.g., the individual conses in a list), no burden is placed on this table, since objects whose reference count is 1 are not explicitly represented in the table. However, large, "rich" data structures, with many interconnections, backward links, cross references, etc, can contribute many entries to the reference count table. For example, if you created a data structure that functioned as a doubly-linked list, such a structure would contribute an entry (reference count 2) for each element.

When the reference count table fills up, the garbage collector can no longer maintain consistent reference counts, so it stops doing so altogether. At this point, a window appears on the screen with the following message, and the debugger is entered:

```
Internal garbage collector tables have overflowed, due
to too many pointers with reference count greater than 1.
*** The garbage collector is now disabled. ***
Save your work and reload as soon as possible.
```

[This message is slightly misleading, in that it should say "count not equal to 1". In the current implementation, the garbage collection of a large pointer array whose elements are not otherwise pointed to can place a special burden on the table, as each element's reference count simultaneously drops to zero and is thus added to the reference count table for the short period before the element is itself reclaimed.]

If you exit the debugger window (e.g., with the RETURN command), your computation can proceed; however, the garbage collector is no longer operating. Thus, your virtual memory will become cluttered with objects no longer accessible, and if you continue for long enough in the same virtual memory image you will eventually fill up the virtual memory backing store and grind to a halt.

PERFORMANCE ISSUES

Garbage collection in Medley is controlled by the following functions and variables:

(RECLAIM) [Function]

Initiates a garbage collection. Returns 0.

(RECLAIMMIN *N*) [Function]

Sets the frequency of garbage collection. Interlisp keeps track of the number of cells of any type that have been allocated; when it reaches a given number, a garbage collection occurs. If *N* is non-NIL, this number is set to *N*. Returns the current setting of the number.

RECLAIMWAIT [Variable]

Medley will invoke a RECLAIM if the system is idle and waiting for your input for RECLAIMWAIT seconds (currently set for 4 seconds).

(GCGAG *MESSAGE*) [Function]

Sets the behavior that occurs while a garbage collection is taking place. If *MESSAGE* is non-NIL, the cursor is complemented during a RECLAIM; if *MESSAGE* = NIL, nothing happens. The value of GCGAG is its previous setting.

(GCTRP) [Function]

Returns the number of cells until the next garbage collection, according to the RECLAIMMIN number.

The amount of storage allocated to different data types, how much of that storage is in use, and the amount of data fragmentation can be determined using the following function:

(STORAGE TYPES *PAGETHRESHOLD*) [Function]

STORAGE prints out a summary, for each data type, of the amount of space allocated to the data type, and how much of that space is currently in use. If *TYPES* is non-NIL, STORAGE only lists statistics for the specified types. *TYPES* can be a symbol or a list of types. If *PAGETHRESHOLD* is non-NIL, then STORAGE only lists statistics for types that have at least *PAGETHRESHOLD* pages allocated to them.

STORAGE prints out a table with the column headings Type, Assigned, Free Items, In use, and Total alloc. Type is the name of the data type. Assigned is how much of your virtual memory is set aside for items of this type. Currently, memory is allocated in quanta of two pages (1024 bytes). The numbers under Assigned show the number of pages and the total number of items that fit on those pages. Free Items shows how many items are available to be allocated (using the create construct, Chapter 8); these constitute the "free list" for that data type. In use shows how many items of this type are currently in use, i.e., have pointers to them and hence have not been garbage collected. If this number is higher than your program seems to warrant, you may want to look for storage leaks. The sum of Free Items and In use is always the same

INTERLISP-D REFERENCE MANUAL

as the total Assigned items. Total alloc is the total number of items of this type that have ever been allocated (see BOXCOUNT, in the Performance Measuring section below).

Note: The information about the number of items of type LISTP is only approximate, because list cells are allocated in a special way that precludes easy computation of the number of items per page.

Note: When a data type is redeclared, the data type name is reassigned. Pages which were assigned to instances of the old data type are labeled **DEALLOC**.

At the end of the table printout, STORAGE prints a "Data Spaces Summary" listing the number of pages allocated to the major data areas in the virtual address space: the space for fixed-length items (including datatypes), the space for variable-length items, and the space for symbols. Variable-length data types such as arrays have fixed-length "headers," which is why they also appear in the printout of fixed-length data types. Thus, the line printed for the BITMAP data type says how many bitmaps have been allocated, but the "assigned pages" column counts only the headers, not the space used by the variable-length part of the bitmap. This summary also lists "Remaining Pages" in relation to the largest possible virtual memory, not the size of the virtual memory backing file in use. This file may fill up, causing a STORAGE FULL error, long before the "Remaining Pages" numbers reach zero.

STORAGE also prints out information about the sizes of the entries on the variable-length data free list. The block sizes are broken down by the value of the variable STORAGE.ARRAYSIZES, initially (4 16 64 256 1024 4096 16384 NIL), which yields a printout of the form:

```
variable-datum free list:
le 4          26 items;    104 cells.
le 16         72 items;    783 cells.
le 64         36 items;    964 cells.
le 256        28 items;   3155 cells.
le 1024        3 items;   1175 cells.
le 4096        5 items;   8303 cells.
le 16384       3 items;  17067 cells.
others        1 items;  17559 cells.
```

This information can be useful in determining if the variable-length data space is fragmented. If most of the free space is composed of small items, then the allocator may not be able to find room for large items, and will extend the variable datum space. If this is extended too much, this could cause an ARRAYS FULL error, even if there is a lot of space left in little chunks.

(STORAGE.LEFT)

[Function]

Provides a programmatic way of determining how much storage is left in the major data areas in the virtual address space. Returns a list of the form (MDSFREE MDSFRAC SMBFRAC ATOMFREE ATOMFRAC), where the elements are interpreted as follows:

MDSFREE The number of free pages left in the main data space (which includes both fixed-length and variable-length data types).

PERFORMANCE ISSUES

MDSFRAC The fraction of the total possible main data space that is free.

8MBFRAC The fraction of the total main data space that is free, relative to eight megabytes.

This number is useful when using Medley on some early computers where the hardware limits the address space to eight megabytes. The function `32MBADDRESSABLE` returns non-NIL if the currently running Medley system can use the full 32 megabyte address space.

ATOMFREE The number of free pages left in the symbol space.

ATOMFRAC The fraction of the total symbol space that is free.

Note: Another important space resource is the amount of the virtual memory backing file in use (see `VMEMSIZE`, Chapter 12). The system will crash if the virtual memory file is full, even if the address space is not exhausted.

Variable Bindings

Different implementations of Lisp use different methods of accessing free variables. The binding of variables occurs when a function or a `PROG` is entered. For example, if the function `FOO` has the definition `(LAMBDA (A B) BODY)`, the variables `A` and `B` are bound so that any reference to `A` or `B` from `BODY` or any function called from `BODY` will refer to the arguments to the function `FOO` and not to the value of `A` or `B` from a higher level function. All variable names (symbols) have a top level value cell which is used if the variable has not been bound in any function. In discussions of variable access, it is useful to distinguish between three types of variable access: local, special and global. Local variable access is the use of a variable that is bound within the function from which it is used. Special variable access is the use of a variable that is bound by another function. Global variable access is the use of a variable that has not been bound in any function. We will often refer to a variable all of whose accesses are local as a "local variable." Similarly, a variable all of whose accesses are global we call a "global variable."

In a "deep" bound system, a variable is bound by saving on the stack the variable's name together with a value cell which contains that variable's new value. When a variable is accessed, its value is found by searching the stack for the most recent binding (occurrence) and retrieving the value stored there. If the variable is not found on the stack, the variable's top level value cell is used.

In a "shallow" bound system, a variable is bound by saving on the stack the variable name and the variable's old value and putting the new value in the variable's top level value cell. When a variable is accessed, its value is always found in its top level value cell.

The deep binding scheme has one disadvantage: the amount of cpu time required to fetch the value of a variable depends on the stack distance between its use and its binding. The compiler can determine

INTERLISP-D REFERENCE MANUAL

local variable accesses and compiles them as fetches directly from the stack. Thus this computation cost only arises in the use of variable not bound in the local frame ("free" variables). The process of finding the value of a free variable is called free variable lookup.

In a shallow bound system, the amount of cpu time required to fetch the value of a variable is constant regardless of whether the variable is local, special or global. The disadvantages of this scheme are that the actual binding of a variable takes longer (thus slowing down function call), the cells that contain the current in use values are spread throughout the space of all symbol value cells (thus increasing the working set size of functions) and context switching between processes requires unwinding and rewinding the stack (thus effectively prohibiting the use of context switching for many applications).

Medley uses deep binding, because of the working set considerations and the speed of context switching. The free variable lookup routine is microcoded, thus greatly reducing the search time. In benchmarks, the largest percentage of free variable lookup time was 20 percent of the total elapsed time; the normal time was between 5 and 10 percent.

Because of the deep binding, you can sometimes significantly improve performance by declaring global variables. If a variable is declared global, the compiler will compile an access to that variable as a retrieval of its top level value, completely bypassing a stack search. This should be done only for variables that are never bound in functions, such as global databases and flags.

Global variable declarations should be done using the `GLOBALVARS` file manager command (Chapter 17). Its form is `(GLOBALVARS VAR ... VAR )`.

Another way of improving performance is to declare variables as local within a function. Normally, all variables bound within a function have their names put on the stack, and these names are scanned during free variable lookup. If a variable is declared to be local within a function, its name is not put on the stack, so it is not scanned during free variable lookup, which may increase the speed of lookups. The compiler can also make some other optimizations if a variable is known to be local to a function.

A variable may be declared as local within a function by including the form `(DECLARE (LOCALVARS VAR ... VAR ))` following the argument list in the definition of the function. Local variable declarations only effect the compilation of a function. Interpreted functions put all of their variable names on the stack, regardless of any declarations.

Performance Measuring

This section describes functions that gather and display statistics about a computation, such as the elapsed time, and the number of data objects of different types allocated. `TIMEALL` and `TIME` gather statistics on the evaluation of a specified form. `BREAKDOWN` gathers statistics on individual functions called during a computation. These functions can be used to determine which parts of a computation are consuming the most resources (time, storage, etc.), and could most profitably be improved.

PERFORMANCE ISSUES

(TIMEALL *TIMEFORM* *NUMBEROFTIMES* *TIMEWHAT* *INTERPFLG*) [NLambda Function]

Evaluates the form *TIMEFORM* and prints statistics on time spent in various categories (elapsed, keyboard wait, swapping time, gc) and data type allocation.

For more accurate measurement on small computations, *NUMBEROFTIMES* may be specified (its default is 1) to cause *TIMEFORM* to be executed *NUMBEROFTIMES* times. To improve the accuracy of timing open-coded operations in this case, *TIMEALL* compiles a form to execute *TIMEFORM* *NUMBEROFTIMES* times (unless *INTERPFLG* is non-NIL), and then times the execution of the compiled form.

Note: If *TIMEALL* is called with *NUMBEROFTIMES* > 1, the dummy form is compiled with compiler optimizations on. This means that it is not meaningful to use *TIMEALL* with very simple forms that are optimized out by the compiler. For example, (TIMEALL ' (IPLUS 2 3) 1000) will time a compiled function which simply returns the number 5, since (IPLUS 2 3) is optimized to the integer 5.

TIMEWHAT restricts the statistics to specific categories. It can be an atom or list of datatypes to monitor, and/or the atom *TIME* to monitor time spent. Note that ordinarily, *TIMEALL* monitors all time and datatype usage, so this argument is rarely needed.

TIMEALL returns the value of the last evaluation of *TIMEFORM*.

(TIME *TIMEEX* *TIMEN* *TIMETYP*) [NLambda Function]

TIME evaluates the form *TIMEEX*, and prints out the number of CONS cells allocated and computation time. Garbage collection time is subtracted out. This function has been largely replaced by *TIMEALL*.

If *TIMEN* is greater than 1, *TIMEEX* is executed *TIMEN* times, and *TIME* prints out (number of conses)/ *TIMEN*, and (computation time)/ *TIMEN*. If *TIMEN* = NIL, it defaults to 1. This is useful for more accurate measurement on small computations.

If *TIMETYP* is 0, *TIME* measures and prints total *real* time as well as computation time. If *TIMETYP* = 3, *TIME* measures and prints garbage collection time as well as computation time. If *TIMETYP* = T, *TIME* measures and prints the number of pagefaults.

TIME returns the value of the last evaluation of *TIMEEX*.

(BOXCOUNT *TYPE* *N*) [Function]

Returns the number of data objects of type *TYPE* allocated since this Interlisp system was created. *TYPE* can be any data type name (see *TYPENAME*, Chapter 8). If *TYPE* is NIL, it defaults to *FIXP*. If *N* is non-NIL, the corresponding counter is reset to *N*.

(CONSCOUNT *N*) [Function]

Returns the number of CONS cells allocated since this Interlisp system was created. If *N* is non-NIL, resets the counter to *N*. Equivalent to (BOXCOUNT ' LISTP *N*).

INTERLISP-D REFERENCE MANUAL

(**PAGEFAULTS**)

[Function]

Returns the number of page faults since this Interlisp system was created.

BREAKDOWN

TIMEALL collects statistics for whole computations. BREAKDOWN is available to analyze the breakdown of computation time (or any other measureable quantity) function by function.

(**BREAKDOWN** *FN* ... *FN*)

[NLambda NoSpread Function]

You call BREAKDOWN giving it a list of function names (unevaluated). These functions are modified so that they keep track of various statistics.

To remove functions from those being monitored, simply UNBREAK (Chapter 15) the functions, thereby restoring them to their original state. To add functions, call BREAKDOWN on the new functions. This will not reset the counters for any functions not on the new list. However (BREAKDOWN) will zero the counters of all functions being monitored.

The procedure used for measuring is such that if one function calls other and both are "broken down", then the time (or whatever quantity is being measured) spent in the inner function is *not* charged to the outer function as well.

BREAKDOWN will *not* give accurate results if a function being measured is not returned from normally, e.g., a lower RETFROM (or ERROR) bypasses it. In this case, all of the time (or whatever quantity is being measured) between the time that function is entered and the time the next function being measured is entered will be charged to the first function.

(**BRKDOWNRESULTS** *RETURNVALUESFLG*)

[Function]

BRKDOWNRESULTS prints the analysis of the statistics requested as well as the number of calls to each function. If *RETURNVALUESFLG* is non-NIL, BRKDOWNRESULTS will not to print the results, but instead return them in the form of a list of elements of the form (*FNNAME* *#CALLS* *VALUE*).

Example:

```
← (BREAKDOWN SUPERPRINT SUBPRINT COMMENT1)
(SUPERPRINT SUBPRINT COMMENT1)
← (PRETTYDEF ' (SUPERPRINT) ' FOO)
FOO.;3
← (BRKDOWNRESULTS)
FUNCTIONS    TIME      #CALLS  PER CALL  %
SUPERPRINT   8.261      365      0.023      20
SUBPRINT     31.910      141      0.226      76
COMMENT1      1.612       8      0.201       4
TOTAL        41.783      514      0.081
NIL
← (BRKDOWNRESULTS T)
((SUPERPRINT 365 8261) (SUBPRINT 141 31910)
 (COMMENT1 8 1612))
```

PERFORMANCE ISSUES

BREAKDOWN can be used to measure other statistics, by setting the following variables:

BRKDWNTYPE

[Variable]

To use BREAKDOWN to measure other statistics, before calling BREAKDOWN, set the variable BRKDWNTYPE to the quantity of interest, e.g., TIME, CONSES, etc, or a list of such quantities. Whenever BREAKDOWN is called with BRKDWNTYPE not NIL, BREAKDOWN performs the necessary changes to its internal state to conform to the new analysis. In particular, if this is the first time an analysis is being run with a particular statistic, a measuring function will be defined, and the compiler will be called to compile it. The functions being broken down will be redefined to call this measuring function. When BREAKDOWN is through initializing, it sets BRKDWNTYPE back to NIL. Subsequent calls to BREAKDOWN will measure the new statistic until BRKDWNTYPE is again set and a new BREAKDOWN performed.

BRKDWNTYPES

[Variable]

The list BRKDWNTYPES contains the information used to analyze new statistics. Each entry on BRKDWNTYPES should be of the form (TYPE FORM FUNCTION), where TYPE is a statistic name (as would appear in BRKDWNTYPE), FORM computes the statistic, and FUNCTION (optional) converts the value of form to some more interesting quantity. For example, (TIME (CLOCK 2) (LAMBDA (X) (FQUOTIENT X 1000))) measures computation time and reports the result in seconds instead of milliseconds. BRKDWNTYPES currently contains entries for TIME, CONSES, PAGEFAULTS, BOXES, and FBOXES.

Example:

```
←(SETQ BRKDWNTYPE '(TIME CONSES))
(TIME CONSES)
←(BREAKDOWN MATCH CONSTRUCT)
(MATCH CONSTRUCT)
←(FLIP '(A B C D E F G H C Z) '(.. $1 .. #2 ..)
'(.. #3 ..))
(A B D E F G H Z)
←(BRKDWNRRESULTS)
FUNCTIONS  TIME      #CALLS  PER CALL  %
MATCH      0.036      1        0.036     54
CONSTRUCT  0.031      1        0.031     46
TOTAL      0.067      2        0.033
FUNCTIONS  CONSES    #CALLS  PER CALL  %
MATCH      32        1       32.000    40
CONSTRUCT  49        1       49.000    60
TOTAL      81        2       40.500
NIL
```

Occasionally, a function being analyzed is sufficiently fast that the overhead involved in measuring it obscures the actual time spent in the function. If you were using TIME, you would specify a value for TIMEN greater than 1 to give greater accuracy. A similar option is available for BREAKDOWN. You can specify that a function(s) be executed a multiple number of times for each measurement, and the average value reported, by including a number in the list of functions given to BREAKDOWN. For example, BREAKDOWN(EDITCOM EDIT4F 10 EDIT4E EQP) means normal breakdown for EDITCOM and EDIT4F but

INTERLISP-D REFERENCE MANUAL

executes (the body of) EDIT4E and EQP 10 times each time they are called. Of course, the functions so measured must not cause any harmful side effects, since they are executed more than once for each call. The printout from BRKDOWNRESULTS will look the same as though each function were run only once, except that the measurement will be more accurate.

Another way of obtaining more accurate measurement is to expand the call to the measuring function in-line. If the value of BRKDOWNCOMPFLG is non-NIL (initially NIL), then whenever a function is broken-down, it will be redefined to call the measuring function, and then recompiled. The measuring function is expanded in-line via an appropriate macro. In addition, whenever BRKDOWNTYPE is reset, the compiler is called for *all* functions for which BRKDOWNCOMPFLG was set at the time they were originally broken-down, i.e. the setting of the flag at the time a function is broken-down determines whether the call to the measuring code is compiled in-line.

GAINSPACE

If you have large programs and databases, you may sometimes find yourself in a situation where you need to obtain more space, and are willing to pay the price of eliminating some or all of the context information that the various user-assistance facilities such as the programmer's assistant, file package, CLISP, etc., have accumulated during the course of his session. The function GAINSPACE provides an easy way to selectively throw away accumulated data:

(GAINSPACE)

[Function]

Prints a list of deletable objects, allowing you to specify at each point what should be discarded and what should be retained. For example:

```
← (GAINSPACE)
purge history lists ? Yes
purge everything, or just the properties, e.g.,
SIDE, LISPXPRINT, etc. ?
just the properties
discard definitions on property lists ? Yes
discard old values of variables ? Yes
erase properties ? No
erase CLISP translations? Yes
```

GAINSPACE is driven by the list GAINSPACEFORMS. Each element on GAINSPACEFORMS is of the form (PRECHECK MESSAGE FORM KEYLST). If PRECHECK, when evaluated, returns NIL, GAINSPACE skips to the next entry. For example, you will not be asked whether or not to purge the history list if it is not enabled. Otherwise, ASKUSER (Chapter 26) is called with the indicated MESSAGE and the (optional) KEYLST. If you respond No, i.e., ASKUSER returns N, GAINSPACE skips to the next entry. Otherwise, FORM is evaluated with the variable RESPONSE bound to the value of ASKUSER. In the above example, the FORM for the "purge history lists" question calls ASKUSER to ask "purge everything, ..." only if you had responded Yes. If you had responded with Everything, the second question would not have been asked.

PERFORMANCE ISSUES

The "erase properties" question is driven by a list SMASHPROPSMENU. Each element on this list is of the form (*MESSAGE* . *PROPS*). You are prompted with *MESSAGE* (by ASKUSER), and if your response is Yes, *PROPS* is added to the list SMASHPROPS. The "discard definitions on property lists" and "discard old values of variables" questions also add to SMASHPROPS. You will not be prompted for any entry on SMASHPROPSMENU for which all of the corresponding properties are already on SMASHPROPS. SMASHPROPS is initially set to the value of SMASHPROPSLST. This permits you to specify in advance those properties which you always want discarded, and not be asked about them subsequently. After finishing all the entries on GAINSPACEFORMS, GAINSPACE checks to see if the value of SMASHPROPS is non-NIL, and if so, does a MAPATOMS, i.e., looks at every atom in the system, and erases the indicated properties.

You can change or add new entries to GAINSPACEFORMS or SMASHPROPSMENU, so that GAINSPACE can also be used to purge structures that your programs have accumulated.

Using Data Types Instead of Records

If a program uses large numbers of large data structures, there are several advantages to representing them as user data types rather than as list structures. The primary advantage is increased speed: accessing and setting the fields of a data type can be significantly faster than walking through a list with repeated CARS and CDRs. Also,

Compiled code for referencing data types is usually smaller. Finally, by reducing the number of objects created (one object against many list cells), this can reduce the expense of garbage collection.

User data types are declared by using the DATATYPE record type (Chapter 8). If a list structure has been defined using the RECORD record type (Chapter 8), and all accessing operations are written using the record package's fetch, replace, and create operations, changing from RECORDS to DATATYPES only requires editing the record declaration (using EDITREC, Chapter 8) to replace declaration type RECORD by DATATYPE, and recompiling.

Note: There are some minor disadvantages: First, there is an upper limit on the number of data types that can exist. Also, space for data types is allocated two pages at a time. Each data type which has any instances allocated has at least two pages assigned to it, which may be wasteful of space if there are only a few examples of a given data type. These problems should not effect most applications programs.

Using "Fast" and "Destructive" Functions

Among the functions used for manipulating objects of various data types, there are a number of functions which have "fast" and "destructive" versions. You should be aware of what these functions do, and when they should be used.

INTERLISP-D REFERENCE MANUAL

“Fast” functions: By convention, a function named by prefixing an existing function name with F indicates that the new function is a "fast" version of the old. These usually have the same definitions as the slower versions, but they compile open and run without any "safety" error checks. For example, FNTH runs faster than NTH, however, it does not make as many checks (for lists ending with anything but NIL, etc). If these functions are given arguments that are not in the form that they expect, their behavior is unpredictable; they may run forever, or cause a system error. In general, you should only use "fast" functions in code that has already been completely debugged, to speed it up.

“Destructive” functions: By convention, a function named by prefixing an existing function with D indicates the new function is a "destructive" version of the old one, which does not make any new structure but cannibalizes its argument(s). For example, REMOVE returns a copy of a list with a particular element removed, but DREMOVE actually changes the list structure of the list. (Unfortunately, not all destructive functions follow this naming convention: the destructive version of APPEND is NCONC.) You should be careful when using destructive functions that they do not inadvertently change data structures.

22. PERFORMANCE ISSUES

This chapter describes a number of areas that often contribute to performance problems in Interlisp-D programs. Many performance problems can be improved by optimizing the use of storage, since allocating and reclaiming large amounts of storage is expensive. Another tactic that can sometimes yield performance improvements is to change the use of variable bindings on the stack to reduce variable lookup time. There are a number of tools that can be used to determine which parts of a computation cause performance bottlenecks.

Storage Allocation and Garbage Collection

As an Interlisp-D applications program runs, it creates data structures (allocated out of free storage space), manipulates them, and then discards them. If there were no way of reclaiming this space, over time the Interlisp-D memory (both the physical memory in the machine and the virtual memory stored on the disk) would fill up, and the computation would come to a halt. Actually, long before this could happen the system would probably become intolerably slow, due to "data fragmentation," which occurs when the data currently in use are spread over many virtual memory pages, so that most of the computer time must be spent swapping disk pages into physical memory. The problem of fragmentation will occur in any situation where the virtual memory is significantly larger than the real physical memory. To reduce swapping, it is desirable to keep the "working set" (the set of pages containing actively referenced data) as small as possible.

It is possible to write programs that don't generate much "garbage" data, or which recycle data, but such programs tend to be overly complicated and difficult to debug. Spending effort writing such programs defeats the whole point of using a system with automatic storage allocation. An important part of any Lisp implementation is the "garbage collector" which identifies discarded data and reclaims its space. There are several well-known approaches to garbage collection. One method is the traditional mark-and-sweep garbage collection algorithm, which identifies "garbage" data by marking all accessible data structures, and then sweeping through the data spaces to find all unmarked objects (i.e., not referenced by any other object). Although this method is guaranteed to reclaim all garbage, it takes time proportional to the number of allocated objects, which may be very large. (Some allocated objects will have been marked during the "mark" phase, and the remainder will be collected during the

"sweep" phase; so all will have to be touched in some way.) Also, the time that a mark-and-sweep garbage collection takes is independent of the amount of garbage collected; it is possible to sweep through the whole virtual memory, and only recover a small amount of garbage.

For interactive applications, it is not acceptable to have long interruptions in a computation for the purpose of garbage collection. Interlisp-D solves this problem by using a reference-counting garbage collector. With this scheme, there is a table containing counts of how many times each object is referenced. This table is incrementally updated as pointers are created and discarded, incurring a small overhead distributed over the computation as a whole. (Note: References from the stack are not counted, but are handled separately at "sweep" time; thus the vast majority of data manipulations do not cause updates to this table.) At opportune moments, the garbage collector scans this table, and reclaims all objects that are no longer accessible (have a reference count of zero). The pause while objects are reclaimed is only the time for scanning the reference count tables (small) plus time proportional to the amount of garbage that has to be collected (typically less than a second). "Opportune" times occur when a certain number of cells have been allocated or when the system has been waiting for the user to type something for long enough. The frequency of garbage collection is controlled by the functions and variables described below. For the best system performance, it is desirable to adjust these parameters for frequent, short garbage collections, which will not interrupt interactive applications for very long, and which will have the added benefit of reducing data fragmentation, keeping the working set small.

One problem with the Interlisp-D garbage collector is that not all garbage is guaranteed to be collected. Circular data structures, which point to themselves directly or indirectly, are never reclaimed, since their reference counts are always at least one. With time, this unreclaimable garbage may increase the working set to unacceptable levels. Some users have worked with the same Interlisp-D virtual memory for a very long time, but it is a good idea to occasionally save all of your functions in files, reinitialize Interlisp-D, and rebuild your system. Many users end their working day by issuing a command to rebuild their system and then leaving the machine to perform this task in their absence. If the system seems to be spending too much time swapping (an indication of fragmented working set), this procedure is definitely recommended.

Garbage collection in Interlisp-D is controlled by the following functions and variables:

(RECLAIM) [Function]

Initiates a garbage collection. Returns 0.

(RECLAIMMIN *N*) [Function]

Sets the frequency of garbage collection. Interlisp keeps track of the number of cells of any type that have been allocated; when it reaches a given number, a

garbage collection occurs. If *N* is non-NIL, this number is set to *N*. Returns the current setting of the number.

RECLAIMWAIT [Variable]

Interlisp-D will invoke a RECLAIM if the system is idle and waiting for your input for RECLAIMWAIT seconds (currently set for 4 seconds).

(GCGAG MESSAGE) [Function]

Sets the behavior that occurs while a garbage collection is taking place. If *MESSAGE* is non-NIL, the cursor is complemented during a RECLAIM; if *MESSAGE*=NIL, nothing happens. The value of GCGAG is its previous setting.

(GCTRP) [Function]

Returns the number of cells until the next garbage collection, according to the RECLAIMMIN number.

The amount of storage allocated to different data types, how much of that storage is in use, and the amount of data fragmentation can be determined using the following function:

(STORAGE TYPES PAGETHRESHOLD) [Function]

STORAGE prints out a summary, for each data type, of the amount of space allocated to the data type, and how much of that space is currently in use. If *TYPES* is non-NIL, STORAGE only lists statistics for the specified types. *TYPES* can be a listatom or a list of types. If *PAGETHRESHOLD* is non-NIL, then STORAGE only lists statistics for types that have at least *PAGETHRESHOLD* pages allocated to them.

STORAGE prints out a table with the column headings **Type**, **Assigned**, **Free Items**, **In use**, and **Total alloc**. **Type** is the name of the data type. **Assigned** is how much of your virtual memory is set aside for items of this type. Currently, memory is allocated in quanta of two pages (1024 bytes). The numbers under **Assigned** show the number of pages and the total number of items that fit on those pages. **Free Items** shows how many items are available to be allocated (using the **create** construct, Chapter 8); these constitute the "free list" for that data type. **In use** shows how many items of this type are currently in use, i.e., have pointers to them and hence have not been garbage collected. If this number is higher than your program seems to warrant, you may want to look for storage leaks. The sum of **Free Items** and **In use** is always the same as the total **Assigned** items. **Total alloc** is the total number of items of this type that have ever been allocated (see BOXCOUNT, in the Performance Measuring section below).

Note: The information about the number of items of type LISTP is only approximate, because list cells are allocated in a special way that precludes easy computation of the number of items per page.

Note: When a data type is redeclared, the data type name is reassigned. Pages which were assigned to instances of the old data type are labeled ****DEALLOC****.

At the end of the table printout, STORAGE prints a "Data Spaces Summary" listing the number of pages allocated to the major data areas in the virtual address space: the space for fixed-length items (including datatypes), the space for variable-length items, and the space for litatoms. Variable-length data types such as arrays have fixed-length "headers," which is why they also appear in the printout of fixed-length data types. Thus, the line printed for the BITMAP data type says how many bitmaps have been allocated, but the "assigned pages" column counts only the headers, not the space used by the variable-length part of the bitmap. This summary also lists "Remaining Pages" in relation to the largest possible virtual memory, not the size of the virtual memory backing file in use. This file may fill up, causing a **STORAGE FULL** error, long before the "Remaining Pages" numbers reach zero.

STORAGE also prints out information about the sizes of the entries on the variable-length data free list. The block sizes are broken down by the value of the variable STORAGE.ARRAYSIZES, initially (4 16 64 256 1024 4096 16384 NIL), which yields a printout of the form:

```
variable-datum free list:
le 4          26 items;    104 cells.
le 16         72 items;    783 cells.
le 64         36 items;    964 cells.
le 256        28 items;   3155 cells.
le 1024        3 items;   1175 cells.
```

```
le 4096      5 items;   8303 cells.  
le 16384     3 items;  17067 cells.  
others      1 items;  17559 cells.
```

This information can be useful in determining if the variable-length data space is fragmented. If most of the free space is composed of small items, then the allocator may not be able to find room for large items, and will extend the variable datum space. If this is extended too much, this could cause an `ARRAYS FULL` error, even if there is a lot of space left in little chunks.

(STORAGE.LEFT)

[Function]

Provides a programmatic way of determining how much storage is left in the major data areas in the virtual address space. Returns a list of the form (*MDSFREE MDSFRAC 8MBFRAC ATOMFREE ATOMFRAC*), where the elements are interpreted as follows:

<i>MDSFREE</i>	The number of free pages left in the main data space (which includes both fixed-length and variable-length data types).
<i>MDSFRAC</i>	The fraction of the total possible main data space that is free.
<i>8MBFRAC</i>	The fraction of the total main data space that is free, relative to eight megabytes. This number is useful when using Interlisp-D on some early computers where the hardware limits the address space to eight megabytes. The function 32MBADDRESSABLE returns non-NIL if the currently running Interlisp-D system can use the full 32 megabyte address space.
<i>ATOMFREE</i>	The number of free pages left in the litatom space.
<i>ATOMFRAC</i>	The fraction of the total litatom space that is free.

Note: Another important space resource is the amount of the virtual memory backing file in use (see VMEMSIZE, Chapter 12). The system will crash if the virtual memory file is full, even if the address space is not exhausted.

Variable Bindings

Different implementations of lisp use different methods of accessing free variables. The binding of variables occurs when a function or a PROG is entered. For example, if the function FOO has the definition (LAMBDA (A B) BODY), the variables A and B are bound so that any reference to A or B from BODY or any function called from BODY will refer to the arguments to the function FOO and not to the value of A or B from a higher level function. All variable names (litatoms) have a top level value cell which is used if the variable has not been bound in any function. In discussions of variable access, it is useful to distinguish between three types of variable access: local, special and global. Local variable access is the use of a variable that is bound within the function from which it is used. Special variable access is the use of a variable that is bound by another function. Global variable access is the use of a variable that has not been bound in any function. We will often refer to a variable all of whose accesses are local as a "local variable." Similarly, a variable all of whose accesses are global we call a "global variable."

In a "deep" bound system, a variable is bound by saving on the stack the variable's name together with a value cell which contains that variable's new value. When a variable is accessed, its value is found by searching the stack for the most recent binding (occurrence) and retrieving the value stored there. If the variable is not found on the stack, the variable's top level value cell is used.

In a "shallow" bound system, a variable is bound by saving on the stack the variable name and the variable's old value and putting the new value in the variable's top level value cell. When a variable is accessed, its value is always found in its top level value cell.

The deep binding scheme has one disadvantage: the amount of cpu time required to fetch the value of a variable depends on the stack distance between its use and its binding. The compiler can determine local variable accesses and compile them as fetches directly from the stack. Thus this computation cost only arises in the use of variable not bound in the local frame ("free" variables). The process of finding the value of a free variable is called free variable lookup.

In a shallow bound system, the amount of cpu time required to fetch the value of a variable is constant regardless of whether the variable is local, special or global. The disadvantages of this scheme are that the actual binding of a variable takes longer (thus slowing down function call), the cells that contain the current in use values are spread throughout the space of all litatom value cells (thus increasing the working set size of functions) and context switching between processes requires unwinding and rewinding the stack (thus effectively prohibiting the use of context switching for many applications).

Interlisp-D uses deep binding, because of the working set considerations and the speed of context switching. The free variable lookup routine is microcoded, thus greatly reducing the search time. In benchmarks, the largest percentage of free variable lookup time was 20 percent of the total elapsed time; the normal time was between 5 and 10 percent.

One consequence of Interlisp-D's deep binding scheme is that users may significantly improve performance by declaring global variables in certain situations. If a variable is declared global, the compiler will compile an access to that variable as a retrieval of its top level value, completely bypassing a stack search. This should be done only for variables that are never bound in functions, such as global databases and flags.

Global variable declarations should be done using the GLOBALVARS file package command (Chapter 17). Its form is (GLOBALVARS VAR ... VAR).

Another way of improving performance is to declare variables as local within a function. Normally, all variables bound within a function have their names put on the stack, and these names are scanned during free variable lookup. If a variable is declared to be local within a function, its name is not put on the stack, so it is not scanned during free variable lookup, which may increase the speed of lookups. The compiler can also make some other optimizations if a variable is known to be local to a function.

A variable may be declared as local within a function by including the form (DECLARE (LOCALVARS VAR ... VAR)) following the argument list in the definition of the function. Local variable declarations only effect the compilation of a function. Interpreted functions put all of their variable names on the stack, regardless of any declarations.

Performance Measuring

This section describes functions that gather and display statistics about a computation, such as the elapsed time, and the number of data objects of different types allocated. `TIMEALL` and `TIME` gather statistics on the evaluation of a specified form. `BREAKDOWN` gathers statistics on individual functions called during a computation. These functions can be used to determine which parts of a computation are consuming the most resources (time, storage, etc.), and could most profitably be improved.

(`TIMEALL` *TIMEFORM* *NUMBEROFTIMES* *TIMEWHAT* *INTERPFLG* —) [NLambda Function]

Evaluates the form *TIMEFORM* and prints statistics on time spent in various categories (elapsed, keyboard wait, swapping time, gc) and data type allocation.

For more accurate measurement on small computations, *NUMBEROFTIMES* may be specified (its default is 1) to cause *TIMEFORM* to be executed *NUMBEROFTIMES* times. To improve the accuracy of timing open-coded operations in this case, `TIMEALL` compiles a form to execute *TIMEFORM* *NUMBEROFTIMES* times (unless *INTERPFLG* is non-NIL), and then times the execution of the compiled form.

Note: If `TIMEALL` is called with *NUMBEROFTIMES*>1, the dummy form is compiled with compiler optimizations on. This means that it is not meaningful to use `TIMEALL` with very simple forms that are optimized out by the compiler. For example, (`TIMEALL` ' (*IPLUS* 2 3) 1000) will time a compiled function which simply returns the number 5, since (*IPLUS* 2 3) is optimized to the integer 5.

TIMEWHAT restricts the statistics to specific categories. It can be an atom or list of datatypes to monitor, and/or the atom `TIME` to monitor time spent. Note that ordinarily, `TIMEALL` monitors all time and datatype usage, so this argument is rarely needed.

`TIMEALL` returns the value of the last evaluation of *TIMEFORM*.

(`TIME` *TIMEX* *TIMEN* *TIMETYP*) [NLambda Function]

`TIME` evaluates the form *TIMEX*, and prints out the number of CONS cells allocated and computation time. Garbage collection time is subtracted out. This function has been largely replaced by `TIMEALL`.

If *TIMEN* is greater than 1, *TIMEX* is executed *TIMEN* times, and `TIME` prints out (number of conses)/*TIMEN*, and (computation time)/*TIMEN*. If *TIMEN*=NIL, it defaults to 1. This is useful for more accurate measurement on small computations.

If *TIMETYP* is 0, *TIME* measures and prints total *real* time as well as computation time. If *TIMETYP* = 3, *TIME* measures and prints garbage collection time as well as computation time. If *TIMETYP*=T, *TIME* measures and prints the number of pagefaults.

TIME returns the value of the last evaluation of *TIMEX*.

(BOXCOUNT *TYPE* *N*)

[Function]

Returns the number of data objects of type *TYPE* allocated since this Interlisp system was created. *TYPE* can be any data type name (see *TYPENAME*, Chapter 8). If *TYPE* is NIL, it defaults to *FIXP*. If *N* is non-NIL, the corresponding counter is reset to *N*.

(CONSCOUNT *N*)

[Function]

Returns the number of CONS cells allocated since this Interlisp system was created. If *N* is non-NIL, resets the counter to *N*. Equivalent to (BOXCOUNT 'LISTP *N*).

(PAGEFAULTS)

[Function]

Returns the number of page faults since this Interlisp system was created.

BREAKDOWN

TIMEALL collects statistics for whole computations. *BREAKDOWN* is available to analyze the breakdown of computation time (or any other measureable quantity) function by function.

(BREAKDOWN *FN* ... *FN*)

[NLambda NoSpread Function]

The user calls *BREAKDOWN* giving it a list of function names (unevaluated). These functions are modified so that they keep track of various statistics.

To remove functions from those being monitored, simply *UNBREAK* (Chapter 15) the functions, thereby restoring them to their original state. To add functions, call *BREAKDOWN* on the new functions. This will not reset the counters for any functions not on the new list. However (*BREAKDOWN*) will zero the counters of all functions being monitored.

The procedure used for measuring is such that if one function calls other and both are "broken down", then the time (or whatever quantity is being measured) spent in the inner function is *not* charged to the outer function as well.

BREAKDOWN will *not* give accurate results if a function being measured is not returned from normally, e.g., a lower RETFROM (or ERROR) bypasses it. In this case, all of the time (or whatever quantity is being measured) between the time that function is entered and the time the next function being measured is entered will be charged to the first function.

(BRKDOWNRESULTS *RETURNVALUESFLG*) [Function]

BRKDOWNRESULTS prints the analysis of the statistics requested as well as the number of calls to each function. If *RETURNVALUESFLG* is non-NIL, BRKDOWNRESULTS will not to print the results, but instead return them in the form of a list of elements of the form (*FNNAME #CALLS VALUE*).

Example:

```

← (BREAKDOWN SUPERPRINT SUBPRINT COMMENT1)
(SUPERPRINT SUBPRINT COMMENT1)
← (PRETTYDEF ' (SUPERPRINT) ' FOO)
FOO.;3
← (BRKDOWNRESULTS)
FUNCTIONS    TIME      #CALLS  PER CALL    %
SUPERPRINT   8.261      365     0.023      20
SUBPRINT     31.910      141     0.226      76
COMMENT1     1.612        8     0.201        4
TOTAL        41.783      514     0.081
NIL
← (BRKDOWNRESULTS T)
((SUPERPRINT 365 8261) (SUBPRINT 141 31910) (COMMENT1 8
1612))

```

BREAKDOWN can be used to measure other statistics, by setting the following variables:

BRKDWNTYPE [Variable]

To use BREAKDOWN to measure other statistics, before calling BREAKDOWN, set the variable BRKDWNTYPE to the quantity of interest, e.g., TIME, CONSES, etc, or a list of such quantities. Whenever BREAKDOWN is called with BRKDWNTYPE not NIL, BREAKDOWN performs the necessary changes to its internal state to conform to the new analysis. In particular, if this is the first time an analysis is being run with a particular statistic, a measuring function will be defined, and the compiler will be called to compile it. The functions being broken down will be redefined to call this measuring function. When BREAKDOWN is through initializing, it sets BRKDWNTYPE back to NIL. Subsequent calls to BREAKDOWN will measure the new statistic until BRKDWNTYPE is again set and a new BREAKDOWN performed.

BRKDWNTYPES

[Variable]

The list BRKDWNTYPES contains the information used to analyze new statistics. Each entry on BRKDWNTYPES should be of the form (*TYPE FORM FUNCTION*), where *TYPE* is a statistic name (as would appear in BRKDWNTYPE), *FORM* computes the statistic, and *FUNCTION* (optional) converts the value of form to some more interesting quantity. For example, (TIME (CLOCK 2) (LAMBDA (X) (FQUOTIENT X 1000))) measures computation time and reports the result in seconds instead of milliseconds. BRKDWNTYPES currently contains entries for TIME, CONSES, PAGEFAULTS, BOXES, and FBOXES.

Example:

```

←(SETQ BRKDWNTYPE '(TIME CONSES))
(TIME CONSES)
←(BREAKDOWN MATCH CONSTRUCT)
(MATCH CONSTRUCT)
←(FLIP '(A B C D E F G H C Z) '(... $1 .. #2 ...) '(... #3
..))
(A B D E F G H Z)
←(BRKDOWNRESULTS)
FUNCTIONS  TIME      #CALLS  PER CALL  %
MATCH      0.036      1        0.036     54
CONSTRUCT  0.031      1        0.031     46
TOTAL      0.067      2        0.033
FUNCTIONS  CONSES    #CALLS  PER CALL  %
MATCH      32        1       32.000    40
CONSTRUCT  49        1       49.000    60
TOTAL      81        2       40.500
NIL

```

Occasionally, a function being analyzed is sufficiently fast that the overhead involved in measuring it obscures the actual time spent in the function. If you were using TIME, you would specify a value for *TIMEN* greater than 1 to give greater accuracy. A similar option is available for BREAKDOWN. You can specify that a function(s) be executed a multiple number of times for each measurement, and the average value reported, by including a number in the list of functions given to BREAKDOWN. For example, BREAKDOWN(EDITCOM EDIT4F 10 EDIT4E EQP) means normal breakdown for EDITCOM and EDIT4F but executes (the body of) EDIT4E and EQP 10 times each time they are called. Of course, the functions so measured must not cause any harmful side effects, since they are executed more than once for each call. The printout from BRKDOWNRESULTS will look the same as though each function were run only once, except that the measurement will be more accurate.

Another way of obtaining more accurate measurement is to expand the call to the measuring function in-line. If the value of BRKDOWNCOMPFLG is non-NIL (initially NIL), then whenever a function is broken-down, it will be redefined to

call the measuring function, and then recompiled. The measuring function is expanded in-line via an appropriate macro. In addition, whenever BRKDWNTYPE is reset, the compiler is called for *all* functions for which BRKDWNCOMPFLG was set at the time they were originally broken-down, i.e. the setting of the flag at the time a function is broken-down determines whether the call to the measuring code is compiled in-line.

GAINSPACE

If you have large programs and databases, you may sometimes find yourself in a situation where you need to obtain more space, and are willing to pay the price of eliminating some or all of the context information that the various user-assistance facilities such as the programmer's assistant, file package, CLISP, etc., have accumulated during the course of his session. The function GAINSPACE provides an easy way to selectively throw away accumulated data:

(GAINSPACE)

[Function]

Prints a list of deletable objects, allowing you to specify at each point what should be discarded and what should be retained. For example:

```
← (GAINSPACE)
purge history lists ? Yes
purge everything, or just the properties, e.g., SIDE,
LISPXPRINT, etc. ?
just the properties
discard definitions on property lists ? Yes
discard old values of variables ? Yes
erase properties ? No
erase CLISP translations? Yes
```

GAINSPACE is driven by the list GAINSPACEFORMS. Each element on GAINSPACEFORMS is of the form (*PRECHECK MESSAGE FORM KEYLST*). If *PRECHECK*, when evaluated, returns NIL, GAINSPACE skips to the next entry. For example, you will not be asked whether or not to purge the history list if it is not enabled. Otherwise, ASKUSER (Chapter 26) is called with the indicated *MESSAGE* and the (optional) *KEYLST*. If you respond **No**, i.e., ASKUSER returns N, GAINSPACE skips to the next entry. Otherwise, *FORM* is evaluated with the variable RESPONSE bound to the value of ASKUSER. In the above example, the *FORM* for the "**purge history lists**" question calls ASKUSER to ask "**purge everything, ...**" only if you had responded **Yes**. If you had responded with **Everything**, the second question would not have been asked.

The "**erase properties**" question is driven by a list SMASHPROPSMENU. Each element on this list is of the form (*MESSAGE . PROPS*). You are prompted with *MESSAGE*

(by ASKUSER), and if your response is **Yes**, *PROPS* is added to the list SMASHPROPS. The "**discard definitions on property lists**" and "**discard old values of variables**" questions also add to SMASHPROPS. You will not be prompted for any entry on SMASHPROPSMENU for which all of the corresponding properties are already on SMASHPROPS. SMASHPROPS is initially set to the value of SMASHPROPSLST. This permits you to specify in advance those properties which you always want discarded, and not be asked about them subsequently. After finishing all the entries on GAINSPACEFORMS, GAINSPACE checks to see if the value of SMASHPROPS is non-NIL, and if so, does a MAPATOMS, i.e., looks at every atom in the system, and erases the indicated properties.

You can change or add new entries to GAINSPACEFORMS or SMASHPROPSMENU, so that GAINSPACE can also be used to purge structures that your programs have accumulated.

Using Data Types Instead of Records

If a program uses large numbers of large data structures, there are several advantages to representing them as user data types rather than as list structures. The primary advantage is increased speed: accessing and setting the fields of a data type can be significantly faster than walking through a list with repeated CARS and CDRs. Also,

compiled code for referencing data types is usually smaller. Finally, by reducing the number of objects created (one object against many list cells), this can reduce the expense of garbage collection.

User data types are declared by using the `DATATYPE` record type (Chapter 8). If a list structure has been defined using the `RECORD` record type (Chapter 8), and all accessing operations are written using the record package's `fetch`, `replace`, and `create` operations, changing from `RECORD`s to `DATATYPE`s only requires editing the record declaration (using `EDITREC`, Chapter 8) to replace declaration type `RECORD` by `DATATYPE`, and recompiling.

Note: There are some minor disadvantages with allocating new data types: First, there is an upper limit on the number of data types which can exist. Also, space for data types is allocated a page at a time, so each data type has at least one page assigned to it, which may be wasteful of space if there are only a few examples of a given data type. These problems should not effect most applications programs.

Using Incomplete File Names

Currently, Interlisp allows you to specify an open file by giving the file name. If the file name is incomplete (it doesn't have the device/host, directory, name, extension, and version number all supplied), the system converts it to a complete file name, by supplying defaults and searching through directories (which may be on remote file servers), and then searches the open streams for one corresponding to that file name. This file name-completion process happens whenever any I/O function is given an incomplete file name, which can cause a serious performance problem if I/O operations are done repeatedly. In general, it is much faster to convert an incomplete file name to a stream once, and use the stream from then on. For example, suppose a file is opened with `(SETQ STRM (OPENSTREAM 'MYNAME 'INPUT))`. After doing this, `(READC 'MYNAME)` and `(READC STRM)` would both work, but `(READC 'MYNAME)` would take longer (sometimes orders of magnitude longer). This could seriously effect the performance if a program which is doing many I/O operations.

At some point in the future, when multiple streams are supported to a single file, the feature of mapping file names to streams will be removed. This is yet another reason why programs should use streams as handles to open files, instead of file names.

For more information on efficiency considerations when using files, see Chapter 24.

Using "Fast" and "Destructive" Functions

Among the functions used for manipulating objects of various data types, there are a number of functions which have "fast" and "destructive" versions. You should be aware of what these functions do, and when they should be used.

"Fast" functions: By convention, a function named by prefixing an existing function name with `F` indicates that the new function is a "fast" version of the old. These usually have the same definitions as the slower versions, but they compile open and run without any "safety" error checks. For example, `FNTH` runs faster than `NTH`, however, it does not make as many checks (for lists ending with anything but `NIL`, etc). If these functions are given arguments that are not in the form that they expect, their behavior is unpredictable; they may run forever, or cause a system error. In general, you should only use "fast" functions in code that has already been completely debugged, to speed it up.

"Destructive" functions: By convention, a function named by prefixing an existing function with `D` indicates the new function is a "destructive" version of the old one, which does not make any new structure but cannibalizes its argument(s). For example, `REMOVE` returns a copy of a list with a particular element removed, but `DREMOVE` actually changes the list structure of the list. (Unfortunately, not all destructive functions follow this naming convention: the destructive version of `APPEND` is `NCONC`.) You should be careful when using destructive functions that they do not inadvertently change data structures.

22. PROCESSES

The Medley Process mechanism provides an environment in which multiple Lisp processes can run in parallel. Each executes in its own stack space, but all share a global address space. The current process implementation is cooperative; i.e., process switches happen voluntarily, either when the process in control has nothing to do or when it is in a convenient place to pause. There is no preemption or guaranteed service, so you cannot run something demanding (e.g., Chat) at the same time as something that runs for long periods without yielding control. Keyboard input and network operations block with great frequency, so processes currently work best for highly interactive tasks (editing, making remote files).

In Medley, the process mechanism is already turned on, and is expected to stay on during normal operations, as some system facilities (in particular, most network operations) require it. However, under exceptional conditions, the following function can be used to turn the world off and on:

(**PROCESSWORLD** *FLG*) [Function]

Starts up the process world, or if *FLG* = OFF, kills all processes and turns it off. Normally does not return. The environment starts out with two processes: a top-level EVALQT (the initial "tty" process) and the "background" process, which runs the window mouse handler and other system background tasks.

PROCESSWORLD is intended to be called at the top level of Interlisp, not from within a program. It does not toggle some sort of switch; rather, it constructs some new processes in a new part of the stack, leaving any callers of PROCESSWORLD in a now inaccessible part of the stack. Calling (PROCESSWORLD 'OFF) is the only way the call to PROCESSWORLD ever returns.

(**HARDRESET**) [Function]

Resets the whole world, and rebuilds the stack from scratch. This is "harder" than doing RESET to every process, because it also resets system internal processes (such as the keyboard handler).

HARDRESET automatically turns the process world on (or resets it if it was on), unless the variable AUTOPROCESSFLG is NIL.

Creating and Destroying Processes

(**ADD.PROCESS** *FORM PROP VALUE ... PROP VALUE*) [NoSpread Function]

Creates a new process evaluating *FORM*, and returns its process handle. The process's stack environment is the top level, i.e., the new process does not have access to the environment in which ADD.PROCESS was called; all such information must be passed as arguments in *FORM*. The process runs until *FORM* returns or the process is explicitly

INTERLISP-D REFERENCE MANUAL

deleted. An untrapped error within the process also deletes the process (unless its `RESTARTABLE` property is T), in which case a message is printed to that effect.

The remaining arguments are alternately property names and values. Any property/value pairs acceptable to `PROCESSPROP` may be given, but the following two are directly relevant to `ADD . PROCESS`:

NAME	Value can be a symbol or a string; if not given, the process name is taken from <code>(CAR FORM)</code> . <code>ADD . PROCESS</code> may pack the name with a number to make it unique. Process names are treated as case-insensitive strings. This name is solely for the convenience of manipulating processes at Lisp type-in; e.g., the name can be given as the <code>PROC</code> argument to most process functions, and the name appears in menus of processes. However, programs should normally only deal in process handles, both for efficiency and to avoid the confusion that can result if two processes have the same defining form.
SUSPEND	If the value is non-NIL, the new process is created but then immediately suspended; i.e., the process does not actually run until woken by a <code>WAKE . PROCESS</code> (below).

`(PROCESSPROP PROC PROP NEWVALUE)`

[NoSpread Function]

Used to get or set the values of certain properties of process `PROC`, in a manner analogous to `WINDOWPROP`. If `NEWVALUE` is supplied (including if it is NIL), property `PROP` is given that value. In all cases, returns the old value of the property. The following properties have special meaning for processes; all others are uninterpreted:

NAME	Value is a symbol used for identifying the process to the user.
FORM	Value is the Lisp form used to start the process (readonly).
RESTARTABLE	Value is a flag indicating the disposition of the process following errors or hard resets: NIL or NO (the default): If an untrapped error (or Control-E or Control-D) causes its form to be exited, the process is deleted. The process is also deleted if a <code>HARDRESET</code> (or Control-D from <code>RAID</code>) occurs, causing the entire Process world to be reinitialized. T or YES: The process is automatically restarted on errors or <code>HARDRESET</code> . This is the normal setting for persistent "background" processes, such as the mouse process, that can safely restart themselves on errors. <code>HARDRESET</code> : The process is deleted as usual if an error causes its form to be exited, but it is restarted on a

PROCESSES

HARDRESET. This setting is preferred for persistent processes for which an error is an unusual condition, one that might repeat itself if the process were simply blindly restarted.

RESTARTFORM If the value is non-NIL, it is the form used if the process is restarted (instead of the value of the **FORM** property). Of course, the process must also have a non-NIL **RESTARTABLE** prop for this to have any effect.

BEFOREEXIT If the value is the atom **DON'T**, it will not be interrupted by a **LOGOUT**. If **LOGOUT** is attempted before the process finishes, a message will appear saying that Interlisp is waiting for the process to finish. If you want the **LOGOUT** to proceed without waiting, you must use the process status window (from the background menu) to delete the process.

AFTEREXIT Value indicates the disposition of the process following a resumption of Lisp after some exit (**LOGOUT**, **SYSOUT**, **MAKESYS**). Possible values are:

DELETE: Delete the process.

SUSPEND: Suspend the process; i.e., do not let it run until it is explicitly woken.

An event: Cause the process to be suspended waiting for the event (See the Events section below).

INFOHOOK Value is a function or form used to provide information about the process, in conjunction with the **INFO** command in the process status window (see the Process Status Window section below).

WINDOW Value is a window associated with the process, the process's "main" window. Used to switch the tty process to this process when you click in this window (see the Switching the TTY Process section below).

Setting the **WINDOW** property does not set the primary I/O stream (**NIL**) or the terminal I/O stream (**T**) to the window. When a process is created, I/O operations to the **NIL** or **T** stream will cause a new window to appear. **TTYDISPLAYSTREAM** (see Chapter 26) should be used to set the terminal I/O stream of a process to a specific window.

TTYENTRYFN Value is a function that is applied to the process when the process is made the tty process (see the Switching the TTY Process section below).

INTERLISP-D REFERENCE MANUAL

TTYEXITFN Value is a function that is applied to the process when the process ceases to be the tty process (see the Switching the TTY Process section below).

(**THIS.PROCESS**) [Function]

Returns the handle of the currently running process, or NIL if the Process world is turned off.

(**DEL.PROCESS** *PROC*) [Function]

Deletes process *PROC*. *PROC* may be a process handle (returned by **ADD.PROCESS**), or its name. If *PROC* is the currently running process, **DEL.PROCESS** does not return!

(**PROCESS.RETURN** *VALUE*) [Function]

Terminates the currently running process, causing it to "return" *VALUE*. There is an implicit **PROCESS.RETURN** around the *FORM* argument given to **ADD.PROCESS**, so that normally a process can finish by simply returning; **PROCESS.RETURN** is supplied for earlier termination.

(**PROCESS.RESULT** *PROCESS WAITFORRESULT*) [Function]

If *PROCESS* has terminated, returns the value, if any, that it returned. This is either the value of a **PROCESS.RETURN** or the value returned from the form given to **ADD.PROCESS**. If the process was aborted, the value is NIL. If *WAITFORRESULT* is true, **PROCESS.RESULT** blocks until *PROCESS* finishes, if necessary; otherwise, it returns NIL immediately if *PROCESS* is still running. *PROCESS* must be the actual process handle returned from **ADD.PROCESS**, not a process name, as the association between handle and name disappears when the process finishes (and the process handle itself is then garbage collected if no one else has a pointer to it).

(**PROCESS.FINISHEDP** *PROCESS*) [Function]

True if *PROCESS* has terminated. The value returned is an indication of how it finished: **NORMAL** or **ERROR**.

(**PROCESSP** *PROC*) [Function]

True if *PROC* is the handle of an active process, i.e., one that has not yet finished.

(**RELPROCESSP** *PROCHANDLE*) [Function]

True if *PROCHANDLE* is the handle of a deleted process. This is analogous to **RELSTKP**. It differs from **PROCESS.FINISHEDP** in that it never causes an error, while **PROCESS.FINISHEDP** can cause an error if its *PROC* argument is not a process at all.

(**RESTART.PROCESS** *PROC*) [Function]

Unwinds *PROC* to its top level and reevaluates its form. This is effectively a **DEL.PROCESS** followed by the original **ADD.PROCESS**.

PROCESSES

(**MAP.PROCESSES** *MAPFN*)

[Function]

Maps over all processes, calling *MAPFN* with three arguments: the process handle, its name, and its form.

(**FIND.PROCESS** *PROC ERRORFLG*)

[Function]

If *PROC* is a process handle or the name of a process, returns the process handle for it, else NIL. If *ERRORFLG* is T, generates an error if *PROC* is not, and does not name, a live process.

Process Control Constructs

(**BLOCK** *MSECSWAIT TIMER*)

[Function]

Yields control to the next waiting process, assuming any is ready to run. If *MSECSWAIT* is specified, it is a number of milliseconds to wait before returning, or T, meaning wait forever (until explicitly woken). Alternatively, *TIMER* can be given as a millisecond timer (as returned by *SETUPTIMER*, Chapter 12) of an absolute time at which to wake up. In any of those cases, the process enters the *waiting* state until the time limit is up. **BLOCK** with no arguments leaves the process in the *runnable* state, i.e., it returns as soon as every other runnable process of the same priority has had a chance.

BLOCK can be aborted by interrupts such as Control-D, Control-E, or Control-B. **BLOCK** will return before its timeout is completed, if the process is woken by *WAKE.PROCESS*, *PROCESS.EVAL*, or *PROCESS.APPLY*.

(**DISMISS** *MSECSWAIT TIMER NOBLOCK*)

[Function]

DISMISS is used to dismiss the current process for a given period of time. Similar to **BLOCK**, except that:

- **DISMISS** is guaranteed not to return until the specified time has elapsed
- *MSECSWAIT* cannot be T to wait forever
- If *NOBLOCK* is T, **DISMISS** will not allow other processes to run, but will busy-wait until the amount of time given has elapsed.

(**WAKE.PROCESS** *PROC STATUS*)

[Function]

Explicitly wakes process *PROC*, i.e., makes it *runnable*, and causes its call to **BLOCK** (or other waiting function) to return *STATUS*. This is one simple way to notify a process of some happening; however, note that if *WAKE.PROCESS* is applied to a process more than once before the process actually gets its turn to run, it sees only the latest *STATUS*.

INTERLISP-D REFERENCE MANUAL

(**SUSPEND.PROCESS** *PROC*) [Function]

Blocks process *PROC* indefinitely, i.e., *PROC* will not run until it is woken by a **WAKE.PROCESS**.

The following three functions allow access to the stack context of some other process. They require a little bit of care, and are computationally non-trivial, but they do provide a more powerful way of manipulating another process than **WAKE.PROCESS** allows.

(**PROCESS.EVALV** *PROC VAR*) [Function]

Performs (**EVALV** *VAR*) in the stack context of *PROC*.

(**PROCESS.EVAL** *PROC FORM WAITFORRESULT*) [Function]

Evaluates *FORM* in the stack context of *PROC*. If *WAITFORRESULT* is true, blocks until the evaluation returns a result, else allows the current process to run in parallel with the evaluation. Any errors that occur will be in the context of *PROC*, so be careful. In particular, note that

```
(PROCESS.EVAL PROC ' (NLSETQ (FOO)))
```

and

```
(NLSETQ (PROCESS.EVAL PROC ' (FOO)))
```

behave quite differently if *FOO* causes an error. And it is quite permissible to intentionally cause an error in *proc* by performing

```
(PROCESS.EVAL PROC ' (ERROR!))
```

If *WAITFORRESULT* is true and the computation in the other process aborts or the other process is killed **PROCESS.EVAL** returns :ABORTED

After *FORM* is evaluated in *PROC*, the process *PROC* is woken up, even if it was running **BLOCK** or **AWAIT.EVENT**. This is necessary because an event of interest may have occurred while the process was evaluating *FORM*.

(**PROCESS.APPLY** *PROC FN ARGS WAITFORRESULT*) [Function]

Performs (**APPLY** *FN ARGS*) in the stack context of *PROC*. Note the same warnings as with **PROCESS.EVAL**.

Events

An "event" is a synchronizing primitive used to coordinate related processes, typically producers and consumers. Consumer processes can "wait" on events, and producers "notify" events.

PROCESSES

(CREATE.EVENT NAME) [Function]

Returns an instance of the `EVENT` datatype, to be used as the event argument to functions listed below. `NAME` is arbitrary, and is used for debugging or status information.

(AWAIT.EVENT EVENT TIMEOUT TIMERP) [Function]

Suspends the current process until `EVENT` is notified, or until a timeout occurs. If `TIMEOUT` is `NIL`, there is no timeout. Otherwise, timeout is either a number of milliseconds to wait, or, if `TIMERP` is `T`, a millisecond timer set to expire at the desired time using `SETUPTIMER` (see Chapter 12).

(NOTIFY.EVENT EVENT ONCEONLY) [Function]

If there are processes waiting for `EVENT` to occur, causes those processes to be placed in the running state, with `EVENT` returned as the value from `AWAIT.EVENT`. If `ONCEONLY` is true, only runs the first process waiting for the event (this should only be done if the programmer knows that there can only be one process capable of responding to the event at once).

The meaning of an event is up to the programmer. In general, however, the notification of an event is merely a hint that something of interest to the waiting process has happened; the process should still verify that the conceptual event actually occurred. That is, *the process should be written so that it operates correctly even if woken up before the timeout and in the absence of the notified event*. In particular, the completion of `PROCESS.EVAL` and related operations in effect wakes up the process in which they were performed, since there is no secure way of knowing whether the event of interest occurred while the process was busy performing the `PROCESS.EVAL`.

There is currently one class of system-defined events, used with the network code. Each Pup and NS socket has associated with it an event that is notified when a packet arrives on the socket; the event can be obtained by calling `PUPSOCKETEVENT` or `NSOCKETEVENT`, respectively (see Chapter 30).

Monitors

It is often the case that cooperating processes perform operations on shared structures, and some mechanism is needed to prevent more than one process from altering the structure at the same time. Some languages have a construct called a monitor, a collection of functions that access a common structure with mutual exclusion provided and enforced by the compiler via the use of monitor locks. Medley has taken this implementation notion as the basis for a mutual exclusion capability suitable for a dynamically-scoped environment.

A monitorlock is an object created by you and associated with (e.g., stored in) some shared structure that is to be protected from simultaneous access. To access the structure, a program waits for the lock to be free, then takes ownership of the lock, accesses the structure, then releases the lock. The functions and macros below are used:

INTERLISP-D REFERENCE MANUAL

(CREAT.MONITORLOCK NAME) [Function]

Returns an instance of the MONITORLOCK datatype, to be used as the lock argument to functions listed below. *NAME* is arbitrary, and is used for debugging or status information.

(WITH.MONITOR LOCK FORM ... FORM) [Macro]

Evaluates *FORM ... FORM* while owning *LOCK*, and returns the value of *FORM*. This construct is implemented so that the lock is released even if the form is exited via error (currently implemented with RESETLST).

Ownership of a lock is dynamically scoped: if the current process already owns the lock (e.g., if the caller was itself inside a WITH.MONITOR for this lock), WITH.MONITOR does not wait for the lock to be free before evaluating *FORM ... FORM*.

(WITH.FAST.MONITOR LOCK FORM ... FORM) [Macro]

Like WITH.MONITOR, but implemented without the RESETLST. User interrupts (e.g., Control-E) are inhibited during the evaluation of *FORM ... FORM*.

Programming restriction: the evaluation of *FORM ... FORM* must not error (the lock would not be released). This construct is mainly useful when the forms perform a small, safe computation that never errors and need never be interrupted.

(MONITOR.AWAIT.EVENT RELEASELOCK EVENT TIMEOUT TIMERP) [Function]

For use in blocking inside a monitor. Performs (AWAIT.EVENT *EVENT TIMEOUT TIMERP*), but releases *RELEASELOCK* first, and reobtains the lock (possibly waiting) on wakeup.

Typical use for MONITOR.AWAIT.EVENT: A function wants to perform some operation on *FOO*, but only if it is in a certain state. It has to obtain the lock on the structure to make sure that the state of the structure does not change between the time it tests the state and performs the operation. If the state turns out to be bad, it then waits for some other process to make the state good, meanwhile releasing the lock so that the other process can alter the structure.

```
(WITH.MONITOR FOO-LOCK
  (until CONDITION-OF-FOO
    do (MONITOR.AWAIT.EVENT FOO-LOCK EVENT-FOO-
      CHANGED TIMEOUT) )
    OPERATE-ON-FOO)
```

It is sometimes convenient for a process to have WITH.MONITOR at its top level and then do all its interesting waiting using MONITOR.AWAIT.EVENT. Not only is this often cleaner, but in the present implementation in cases where the lock is frequently accessed, it saves the RESETLST overhead of WITH.MONITOR.

Programming restriction: There must not be an ERRORSET between the enclosing WITH.MONITOR and the call to MONITOR.AWAIT.EVENT such that the ERRORSET would catch an ERROR! and continue inside the monitor, for the lock would not have been reobtained. (The reason for this restriction is that, although MONITOR.AWAIT.EVENT

PROCESSES

won't itself error, you could have caused an error with an interrupt, or a `PROCESS.EVAL` in the context of the waiting process that produced an error.)

On rare occasions it may be useful to manipulate monitor locks directly. The following two functions are used in the implementation of `WITH.MONITOR`:

(OBTAIN.MONITORLOCK *LOCK DONTWAIT UNWINDSAVE*) [Function]

Takes possession of *LOCK*, waiting if necessary until it is free, unless *DONTWAIT* is true, in which case it returns `NIL` immediately. If *UNWINDSAVE* is true, performs a `RESETSAVE` to be unwound when the enclosing `RESETLST` exits. Returns *LOCK* if *LOCK* was successfully obtained, `T` if the current process already owned *LOCK*.

(RELEASE.MONITORLOCK *LOCK EVENIFNOTMINE*) [Function]

Releases *LOCK* if it is owned by the current process, and wakes up the next process, if any, waiting to obtain the lock.

If *EVENIFNOTMINE* is non-`NIL`, the lock is released even if it is not owned by the current process.

When a process is deleted, any locks it owns are released.

Global Resources

The biggest source of problems in the multi-processing environment is the matter of global resources. Two processes cannot both use the same global resource if there can be a process switch in the middle of their use (currently this means calls to `BLOCK`, but ultimately with a preemptive scheduler means anytime). Thus, user code should be wary of its own use of global variables, if it ever makes sense for the code to be run in more than one process at a time. "State" variables private to a process should generally be bound in that process; structures that are shared among processes (or resources used privately but expensive to duplicate per process) should be protected with monitor locks or some other form of synchronization.

Aside from user code, however, there are many *system* global variables and resources. Most of these arise historically from the single-process Interlisp-10 environment, and will eventually be changed in Medley to behave appropriately in a multi-processing environment. Some have already been changed, and are described below. Two other resources not generally thought of as global variables—the keyboard and the mouse—are particularly idiosyncratic, and are discussed in the next section.

The following resources, which are global in Interlisp-10, are allocated per process in Medley: primary input and output (the streams affected by `INPUT` and `OUTPUT`), terminal input and output (the streams designated by the name `T`), the primary read table and primary terminal table, and dribble files. Thus, each process can print to its own primary output, print to the terminal, read from a different primary input, all without interfering with another process's reading and printing.

INTERLISP-D REFERENCE MANUAL

Each process begins life with its primary and terminal input/output streams set to a dummy stream. If the process attempts input or output using any of those dummy streams, e.g., by calling (READ T), or (PRINT & T), a tty window is automatically created for the process, and that window becomes the primary input/output and terminal input/output for the process. The default tty window is created at or near the region specified in the variable DEFAULTTTYREGION.

A process can, of course, call TTYDISPLAYSTREAM explicitly to give itself a tty window of its own choosing, in which case the automatic mechanism never comes into play. Calling TTYDISPLAYSTREAM when a process has no tty window not only sets the terminal streams, but also sets the primary input and output streams to be that window, assuming they were still set to the dummy streams.

(HASTTYWINDOWP *PROCESS*)

[Function]

Returns T if the process *PROCESS* has a tty window; NIL otherwise. If *PROCESS* is NIL, it defaults to the current process.

Other system resources that are typically changed by RESETFORM, RESETLST, or RESETVARS are all global entities. In the multiprocessing environment, these constructs are suspect, as there is no provision for "undoing" them when a process switch occurs. For example, in the current release of Medley, it is not possible to set the print radix to 8 inside only one process, as the print radix is a global entity.

Note that RESETFORM and similar expressions are perfectly valid in the process world, and even quite useful, when they manipulate things strictly within one process. The process world is arranged so that deleting a process also unwinds any RESETxxx expressions that were performed in the process and are still waiting to be unwound, exactly as if a Control-D had reset the process to the top. Additionally, there is an implicit RESETLST at the top of each process, so that RESETSAVE can be used as a way of providing "cleanup" functions for when a process is deleted. For these, the value of RESETSTATE (see Chapter 14) is NIL if the process finished normally, ERROR if it was aborted by an error, RESET if the process was explicitly deleted, and HARDRESET if the process is being restarted after a HARDRESET or a RESTART . PROCESS.

Typein and the TTY Process

There is one global resource, the keyboard, that is particularly problematic to share among processes. Consider, for example, having two processes both performing (READ T). Since the keyboard input routines block while there is no input, both processes would spend most of their time blocking, and it would simply be a matter of chance which process received each character of type-in.

To resolve such dilemmas, the system designates a distinguished process, termed the *tty process*, that is assumed to be the process that is involved in terminal interaction. Any type-in from the keyboard goes to that process. If a process other than the tty process requests keyboard input, it blocks until it becomes the tty process. When the tty process is switched (in any of the ways described further below), any typeahead that occurred before the switch is saved and associated with the current tty

PROCESSES

process. Thus, it is always the case that keystrokes are sent to the process that is the tty process at the time of the keystrokes, regardless of when that process actually gets around to reading them.

BACKGROUND FNS

[Variable]

A list of functions to call "in the background". The system runs a process (called "BACKGROUND") whose sole task is to call each of the functions on the list BACKGROUND FNS repeatedly. Each element is the name of a function of no arguments. This is a good place to put cheap background tasks that only do something once in a while and hence do not want to spend their own separate process on it. However, note that it is considered good citizenship for a background function with a time-consuming task to spawn a separate process to do it, so that the other background functions are not delayed.

TTYBACKGROUND FNS

[Variable]

This list is like BACKGROUND FNS, but the functions are only called while in a tty input wait. That is, they always run in the tty process, and only when the user is not actively typing. For example, the flashing caret is implemented by a function on this list. Again, functions on this list should spend very little time (much less than a second), or else spawn a separate process.

It is less immediately obvious how to handle keyboard interrupt characters, as their action is asynchronous and not always tied to type-in. Interrupt handling is described in the Handling of Interrupts section below.

Switching the TTY Process

Any process can make itself be the tty process by calling `TTY . PROCESS`.

(TTY . PROCESS *PROC*)

[Function]

Returns the handle of the current tty process. In addition, if *PROC* is non-NIL, makes it be the tty process. The special case of *PROC* = T is interpreted to mean the executive process; this is sometimes useful when a process wants to explicitly give up being the tty process.

(TTY . PROCESSP *PROC*)

[Function]

True if *PROC* is the tty process; *PROC* defaults to the running process. Thus, (TTY . PROCESSP) is true if the caller is the tty process.

(WAIT . FOR . TTY *MSECS NEEDWINDOW*)

[Function]

Efficiently waits until (TTY . PROCESSP) is true. WAIT . FOR . TTY is called internally by the system functions that read from the terminal; user code thus need only call it in special cases.

INTERLISP-D REFERENCE MANUAL

If *MSECS* is non-NIL, it is the number of milliseconds to wait before timing out. If *WAIT.FOR.TTY* times out before (*TTY.PROCESSP*) is true, it returns NIL, otherwise it returns T. If *MSECS* is NIL, *WAIT.FOR.TTY* will not time out.

If *NEEDWINDOW* is non-NIL, *WAIT.FOR.TTY* opens a TTY window for the current process if one isn't already open.

WAIT.FOR.TTY spawns a new mouse process if called under the mouse process (see *SPAWN.MOUSE*, in the Keeping the Mouse Alive section below).

In some cases, such as in functions invoked as a result of mouse action or a user's typed-in call, it is reasonable for the function to invoke *TTY.PROCESS* itself so that it can take subsequent user type in. In other cases, however, this is too undisciplined; it is desirable to let you designate which process type-in should be directed to. This is most conveniently done by mouse action.

The system supports the model that "to type to a process, you click in its window." To cooperate with this model, any process desiring keyboard input should put its process handle as the *PROCESS* property of its window(s). To handle the common case, the function *TTYDISPLAYSTREAM* does this automatically when the *ttydisplaystream* is switched to a new window. A process can own any number of windows; clicking in any of those windows gives the process the *tty*.

This mechanism suffices for most casual process writers. For example, if a process wants all its input/output interaction to occur in a particular window that it has created, it should just make that window be its *tty* window by calling *TTYDISPLAYSTREAM*. Thereafter, it can *PRINT* or *READ* to/from the T stream; if the process is not the *tty* process at the time that it calls *READ*, it will block until the user clicks in the window.

For those needing tighter control over the *tty*, the default behavior can be overridden or supplemented. The remainder of this section describes the mechanisms involved.

There is a window property *WINDOWENTRYFN* that controls whether and how to switch the *tty* to the process owning a window. The mouse handler, before invoking any normal *BUTTONEVENTFN*, specifically notices the case of a button going down in a window that belongs to a process (i.e., has a *PROCESS* window property) that is not the *tty* process. In this case, it invokes the window's *WINDOWENTRYFN* of one argument (*WINDOW*). *WINDOWENTRYFN* defaults to *GIVE.TTY.PROCESS*:

(GIVE.TTY.PROCESS WINDOW)

[Function]

If *WINDOW* has a *PROCESS* property, performs (*TTY.PROCESS* (*WINDOWPROP WINDOW* 'PROCESS)) and then invokes *WINDOW*'s *BUTTONEVENTFN* function (or *RIGHTBUTTONFN* if the right button is down).

There are some cases where clicking in a window does not always imply that the user wants to talk to that window. For example, clicking in a text editor window with a shift key held down means to "shift-select" some piece of text into the input buffer of the *current* *tty* process. The editor supports this by supplying a *WINDOWENTRYFN* that performs *GIVE.TTY.PROCESS* if no shift key is down, but goes into its shift-select mode, without changing the *tty* process, if a shift key is down. The shift-select

PROCESSES

mode performs a `BKSYSBUF` of the selected text when the shift key is let up, the `BKSYSBUF` feeding input to the current tty process.

Sometimes a process wants to be notified when it becomes the tty process, or stops being the tty process. To support this, there are two process properties, `TTYEXITFN` and `TTYENTRYFN`. The actions taken by `TTY.PROCESS` when it switches the tty to a new process are as follows: the former tty process's `TTYEXITFN` is called with two arguments (`OLDTTYPROCESS NEWTTYPROCESS`); the new process is made the tty process; finally, the new tty process's `TTYENTRYFN` is called with two arguments (`NEWTTYPROCESS OLDTTYPROCESS`). Normally the `TTYENTRYFN` and `TTYEXITFN` need only their first argument, but the other process involved in the switch is supplied for completeness. In the present system, most processes want to interpret the keyboard in the same way, so it is considered the responsibility of any process that changes the keyboard interpretation to restore it to the normal state by its `TTYEXITFN`.

A window is "owned" by the last process that anyone gave as the window's `PROCESS` property. Ordinarily there is no conflict here, as processes tend to own disjoint sets of windows (though, of course, cooperating processes can certainly try to confuse each other). The only likely problem arises with that most global of windows, `PROMPTWINDOW`. Programs should not be tempted to read from `PROMPTWINDOW`. This is not usually necessary anyway, as the first attempt to read from `T` in a process that has not set its `TTYDISPLAYSTREAM` to its own window causes a tty window to be created for the process (see the Global Resources section above).

Handling of Interrupts

At the time that a keyboard interrupt character (see Chapter 29) is struck, any process could be running, and some decision must be made as to which process to actually interrupt. To the extent that keyboard interrupts are related to type-in, most interrupts are taken in the tty process; however, the following are handled specially:

RESET (initially Control-D)

ERROR (initially Control-E)

These interrupts are taken in the mouse process, if the mouse is not in its idle state; otherwise they are taken in the tty process. Thus, Control-E can be used to abort some mouse-invoked window action, such as the Shape command. As a consequence, note that if the mouse invokes some lengthy computation that the user thinks of as "background", Control-E still aborts it, even though that may not have been what the user intended. Such lengthy computations, for various reasons, should generally be performed by spawning a separate process to perform them. The **RESET** interrupt in a process other than the executive is interpreted exactly as if an error unwound the process to its top level: if the process was

INTERLISP-D REFERENCE MANUAL

	designated <code>RESTARTABLE = T</code> , it is restarted; otherwise it is killed.
HELP (initially Control-G)	A menu of processes is presented to the user, who is asked to select which one the interrupt should occur in. The current tty process appears with a * next to its name at the top of the menu. The menu also includes an entry "[Spawn Mouse]", for the common case of needing a mouse because the mouse process is currently tied up running someone's <code>BUTTONEVENTFN</code> ; selecting this entry spawns a new mouse process, and no break occurs.
BREAK (initially Control-B)	Performs the <code>HELP</code> interrupt in the mouse process, if the mouse is not in its idle state; otherwise it is performed in the tty process.
RUBOUT (initially DELETE)	This interrupt clears typeahead in <i>all</i> processes.
RAID, STACK OVERFLOW STORAGE FULL	These interrupts always occur in whatever process was running at the time the interrupt struck. In the cases of <code>STACK OVERFLOW</code> and <code>STORAGE FULL</code> , this means that the interrupt is more likely to strike in the offending process (especially if it is a "runaway" process that is not blocking). Note, however, that this process is still not necessarily the guilty party; it could be an innocent bystander that just happened to use up the last of a resource prodigiously consumed by some other process.

Keeping the Mouse Alive

Since the window mouse handler runs in its own process, it is not available while a window's `BUTTONEVENTFN` function (or any of the other window functions invoked by mouse action) is running. This leads to two sorts of problems: (1) a long computation underneath a `BUTTONEVENTFN` deprives the user of the mouse for other purposes, and (2) code that runs as a `BUTTONEVENTFN` cannot rely on other `BUTTONEVENTFN`s running, which means that there some pieces of code that run differently from normal when run under the mouse process. These problems are addressed by the following functions:

PROCESSES

(**SPAWN.MOUSE** —)

[Function]

Spawns another mouse process, allowing the mouse to run even if it is currently "tied up" under the current mouse process. This function is intended mainly to be typed in at the Lisp executive when you notice the mouse is busy.

(**ALLOW.BUTTON.EVENTS**)

[Function]

Performs a (**SPAWN.MOUSE**) only when called underneath the mouse process. This should be called (once, on entry) by any function that relies on **BUTTONEVENTFN**s for completion, if there is any possibility that the function will itself be invoked by a mouse function.

It never hurts, at least logically, to call **SPAWN.MOUSE** or **ALLOW.BUTTON.EVENTS** needlessly, as the mouse process arranges to quietly kill itself if it returns from the user's **BUTTONEVENTFN** and finds that another mouse process has sprung up in the meantime. (There is, of course, some computational expense.)

Process Status Window

The background menu command **PSW** (see Chapter 27) and the function **PROCESS.STATUS.WINDOW** (below) create a "Process Status Window", that allows you to examine and manipulate all of the existing processes:

SPACEWINDOW		
Tedit		
MOUSE		
ERIS#LEAF		
\10MBWATCHER		
EXEC		
\NSGATELISTENER		
\PUPGATELISTENER		
\TIMER.PROCESS		
BACKGROUND		
BT	WHO?	KILL
BTV	KBD←	RESTART
BTV*	INFO	WAKE
BTV!	BREAK	SUSPEND

The window consists of two menus. The top menu lists all the processes at the moment. Commands in the bottom menu operate on the process selected in the top menu (**EXEC** in the example above). The commands are:

BT, BTV, BTV*, BTV! Displays a backtrace of the selected process.

WHO? Changes the selection to the tty process, i.e., the one currently in control of the keyboard.

INTERLISP-D REFERENCE MANUAL

KBD←	Associates the keyboard with the selected process; i.e., makes the selected process be the tty process.
INFO	If the selected process has an INFOHOOK property, calls it. The hook may be a function, which is then applied to two arguments, the process and the button (LEFT or MIDDLE) used to invoke INFO, or a form, which is simply EVAL'ed. The APPLY or EVAL happens in the context of the selected process, using PROCESS.APPLY or PROCESS.EVAL. The INFOHOOK process property can be set using PROCESSPROP (see the Creating and Destroying Processes section above).
BREAK	Enter a break under the selected process. This has the side effect of waking the process with the value returned from the break.
KILL	Deletes the selected process.
RESTART	Restarts the selected process.
WAKE	Wakes the selected process. Prompts for a value to wake it with (see WAKE.PROCESS).
SUSPEND	Suspends the selected process; i.e., causes it to block indefinitely (until explicitly woken).

(PROCESS.STATUS.WINDOW *WHERE*)

[Function]

Puts up a process status window that provides several debugging commands for manipulating running processes. If the window is already up, PROCESS.STATUS.WINDOW refreshes it. If *WHERE* is a position, the window is placed in that position; otherwise, you are prompted for a position.

Currently, the process status window runs under the mouse process, like other menus, so if the mouse is unavailable (e.g., a mouse function is performing an extensive computation), you may be unable to use the process status window (you can try SPAWN.MOUSE, of course).

Non-Process Compatibility

This section describes some considerations for authors of programs that ran in the old single-process Medley environment, and now want to make sure they run properly in the multi-processing world. The biggest problem to watch out for is code that runs underneath the mouse handler. Writers of mouse handler functions should remember that in the process world the mouse handler runs in its own process, and hence (a) you cannot depend on finding information on the stack (stash it in the window instead), and (b) while your function is running, the mouse is not available (if you have any non-trivial computation to do, spawn a process to do it, notify one of your existing processes to do it, or use PROCESS.EVAL to run it under some other process).

PROCESSES

The following functions are meaningful even if the process world is not on: `BLOCK` (invokes the system background routine, which includes handling the mouse); `TTY.PROCESS`, `THIS.PROCESS` (both return `NIL`); and `TTY.PROCESSP` (returns `T`, i.e., anyone is allowed to take tty input). In addition, the following two functions exist in both worlds:

`(EVAL.AS.PROCESS FORM)` [Function]

Same as `(ADD.PROCESS FORM 'RESTARTABLE 'NO)`, when processes are running, `EVAL` when not. This is highly recommended for mouse functions that perform any non-trivial activity.

`(EVAL.IN.TTY.PROCESS FORM WAITFORRESULT)` [Function]

Same as `(PROCESS.EVAL (TTY.PROCESS) FORM WAITFORRESULT)`, when processes are running, `EVAL` when not.

Most of the process functions that do not take a process argument can be called even if processes aren't running. `ADD.PROCESS` creates, but does not run, a new process (it runs when `PROCESSWORLD` is called).

23. PROCESSES

The Interlisp-D Process mechanism provides an environment in which multiple Lisp processes can run in parallel. Each executes in its own stack space, but all share a global address space. The current process implementation is cooperative; i.e., process switches happen voluntarily, either when the process in control has nothing to do or when it is in a convenient place to pause. There is no preemption or guaranteed service, so you cannot run something demanding (e.g., Chat) at the same time as something that runs for long periods without yielding control. Keyboard input and network operations block with great frequency, so processes currently work best for highly interactive tasks (editing, making remote files).

In Interlisp-D, the process mechanism is already turned on, and is expected to stay on during normal operations, as some system facilities (in particular, most network operations) require it. However, under exceptional conditions, the following function can be used to turn the world off and on:

(PROCESSWORLD *FLG*) [Function]

Starts up the process world, or if *FLG* = OFF, kills all processes and turns it off. Normally does not return. The environment starts out with two processes: a top-level EVALQT (the initial "tty" process) and the "background" process, which runs the window mouse handler and other system background tasks.

PROCESSWORLD is intended to be called at the top level of Interlisp, not from within a program. It does not toggle some sort of switch; rather, it constructs some new processes in a new part of the stack, leaving any callers of PROCESSWORLD in a now inaccessible part of the stack. Calling (PROCESSWORLD 'OFF) is the only way the call to PROCESSWORLD ever returns.

(HARDRESET) [Function]

Resets the whole world, and rebuilds the stack from scratch. This is "harder" than doing RESET to every process, because it also resets system internal processes (such as the keyboard handler).

HARDRESET automatically turns the process world on (or resets it if it was on), unless the variable AUTOPROCESSFLG is NIL.

Creating and Destroying Processes

(ADD.PROCESS *FORM PROP VALUE ... PROP VALUE*) [NoSpread Function]

Creates a new process evaluating *FORM*, and returns its process handle. The process's stack environment is the top level, i.e., the new process does not have access to the environment in which ADD.PROCESS was called; all such information must be passed as arguments in *FORM*. The process runs until *FORM* returns or the process is explicitly deleted. An untrapped error within the process also deletes the process (unless its RESTARTABLE property is T), in which case a message is printed to that effect.

The remaining arguments are alternately property names and values. Any property/value pairs acceptable to PROCESSPROP may be given, but the following two are directly relevant to ADD.PROCESS:

NAME	Value should be a litatom; if not given, the process name is taken from (CAR <i>FORM</i>). ADD.PROCESS may pack the name with a number to make it unique. This name is solely for the convenience of manipulating processes at Lisp typein; e.g., the name can be given as the <i>PROC</i> argument to most process functions, and the name appears in menus of processes. However, programs should normally only deal in process handles, both for efficiency and to avoid the confusion that can result if two processes have the same defining form.
SUSPEND	If the value is non-NIL, the new process is created but then immediately suspended; i.e., the process does not actually run until woken by a WAKE.PROCESS (below).

(PROCESSPROP *PROC PROP NEWVALUE*) [NoSpread Function]

Used to get or set the values of certain properties of process *PROC*, in a manner analogous to WINDOWPROP. If *NEWVALUE* is supplied (including if it is NIL), property *PROP* is given that value. In all cases, returns the old value of the property. The following properties have special meaning for processes; all others are uninterpreted:

NAME	Value is a litatom used for identifying the process to the user.
FORM	Value is the Lisp form used to start the process (readonly).
RESTARTABLE	Value is a flag indicating the disposition of the process following errors or hard resets:

NIL or NO (the default): If an untrapped error (or Control-E or Control-D) causes its form to be exited, the process is deleted. The process is also deleted if a HARDRESET (or Control-D from RAID) occurs, causing the entire Process world to be reinitialized.

T or YES: The process is automatically restarted on errors or HARDRESET. This is the normal setting for persistent "background" processes, such as the mouse process, that can safely restart themselves on errors.

HARDRESET: The process is deleted as usual if an error causes its form to be exited, but it is restarted on a HARDRESET. This setting is preferred for persistent processes for which an error is an unusual condition, one that might repeat itself if the process were simply blindly restarted.

RESTARTFORM	If the value is non-NIL, it is the form used if the process is restarted (instead of the value of the FORM property). Of course, the process must also have a non-NIL RESTARTABLE prop for this to have any effect.
BEFOREEXIT	If the value is the atom DON'T, it will not be interrupted by a LOGOUT. If LOGOUT is attempted before the process finishes, a message will appear saying that Interlisp is waiting for the process to finish. If you want the LOGOUT to proceed without waiting, you must use the process status window (from the background menu) to delete the process.
AFTEREXIT	Value indicates the disposition of the process following a resumption of Lisp after some exit (LOGOUT, SYSOUT, MAKESYS). Possible values are: DELETE: Delete the process. SUSPEND: Suspend the process; i.e., do not let it run until it is explicitly woken. An event: Cause the process to be suspended waiting for the event (See the Events section below).
INFOHOOK	Value is a function or form used to provide information about the process, in conjunction with the INFO command in the process status window (see the Process Status Window section below).

WINDOW	Value is a window associated with the process, the process's "main" window. Used to switch the tty process to this process when you click in this window (see the Switching the TTY Process section below). Setting the WINDOW property does not set the primary I/O stream (NIL) or the terminal I/O stream (T) to the window. When a process is created, I/O operations to the NIL or T stream will cause a new window to appear. TTYDISPLAYSTREAM (see Chapter 28) should be used to set the terminal i/o stream of a process to a specific window.
TTYENTRYFN	Value is a function that is applied to the process when the process is made the tty process (see the Switching the TTY Process section below).
TTYEXITFN	Value is a function that is applied to the process when the process ceases to be the tty process (see the Switching the TTY Process section below).
(THIS.PROCESS)	[Function]
	Returns the handle of the currently running process, or NIL if the Process world is turned off.
(DEL.PROCESS PROC —)	[Function]
	Deletes process <i>PROC</i> . <i>PROC</i> may be a process handle (returned by ADD.PROCESS), or its name. If <i>PROC</i> is the currently running process, DEL.PROCESS does not return!
(PROCESS.RETURN VALUE)	[Function]
	Terminates the currently running process, causing it to "return" <i>VALUE</i> . There is an implicit PROCESS.RETURN around the <i>FORM</i> argument given to ADD.PROCESS, so that normally a process can finish by simply returning; PROCESS.RETURN is supplied for earlier termination.
(PROCESS.RESULT PROCESS WAITFORRESULT)	[Function]
	If <i>PROCESS</i> has terminated, returns the value, if any, that it returned. This is either the value of a PROCESS.RETURN or the value returned from the form given to ADD.PROCESS. If the process was aborted, the value is NIL. If <i>WAITFORRESULT</i> is true, PROCESS.RESULT blocks until <i>PROCESS</i> finishes, if necessary; otherwise, it returns NIL immediately if <i>PROCESS</i> is still running. <i>PROCESS</i> must be the actual process handle returned from

ADD.PROCESS, not a process name, as the association between handle and name disappears when the process finishes (and the process handle itself is then garbage collected if no one else has a pointer to it).

(PROCESS.FINISHEDP *PROCESS*) [Function]

True if *PROCESS* has terminated. The value returned is an indication of how it finished: NORMAL or ERROR.

(PROCESSP *PROC*) [Function]

True if *PROC* is the handle of an active process, i.e., one that has not yet finished.

(RELPROCESSP *PROCHANDLE*) [Function]

True if *PROCHANDLE* is the handle of a deleted process. This is analogous to RELSTKP. It differs from PROCESS.FINISHEDP in that it never causes an error, while PROCESS.FINISHEDP can cause an error if its *PROC* argument is not a process at all.

(RESTART.PROCESS *PROC*) [Function]

Unwinds *PROC* to its top level and reevaluates its form. This is effectively a DEL.PROCESS followed by the original ADD.PROCESS.

(MAP.PROCESSES *MAPFN*) [Function]

Maps over all processes, calling *MAPFN* with three arguments: the process handle, its name, and its form.

(FIND.PROCESS *PROC ERRORFLG*) [Function]

If *PROC* is a process handle or the name of a process, returns the process handle for it, else NIL. If *ERRORFLG* is T, generates an error if *PROC* is not, and does not name, a live process.

Process Control Constructs

(BLOCK *MSECSWAIT TIMER*) [Function]

Yields control to the next waiting process, assuming any is ready to run. If *MSECSWAIT* is specified, it is a number of milliseconds to wait before returning, or T, meaning wait forever (until explicitly woken). Alternatively, *TIMER* can be given as a millisecond timer (as returned by SETUPTIMER, Chapter 12) of an absolute time at which to wake up. In any of those cases, the

process enters the *waiting* state until the time limit is up. `BLOCK` with no arguments leaves the process in the *runnable* state, i.e., it returns as soon as every other runnable process of the same priority has had a chance.

`BLOCK` can be aborted by interrupts such as Control-D, Control-E, or Control-B. `BLOCK` will return before its timeout is completed, if the process is woken by `WAKE.PROCESS`, `PROCESS.EVAL`, or `PROCESS.APPLY`.

(`DISMISS MSECWAIT TIMER NOBLOCK`) [Function]

`DISMISS` is used to dismiss the current process for a given period of time. Similar to `BLOCK`, except that:

- `DISMISS` is guaranteed not to return until the specified time has elapsed
- `MSECWAIT` cannot be `T` to wait forever
- If `NOBLOCK` is `T`, `DISMISS` will not allow other processes to run, but will busy-wait until the amount of time given has elapsed.

(`WAKE.PROCESS PROC STATUS`) [Function]

Explicitly wakes process `PROC`, i.e., makes it *runnable*, and causes its call to `BLOCK` (or other waiting function) to return `STATUS`. This is one simple way to notify a process of some happening; however, note that if `WAKE.PROCESS` is applied to a process more than once before the process actually gets its turn to run, it sees only the latest `STATUS`.

(`SUSPEND.PROCESS PROC`) [Function]

Blocks process `PROC` indefinitely, i.e., `PROC` will not run until it is woken by a `WAKE.PROCESS`.

The following three functions allow access to the stack context of some other process. They require a little bit of care, and are computationally non-trivial, but they do provide a more powerful way of manipulating another process than `WAKE.PROCESS` allows.

(`PROCESS.EVALV PROC VAR`) [Function]

Performs (`EVALV VAR`) in the stack context of `PROC`.

(`PROCESS.EVAL PROC FORM WAITFORRESULT`) [Function]

Evaluates `FORM` in the stack context of `PROC`. If `WAITFORRESULT` is true, blocks until the evaluation returns a result, else allows the current process to run in parallel with the evaluation. Any errors that occur will be in the context of `PROC`, so be careful. In particular, note that

(`PROCESS.EVAL PROC ' (NLSETQ (FOO))`)

and

```
(NLSETQ (PROCESS.EVAL PROC ' (FOO) ) )
```

behave quite differently if *FOO* causes an error. And it is quite permissible to intentionally cause an error in *proc* by performing

```
(PROCESS.EVAL PROC ' (ERROR! ) )
```

If errors are possible and *WAITFORRESULT* is true, the caller should almost certainly make sure that *FORM* traps the errors; otherwise the caller could end up waiting forever if *FORM* unwinds back into the pre-existing stack context of *PROC*.

After *FORM* is evaluated in *PROC*, the process *PROC* is woken up, even if it was running *BLOCK* or *AWAIT.EVENT*. This is necessary because an event of interest may have occurred while the process was evaluating *FORM*.

```
(PROCESS.APPLY PROC FN ARGS WAITFORRESULT) [Function]
```

Performs *(APPLY FN ARGS)* in the stack context of *PROC*. Note the same warnings as with *PROCESS.EVAL*.

Events

An "event" is a synchronizing primitive used to coordinate related processes, typically producers and consumers. Consumer processes can "wait" on events, and producers "notify" events.

```
(CREATE.EVENT NAME) [Function]
```

Returns an instance of the *EVENT* datatype, to be used as the event argument to functions listed below. *NAME* is arbitrary, and is used for debugging or status information.

```
(AWAIT.EVENT EVENT TIMEOUT TIMERP) [Function]
```

Suspends the current process until *EVENT* is notified, or until a timeout occurs. If *TIMEOUT* is *NIL*, there is no timeout. Otherwise, timeout is either a number of milliseconds to wait, or, if *TIMERP* is *T*, a millisecond timer set to expire at the desired time using *SETUPTIMER* (see Chapter 12).

```
(NOTIFY.EVENT EVENT ONCEONLY) [Function]
```

If there are processes waiting for *EVENT* to occur, causes those processes to be placed in the running state, with *EVENT* returned as the value from *AWAIT.EVENT*. If *ONCEONLY* is true, only runs the first process waiting for

the event (this should only be done if the programmer knows that there can only be one process capable of responding to the event at once).

The meaning of an event is up to the programmer. In general, however, the notification of an event is merely a hint that something of interest to the waiting process has happened; the process should still verify that the conceptual event actually occurred. That is, *the process should be written so that it operates correctly even if woken up before the timeout and in the absence of the notified event*. In particular, the completion of `PROCESS.EVAL` and related operations in effect wakes up the process in which they were performed, since there is no secure way of knowing whether the event of interest occurred while the process was busy performing the `PROCESS.EVAL`.

There is currently one class of system-defined events, used with the network code. Each Pup and NS socket has associated with it an event that is notified when a packet arrives on the socket; the event can be obtained by calling `PUPSOCKETEVENT` or `NSOCKETEVENT`, respectively (see Chapter 32).

Monitors

It is often the case that cooperating processes perform operations on shared structures, and some mechanism is needed to prevent more than one process from altering the structure at the same time. Some languages have a construct called a monitor, a collection of functions that access a common structure with mutual exclusion provided and enforced by the compiler via the use of monitor locks. Interlisp-D has taken this implementation notion as the basis for a mutual exclusion capability suitable for a dynamically-scoped environment.

A monitorlock is an object created by you and associated with (e.g., stored in) some shared structure that is to be protected from simultaneous access. To access the structure, a program waits for the lock to be free, then takes ownership of the lock, accesses the structure, then releases the lock. The functions and macros below are used:

`(CREATE.MONITORLOCK NAME —)` [Function]

Returns an instance of the `MONITORLOCK` datatype, to be used as the lock argument to functions listed below. *NAME* is arbitrary, and is used for debugging or status information.

`(WITH.MONITOR LOCK FORM ... FORM )` [Macro]

Evaluates *FORM ... FORM* while owning *LOCK*, and returns the value of *FORM* . This construct is implemented so that the lock is released even if the form is exited via error (currently implemented with `RESETLST`).

Ownership of a lock is dynamically scoped: if the current process already owns the lock (e.g., if the caller was itself inside a `WITH.MONITOR` for this lock), `WITH.MONITOR` does not wait for the lock to be free before evaluating *FORM ... FORM* .

(`WITH.FAST.MONITOR LOCK FORM ... FORM`) [Macro]

Like `WITH.MONITOR`, but implemented without the `RESETLST`. User interrupts (e.g., Control-E) are inhibited during the evaluation of *FORM ... FORM* .

Programming restriction: the evaluation of *FORM ... FORM* must not error (the lock would not be released). This construct is mainly useful when the forms perform a small, safe computation that never errors and need never be interrupted.

(`MONITOR.AWAIT.EVENT RELEASELOCK EVENT TIMEOUT TIMERP`) [Function]

For use in blocking inside a monitor. Performs (`AWAIT.EVENT EVENT TIMEOUT TIMERP`), but releases `RELEASELOCK` first, and reobtains the lock (possibly waiting) on wakeup.

Typical use for `MONITOR.AWAIT.EVENT`: A function wants to perform some operation on *FOO*, but only if it is in a certain state. It has to obtain the lock on the structure to make sure that the state of the structure does not change between the time it tests the state and performs the operation. If the state turns out to be bad, it then waits for some other process to make the state good, meanwhile releasing the lock so that the other process can alter the structure.

```
(WITH.MONITOR FOO-LOCK
  (until CONDITION-OF-FOO
    do (MONITOR.AWAIT.EVENT FOO-LOCK EVENT-FOO-
      CHANGED TIMEOUT) )
  OPERATE-ON-FOO)
```

It is sometimes convenient for a process to have `WITH.MONITOR` at its top level and then do all its interesting waiting using `MONITOR.AWAIT.EVENT`. Not only is this often cleaner, but in the present implementation in cases where the lock is frequently accessed, it saves the `RESETLST` overhead of `WITH.MONITOR`.

Programming restriction: There must not be an `ERRORSET` between the enclosing `WITH.MONITOR` and the call to `MONITOR.AWAIT.EVENT` such that the `ERRORSET` would catch an `ERROR!` and continue inside the monitor, for the lock would not have been reobtained. (The reason for this restriction is that, although `MONITOR.AWAIT.EVENT` won't itself error, you could have caused an error with an interrupt, or a `PROCESS.EVAL` in the context of the waiting process that produced an error.)

On rare occasions it may be useful to manipulate monitor locks directly. The following two functions are used in the implementation of `WITH.MONITOR`:

(OBTAIN.MONITORLOCK *LOCK DONTWAIT UNWINDSAVE*) [Function]

Takes possession of *LOCK*, waiting if necessary until it is free, unless *DONTWAIT* is true, in which case it returns *NIL* immediately. If *UNWINDSAVE* is true, performs a *RESETSAVE* to be unwound when the enclosing *RESETLST* exits. Returns *LOCK* if *LOCK* was successfully obtained, *T* if the current process already owned *LOCK*.

(RELEASE.MONITORLOCK *LOCK EVENIFNOTMINE*) [Function]

Releases *LOCK* if it is owned by the current process, and wakes up the next process, if any, waiting to obtain the lock.

If *EVENIFNOTMINE* is non-*NIL*, the lock is released even if it is not owned by the current process.

When a process is deleted, any locks it owns are released.

Global Resources

The biggest source of problems in the multi-processing environment is the matter of global resources. Two processes cannot both use the same global resource if there can be a process switch in the middle of their use (currently this means calls to *BLOCK*, but ultimately with a preemptive scheduler means anytime). Thus, user code should be wary of its own use of global variables, if it ever makes sense for the code to be run in more than one process at a time. "State" variables private to a process should generally be bound in that process; structures that are shared among processes (or resources used privately but expensive to duplicate per process) should be protected with monitor locks or some other form of synchronization.

Aside from user code, however, there are many *system* global variables and resources. Most of these arise historically from the single-process Interlisp-10 environment, and will eventually be changed in Interlisp-D to behave appropriately in a multi-processing environment. Some have already been changed, and are described below. Two other resources not generally thought of as global variables—the keyboard and the mouse—are particularly idiosyncratic, and are discussed in the next section.

The following resources, which are global in Interlisp-10, are allocated per process in Interlisp-D: primary input and output (the streams affected by **INPUT** and **OUTPUT**), terminal input and output (the streams designated by the name **T**), the primary read table and primary terminal table, and dribble files. Thus, each process can print to its own primary output, print to the terminal, read from a different primary input, all without interfering with another process's reading and printing.

Each process begins life with its primary and terminal input/output streams set to a dummy stream. If the process attempts input or output using any of those dummy streams, e.g., by calling (READ T), or (PRINT & T), a tty window is automatically created for the process, and that window becomes the primary input/output and terminal input/output for the process. The default tty window is created at or near the region specified in the variable DEFAULTTTYREGION.

A process can, of course, call TTYDISPLAYSTREAM explicitly to give itself a tty window of its own choosing, in which case the automatic mechanism never comes into play. Calling TTYDISPLAYSTREAM when a process has no tty window not only sets the terminal streams, but also sets the primary input and output streams to be that window, assuming they were still set to the dummy streams.

(HASTTYWINDOWP *PROCESS*)

[Function]

Returns T if the process *PROCESS* has a tty window; NIL otherwise. If *PROCESS* is NIL, it defaults to the current process.

Other system resources that are typically changed by RESETFORM, RESETLST, or RESETVARS are all global entities. In the multiprocessing environment, these constructs are suspect, as there is no provision for "undoing" them when a process switch occurs. For example, in the current release of Interlisp-D, it is not possible to set the print radix to 8 inside only one process, as the print radix is a global entity.

Note that RESETFORM and similar expressions are perfectly valid in the process world, and even quite useful, when they manipulate things strictly within one process. The process world is arranged so that deleting a process also unwinds any RESETxxx expressions that were performed in the process and are still waiting to be unwound, exactly as if a Control-D had reset the process to the top. Additionally, there is an implicit RESETLST at the top of each process, so that RESETSAVE can be used as a way of providing "cleanup" functions for when a process is deleted. For these, the value of RESETSTATE (see Chapter 14) is NIL if the process finished normally, ERROR if it was aborted by an error, RESET if the process was explicitly deleted, and HARDRESET if the process is being restarted after a HARDRESET or a RESTART . PROCESS.

Typein and the TTY Process

There is one global resource, the keyboard, that is particularly problematic to share among processes. Consider, for example, having two processes both performing (READ T). Since the keyboard input routines block while there is no input, both processes would spend most of their time blocking, and it would simply be a matter of chance which process received each character of typein.

To resolve such dilemmas, the system designates a distinguished process, termed the *tty process*, that is assumed to be the process that is involved in terminal interaction. Any typein from the keyboard goes to that process. If a process other than the tty process requests keyboard input, it blocks until it becomes the tty process. When the tty process is switched (in any of the ways described further below), any typeahead that occurred before the switch is saved and associated with the current tty process. Thus, it is always the case that keystrokes are sent to the process that is the tty process at the time of the keystrokes, regardless of when that process actually gets around to reading them.

It is less immediately obvious how to handle keyboard interrupt characters, as their action is asynchronous and not always tied to typein. Interrupt handling is described in the Handling of Interrupts section below.

Switching the TTY Process

Any process can make itself be the tty process by calling `TTY.PROCESS`.

`(TTY.PROCESS PROC)` [Function]

Returns the handle of the current tty process. In addition, if *PROC* is non-NIL, makes it be the tty process. The special case of *PROC* = T is interpreted to mean the executive process; this is sometimes useful when a process wants to explicitly give up being the tty process.

`(TTY.PROCESSP PROC)` [Function]

True if *PROC* is the tty process; *PROC* defaults to the running process. Thus, `(TTY.PROCESSP)` is true if the caller is the tty process.

`(WAIT.FOR.TTY MSECS NEEDWINDOW)` [Function]

Efficiently waits until `(TTY.PROCESSP)` is true. `WAIT.FOR.TTY` is called internally by the system functions that read from the terminal; user code thus need only call it in special cases.

If *MSECS* is non-NIL, it is the number of milliseconds to wait before timing out. If `WAIT.FOR.TTY` times out before `(TTY.PROCESSP)` is true, it returns NIL, otherwise it returns T. If *MSECS* is NIL, `WAIT.FOR.TTY` will not time out.

If *NEEDWINDOW* is non-NIL, `WAIT.FOR.TTY` opens a TTY window for the current process if one isn't already open.

`WAIT.FOR.TTY` spawns a new mouse process if called under the mouse process (see `SPAWN.MOUSE`, in the Keeping the Mouse Alive section below).

In some cases, such as in functions invoked as a result of mouse action or a user's typed-in call, it is reasonable for the function to invoke `TTY.PROCESS` itself so that it can take subsequent user type in. In other cases, however, this is too undisciplined; it is desirable to let the user designate which process typein should be directed to. This is most conveniently done by mouse action.

The system supports the model that "to type to a process, you click in its window." To cooperate with this model, any process desiring keyboard input should put its process handle as the `PROCESS` property of its window(s). To handle the common case, the function `TTYDISPLAYSTREAM` does this automatically when the `ttydisplaystream` is switched to a new window. A process can own any number of windows; clicking in any of those windows gives the process the `tty`.

This mechanism suffices for most casual process writers. For example, if a process wants all its input/output interaction to occur in a particular window that it has created, it should just make that window be its `tty` window by calling `TTYDISPLAYSTREAM`. Thereafter, it can `PRINT` or `READ` to/from the `T` stream; if the process is not the `tty` process at the time that it calls `READ`, it will block until the user clicks in the window.

For those needing tighter control over the `tty`, the default behavior can be overridden or supplemented. The remainder of this section describes the mechanisms involved.

There is a window property `WINDOWENTRYFN` that controls whether and how to switch the `tty` to the process owning a window. The mouse handler, before invoking any normal `BUTTONEVENTFN`, specifically notices the case of a button going down in a window that belongs to a process (i.e., has a `PROCESS` window property) that is not the `tty` process. In this case, it invokes the window's `WINDOWENTRYFN` of one argument (`WINDOW`). `WINDOWENTRYFN` defaults to `GIVE.TTY.PROCESS`:

(`GIVE.TTY.PROCESS WINDOW`) [Function]

If `WINDOW` has a `PROCESS` property, performs (`TTY.PROCESS (WINDOWPROP WINDOW 'PROCESS)`) and then invokes `WINDOW`'s `BUTTONEVENTFN` function (or `RIGHTBUTTONFN` if the right button is down).

There are some cases where clicking in a window does not always imply that the user wants to talk to that window. For example, clicking in a text editor window with a shift key held down means to "shift-select" some piece of text into the input buffer of the *current* `tty` process. The editor supports this by supplying a `WINDOWENTRYFN` that performs `GIVE.TTY.PROCESS` if no shift key is down, but goes into its shift-select mode, without changing the `tty` process, if a shift key is down. The shift-select mode performs a `BKSYSEBUF` of the selected text when the shift key is let up, the `BKSYSEBUF` feeding input to the current `tty` process.

Sometimes a process wants to be notified when it becomes the tty process, or stops being the tty process. To support this, there are two process properties, `TTYEXITFN` and `TTYENTRYFN`. The actions taken by `TTY.PROCESS` when it switches the tty to a new process are as follows: the former tty process's `TTYEXITFN` is called with two arguments (`OLDTTYPROCESS NEWTTYPROCESS`); the new process is made the tty process; finally, the new tty process's `TTYENTRYFN` is called with two arguments (`NEWTTYPROCESS OLDTTYPROCESS`). Normally the `TTYENTRYFN` and `TTYEXITFN` need only their first argument, but the other process involved in the switch is supplied for completeness. In the present system, most processes want to interpret the keyboard in the same way, so it is considered the responsibility of any process that changes the keyboard interpretation to restore it to the normal state by its `TTYEXITFN`.

A window is "owned" by the last process that anyone gave as the window's `PROCESS` property. Ordinarily there is no conflict here, as processes tend to own disjoint sets of windows (though, of course, cooperating processes can certainly try to confuse each other). The only likely problem arises with that most global of windows, `PROMPTWINDOW`. Programs should not be tempted to read from `PROMPTWINDOW`. This is not usually necessary anyway, as the first attempt to read from `T` in a process that has not set its `TTYDISPLAYSTREAM` to its own window causes a tty window to be created for the process (see the Global Resources section above).

Handling of Interrupts

At the time that a keyboard interrupt character (see Chapter 30) is struck, any process could be running, and some decision must be made as to which process to actually interrupt. To the extent that keyboard interrupts are related to typein, most interrupts are taken in the tty process; however, the following are handled specially:

`RESET` (initially Control-D)

`ERROR` (initially Control-E)

These interrupts are taken in the mouse process, if the mouse is not in its idle state; otherwise they are taken in the tty process. Thus, Control-E can be used to abort some mouse-invoked window action, such as the Shape command. As a consequence, note that if the mouse invokes some lengthy computation that the user thinks of as "background", Control-E still aborts it, even though that may not have been what the user intended. Such lengthy computations, for various reasons, should generally be performed by spawning a separate process to perform them. The `RESET` interrupt in a process other than the executive is interpreted exactly as if an error unwound the process to its top level: if the process was

	designated <code>RESTARTABLE = T</code> , it is restarted; otherwise it is killed.
<code>HELP</code> (initially <code>Control-G</code>)	A menu of processes is presented to the user, who is asked to select which one the interrupt should occur in. The current tty process appears with a * next to its name at the top of the menu. The menu also includes an entry "[Spawn Mouse]", for the common case of needing a mouse because the mouse process is currently tied up running someone's <code>BUTTONEVENTFN</code> ; selecting this entry spawns a new mouse process, and no break occurs.
<code>BREAK</code> (initially <code>Control-B</code>)	Performs the <code>HELP</code> interrupt in the mouse process, if the mouse is not in its idle state; otherwise it is performed in the tty process.
<code>RUBOUT</code> (initially <code>DELETE</code>)	This interrupt clears typeahead in <i>all</i> processes.
<code>RAID</code> , <code>STACK OVERFLOW</code> <code>STORAGE FULL</code>	These interrupts always occur in whatever process was running at the time the interrupt struck. In the cases of <code>STACK OVERFLOW</code> and <code>STORAGE FULL</code> , this means that the interrupt is more likely to strike in the offending process (especially if it is a "runaway" process that is not blocking). Note, however, that this process is still not necessarily the guilty party; it could be an innocent bystander that just happened to use up the last of a resource prodigiously consumed by some other process.

Keeping the Mouse Alive

Since the window mouse handler runs in its own process, it is not available while a window's `BUTTONEVENTFN` function (or any of the other window functions invoked by mouse action) is running. This leads to two sorts of problems: (1) a long computation underneath a `BUTTONEVENTFN` deprives the user of the mouse for other purposes, and (2) code that runs as a `BUTTONEVENTFN` cannot rely on other `BUTTONEVENTFN`s running, which means that there some pieces of code that run differently from normal when run under the mouse process. These problems are addressed by the following functions:

(SPAWN.MOUSE —) [Function]

Spawns another mouse process, allowing the mouse to run even if it is currently "tied up" under the current mouse process. This function is intended mainly to be typed in at the Lisp executive when the user notices the mouse is busy.

(ALLOW.BUTTON.EVENTS) [Function]

Performs a (SPAWN.MOUSE) only when called underneath the mouse process. This should be called (once, on entry) by any function that relies on BUTTONEVENTFNs for completion, if there is any possibility that the function will itself be invoked by a mouse function.

It never hurts, at least logically, to call SPAWN.MOUSE or ALLOW.BUTTON.EVENTS needlessly, as the mouse process arranges to quietly kill itself if it returns from the user's BUTTONEVENTFN and finds that another mouse process has sprung up in the meantime. (There is, of course, some computational expense.)

Process Status Window

The background menu command PSW (see Chapter 28) and the function PROCESS.STATUS.WINDOW (below) create a "Process Status Window", that allows the user to examine and manipulate all of the existing processes:

SPACEWINDOW		
Tedit		
MOUSE		
ERIS#LEAF		
\10MBWATCHER		
EXEC		
\NSGATELISTENER		
\PUPGATELISTENER		
\TIMER.PROCESS		
BACKGROUND		
BT	WHO?	KILL
BTV	KBD←	RESTART
BTV*	INFO	WAKE
BTV!	BREAK	SUSPEND

The window consists of two menus. The top menu lists all the processes at the moment. Commands in the bottom menu operate on the process selected in the top menu (EXEC in the example above). The commands are:

BT, BTV, BTV*, BTV!	Displays a backtrace of the selected process.
WHO?	Changes the selection to the tty process, i.e., the one currently in control of the keyboard.
KBD←	Associates the keyboard with the selected process; i.e., makes the selected process be the tty process.
INFO	If the selected process has an INFOHOOK property, calls it. The hook may be a function, which is then applied to two arguments, the process and the button (LEFT or MIDDLE) used to invoke INFO, or a form, which is simply EVAL'ed. The APPLY or EVAL happens in the context of the selected process, using PROCESS.APPLY or PROCESS.EVAL. The INFOHOOK process property can be set using PROCESSPROP (see the Creating and Destroying Processes section above).
BREAK	Enter a break under the selected process. This has the side effect of waking the process with the value returned from the break.
KILL	Deletes the selected process.
RESTART	Restarts the selected process.
WAKE	Wakes the selected process. Prompts for a value to wake it with (see WAKE.PROCESS).
SUSPEND	Suspends the selected process; i.e., causes it to block indefinitely (until explicitly woken).

(PROCESS.STATUS.WINDOW *WHERE*)

[Function]

Puts up a process status window that provides several debugging commands for manipulating running processes. If the window is already up, PROCESS.STATUS.WINDOW refreshes it. If *WHERE* is a position, the window is placed in that position; otherwise, the user is prompted for a position.

Currently, the process status window runs under the mouse process, like other menus, so if the mouse is unavailable (e.g., a mouse function is performing an extensive computation), you may be unable to use the process status window (you can try `SPAWN.MOUSE`, of course).

Non-Process Compatibility

This section describes some considerations for authors of programs that ran in the old single-process Interlisp-D environment, and now want to make sure they run properly in the Multi-processing world. The biggest problem to watch out for is code that runs underneath the mouse handler. Writers of mouse handler functions should remember that in the process world the mouse handler runs in its own process, and hence (a) you cannot depend on finding information on the stack (stash it in the window instead), and (b) while your function is running, the mouse is not available (if you have any non-trivial computation to do, spawn a process to do it, notify one of your existing processes to do it, or use `PROCESS.EVAL` to run it under some other process).

The following functions are meaningful even if the process world is not on: `BLOCK` (invokes the system background routine, which includes handling the mouse); `TTY.PROCESS`, `THIS.PROCESS` (both return `NIL`); and `TTY.PROCESSP` (returns `T`, i.e., anyone is allowed to take tty input). In addition, the following two functions exist in both worlds:

`(EVAL.AS.PROCESS FORM)` [Function]

Same as `(ADD.PROCESS FORM 'RESTARTABLE 'NO)`, when processes are running, `EVAL` when not. This is highly recommended for mouse functions that perform any non-trivial activity.

`(EVAL.IN.TTY.PROCESS FORM WAITFORRESULT)` [Function]

Same as `(PROCESS.EVAL (TTY.PROCESS) FORM WAITFORRESULT)`, when processes are running, `EVAL` when not.

Most of the process functions that do not take a process argument can be called even if processes aren't running. `ADD.PROCESS` creates, but does not run, a new process (it runs when `PROCESSWORLD` is called).

23. STREAMS AND FILES

A stream is an object that provides an interface to a physical or logical device. The stream object contains local data and methods that operate on the stream object. Medley's general-purpose I/O functions take a stream as one of their arguments. Not every device is capable of implementing every I/O operation, while some devices offer special functions for that device alone. Such restrictions and extensions are noted in the documentation of each device. The majority of the streams used in Medley fall into two categories: file streams and image streams.

A file is a sequence of data stored on some device that allows the data to be retrieved at a later time. Files are identified by a name specifying their storage devices. Input or output to a file is performed through a stream to the file, using `OPENSTREAM` (below). In addition, there are functions that manipulate the files themselves, rather than their data content.

An image stream is an output stream to a display device, such as the display screen or a printer. In addition to the standard output operations, an image stream implements a variety of graphics operations, such as drawing lines and displaying characters in multiple fonts. Unlike a file, the "content" of an image stream cannot be retrieved. Image streams are described in Chapter 26.

This chapter describes operations specific to file devices: how to name files, how to open streams to files, and how to manipulate files on their devices.

Opening and Closing File Streams

To perform input from or output to a file, you must create a stream to the file, using `OPENSTREAM`:

(**OPENSTREAM** *FILE ACCESS RECOG PARAMETERS* →) [Function]

Opens and returns a stream for the file specified by *FILE*, a file name. *FILE* can be either a string or a symbol. The syntax and manipulation of file names is described at length in the `FILENAMES` section below. Incomplete file names are interpreted with respect to the connected directory (below).

RECOG specifies the recognition mode of *FILE* (below). If *RECOG* = `NIL`, it defaults according to the value of *ACCESS*.

ACCESS specifies the "access rights" to be used when opening the file. Possible values are:

- INPUT** Only input operations are permitted on the already existing file. Starts reading at the beginning of the file. *RECOG* defaults to `OLD`.
- OUTPUT** Only output operations are permitted on the initially empty file. Starts writing at the beginning of the file. While the file is open, other users or processes are unable to open the file for either input or output. *RECOG* defaults to `NEW`.
- BOTH** Both input and output operations are permitted on the file. Starts reading or writing at the beginning of the file. *RECOG* defaults to

INTERLISP-D REFERENCE MANUAL

OLD/NEW. *ACCESS* = BOTH implies random access (Chapter 25), and may not be possible for files on some devices.

APPEND Only sequential output operations are permitted on the file. Starts writing at the end of the file. *RECOG* defaults to OLD/NEW. *ACCESS* = APPEND may not be allowed for files on some devices.

Note: *ACCESS* = OUTPUT implies that you intend to write a new or different file, even if a version number was specified and the corresponding file already exists. Any previous contents of the file are discarded, and the file is empty immediately after the OPENSTREAM. If you want to write on an already existing file while preserving the old contents, the file must be opened for access BOTH or APPEND.

PARAMETERS is a list of pairs (*ATTRIB VALUE*), where *ATTRIB* is a file attribute (see SETFILEINFO below). A non-list *ATTRIB* in *PARAMETERS* is treated as the pair (*ATTRIB T*). Generally speaking, attributes that belong to the permanent file (e.g., *TYPE*) can only be set when creating a new file, while attributes that belong only to a particular opening of a file (e.g., *ENDOFSTREAMOP*) can be set on any call to OPENSTREAM. Not all devices honor all attributes; those not recognized by a particular device are simply ignored.

In addition to the attributes permitted by SETFILEINFO, the following attributes are accepted by OPENSTREAM as values of *ATTRIB* in its *PARAMETERS* argument:

DON'T.CHANGE.DATE If *VALUE* is non-NIL, the file's creation date is not changed when the file is opened. This option is meaningful only for old files opened for BOTH access. You should use this only for specialized applications where the caller does not want the file system to believe the file's content has been changed.

SEQUENTIAL If *VALUE* is non-NIL, this opening of the file need support only sequential access; i.e., the caller intends never to use SETFILEPTR. For some devices, sequential access to files is much more efficient than random access. Note that the device may choose to ignore this attribute and still open the file in a manner that permits random access. Also note that this attribute does not make sense with *ACCESS* = BOTH.

If *FILE* is not recognized by the file system, OPENSTREAM causes the error FILE NOT FOUND. Ordinarily, this error is intercepted via an entry on ERRORTYPELIST (Chapter 24), which causes SPELLFILE (see the Searching File Directories below) to be called. SPELLFILE searches alternate directories and possibly attempts spelling correction on the file name. Only if SPELLFILE is unsuccessful will the FILE NOT FOUND error actually occur.

If *FILE* exists but cannot be opened, OPENSTREAM causes one of several other errors: FILE WON'T OPEN if the file is already opened for conflicting access by someone else; PROTECTION VIOLATION if the file is protected against the operation; FILE SYSTEM RESOURCES EXCEEDED if there is no more room in the file system.

STREAMS & FILES

(**CLOSEF** *FILE*)

[Function]

Closes *FILE* and returns its full file name. Generates an error, `FILE NOT OPEN`, if *FILE* does not designate an open stream. After closing a stream, no further input/output operations are permitted on it.

If *FILE* is `NIL`, it is defaulted to the primary input stream if that is not the terminal stream, or else the primary output stream if that is not the terminal stream. If both primary input and output streams are the terminal input/output streams, `CLOSEF` returns `NIL`. If `CLOSEF` closes either the primary input stream or the primary output stream (either explicitly or in the *FILE* = `NIL` case), it resets the primary stream for that direction to be the corresponding terminal stream. See Chapter 25 for information on the primary input/output streams.

`WHENCLOSE` (below) allows you to "advise" `CLOSEF` to perform various operations when a file is closed.

Because of buffering, the contents of a file open for output are not guaranteed to be written to the actual physical file device until `CLOSEF` is called. Buffered data can be forced out to a file without closing the file by using the function `FORCEOUTPUT` (Chapter 25).

Some network file devices perform their transactions in the background. As a result, it is possible for a file to be closed by `CLOSEF` and yet not be "fully" closed for a small time period afterward. During this time the file appears to be busy and cannot be opened for conflicting access by others.

(**CLOSEF?** *FILE*)

[Function]

Closes *FILE* if it is open, returning the value of `CLOSEF`; otherwise does nothing and returns `NIL`.

In the present implementation of Medley, all open streams to files are kept in a registry of "open files". This registry does not include nameless streams, such as string streams (below), display streams (Chapter 28), and the terminal input and output streams; nor streams explicitly hidden from you, such as dribble streams (Chapter 30). This registry may not persist in future implementations of Medley, but at the present time it is accessible by the following two functions:

(**OPENP** *FILE ACCESS*)

[Function]

ACCESS is an access mode for a stream opening (see `OPENSTREAM`), or `NIL` for any access.

If *FILE* is a stream, returns its full name if it is open for the specified access, otherwise `NIL`.

If *FILE* is a file name (a symbol), *FILE* is processed according to the rules of file recognition (below). If a stream open to a file by that name is registered and open for the specified access, then the file's full name is returned. If the file name is not recognized, or no stream is open to the file with the specified access, `NIL` is returned.

If *FILE* is `NIL`, returns a list of the full names of all registered streams that are open for the specified access.

INTERLISP-D REFERENCE MANUAL

(**CLOSEALL** *ALLFLG*)

[Function]

Closes all streams in the value of (OPENP). Returns a list of the files closed.

WHENCLOSE (below) allows certain files to be "protected" from CLOSEALL. If *ALLFLG* is T, all files, including those protected by WHENCLOSE, are closed.

File Names

A file name in Medley is a string or symbol whose characters specify a "path" to the actual file: on what host or device the file resides, in which directory, and so forth. Because Medley supports a variety of non-local file devices, parts of the path could be device-dependent. However, it is desirable for programs to be able to manipulate file names in a device-independent manner. To this end, Medley specifies a uniform file name syntax over all devices; the functions that perform the actual file manipulation for a particular device are responsible for any translation to that device's naming conventions.

A file name is composed of a collection of *fields*, some of which have specific meanings. The functions described below refer to each field by a *field name*, a literal atom from among the following: HOST, DEVICE, DIRECTORY, NAME, EXTENSION, and VERSION. The standard syntax for a file name is {HOST}DEVICE:<DIRECTORY>NAME.EXTENSION;VERSION. Some host's file systems do not use all of those fields in their file names.

HOST	Specifies the host whose file system contains the file. In the case of local file devices, the "host" is the name of the device, e.g., DSK or FLOPPY.
DEVICE	Specifies, for those hosts that divide their file system's name space among multiple physical devices, the device or logical structure on which the file resides. This should not be confused with Medley's abstract "file device", which denotes either a host or a local physical device and is specified by the HOST field.
DIRECTORY	Specifies the "directory" containing the file. A directory usually is a grouping of a possibly large set of loosely related files, e.g., the personal files of a particular user, or the files belonging to some project. The DIRECTORY field usually consists of a principal directory and zero or more subdirectories that together describe a path through a file system's hierarchy. Each subdirectory name is set off from the previous directory or subdirectory by the character ">"; e.g., "LISP>LIBRARY>NEW".
NAME	This field carries no specific meaning, but generally names a set of files thought of as being different renditions of the "same" abstract file.
EXTENSION	This field also carries no specific meaning, but generally distinguishes the form of files having the same name. Most files systems have some "conventional" extensions that denote something about the content of the file. For example, in Medley, the extension DCOM, LCOM or DFASL denotes files containing compiled function definitions.

STREAMS & FILES

VERSION A number used to distinguish the versions or "generations" of the files having a common name and extension. The version number is incremented each time a new file by the same name is created.

Most functions that take as input "a directory" accept either a directory name (the contents of the **DIRECTORY** field of a file name) or a "full" directory specification—a file name fragment consisting of only the fields **HOST**, **DEVICE**, and **DIRECTORY**. In particular, the "connected directory" (see below) consists, in general, of all three fields.

For convenience in dealing with certain operating systems, Medley also recognizes `[]` and `()` as host delimiters (synonymous with `{}`), and `/` as a director delimiter (synonymous with `<` at the beginning of a directory specification and `>` to terminate directory or subdirectory specification). For example, a file on a Unix file server **UNIX** with the name `/usr/foo/bar/stuff.tedit`, whose **DIRECTORY** field is thus `usr/foo/bar`, could be specified as `{UNIX}/usr/foo/bar/stuff.tedit`, or `(UNIX)<usr/foo/bar>stuff.tedit`, or several other variations. Note that when using `[]` or `()` as host delimiters, they usually must be escaped with the reader's escape character if the file name is expressed as a symbol rather than a string.

Different hosts have different requirements for valid characters in file names. In Medley, all characters are valid. However, in order to be able to parse a file name into its component fields, it is necessary that those characters that are conventionally used as file name delimiters be quoted when they appear inside of fields where there could be ambiguity. The file name quoting character is `" ' "` (single quote). Thus, the following characters must be quoted when not used as delimiters: `>`, `:`, `;`, `/`, and `'` itself. The character `.` (period) need only be quoted if it is to be considered a part of the **EXTENSION** field. The characters `}`, `]`, and `)` need only be quoted in a file name when the host field of the name is introduced by `{`, `[`, and `(`, respectively. The characters `{`, `[`, `(`, and `<` need only be quoted if they appear as the first character of a file name fragment, where they would otherwise be assumed to introduce the **HOST** or **DIRECTOR** `il`s
`lds`.

The following functions are the standard way to manipulate file names in Medley. Their operation is purely syntactic—they perform no file system operations themselves.

(UNPACKFILENAME.STRING *FILENAME*)

[Function]

Parses *FILENAME*, returning a list in property list format of alternating field names and field contents. The field contents are returned as strings. If it is a stream, its full name is used.

Only those fields actually present in *FILENAME* are returned. A field is considered present if its delimiting punctuation is present, even if the field itself is empty. Empty fields are denoted by `""` (the empty string).

Examples:

```
(UNPACKFILENAME.STRING "FOO.BAR") =>
  (NAME "FOO" EXTENSION "BAR")
(UNPACKFILENAME.STRING "FOO.;2") =>
  (NAME "FOO" EXTENSION "" VERSION "2")
(UNPACKFILENAME.STRING "FOO;") =>
  (NAME "FOO" VERSION "")
```

INTERLISP-D REFERENCE MANUAL

```
(UNPACKFILENAME.STRING
  "{ERIS}<LISP>CURRENT>IMTRAN.DCOM;21")
=> (HOST "ERIS" DIRECTORY "LISP>CURRENT"
    NAME "IMTRAN" EXTENSION "DCOM"
    VERSION "21")
```

(**UNPACKFILENAME** *FILE*) [Function]

Old version of UNPACKFILENAME.STRING that returns the field values as atoms, rather than as strings. UNPACKFILENAME.STRING is now considered the "correct" way of unpacking file names, because it does not lose information when the contents of a field are numeric. For example,

```
(UNPACKFILENAME 'STUFF.TXT) =>
(NAME STUFF EXTENSION TXT)
```

but

```
(UNPACKFILENAME 'STUFF.029) =>
(NAME STUFF EXTENSION 29)
```

Explicitly omitted fields are denoted by the atom NIL, rather than the empty string.

Note: Both UNPACKFILENAME and UNPACKFILENAME.STRING leave the trailing colon on the device field, so that the Tenex device NIL: can be distinguished from the absence of a device. Although UNPACKFILENAME.STRING is capable of making the distinction, it retains this behavior for backward compatibility. Thus,

```
(UNPACKFILENAME.STRING '{TOAST}DSK:FOO) =>
(HOST "TOAST" DEVICE "DSK:" NAME "FOO")
```

(**FILENAMEFIELD** *FILENAME FIELDNAME*) [Function]

Returns, as an atom, the contents of the *FIELDNAME* field of *FILENAME*. If *FILENAME* is a stream, its full name is used.

(**PACKFILENAME.STRING** *FIELD CONTENTS ... FIELD CONTENTS*) [NoSpread Function]

Takes a sequence of alternating field names and field contents (atoms or strings), and returns the corresponding file name, as a string.

If PACKFILENAME.STRING is given a single argument, it is interpreted as a list of alternating field names and field contents. Thus PACKFILENAME.STRING and UNPACKFILENAME.STRING operate as inverses.

If the same field name is given twice, the *first* occurrence is used.

The contents of the field name DIRECTORY may be either a directory name or a full directory specification as described above.

PACKFILENAME.STRING also accepts the "field name" BODY to mean that its contents should itself be unpacked and spliced into the argument list at that point. This feature, in conjunction with the rule that fields early in the argument list override later duplicates, is useful for altering existing file names. For example, to provide a default field, place BODY

STREAMS & FILES

first in the argument list, then the default fields. To override a field, place the new fields first and BODY last.

If the value of the BODY field is a stream, its full name is used.

Examples:

```
(PACKFILENAME.STRING 'DIRECTORY "LISP"
  'NAME "NET")
=> "<LISP>NET"

(PACKFILENAME.STRING 'NAME "NET"
  'DIRECTORY "{DSK}<LISPFILES>")
=> "{DSK}<LISPFILES>NET"

(PACKFILENAME.STRING 'DIRECTORY "{DSK}"
  'BODY "{TOAST}<FOO>BAR")
=> "{DSK}BAR"

(PACKFILENAME.STRING 'DIRECTORY "FRED"
  'BODY "{TOAST}<FOO>BAR")
=> "{TOAST}<FRED>BAR"

(PACKFILENAME.STRING 'BODY "{TOAST}<FOO>BAR"
  'DIRECTORY "FRED")
=> "{TOAST}<FOO>BAR"

(PACKFILENAME.STRING 'VERSION NIL
  'BODY "{TOAST}<FOO>BAR.DCOM;2")
=> "{TOAST}<FOO>BAR.DCOM"

(PACKFILENAME.STRING 'BODY "{TOAST}<FOO>BAR.DCOM"
  'VERSION 1)
=> "{TOAST}<FOO>BAR.DCOM;1"

(PACKFILENAME.STRING 'BODY "{TOAST}<FOO>BAR.DCOM;"
  'VERSION 1)
=> "{TOAST}<FOO>BAR.DCOM;"

(PACKFILENAME.STRING 'BODY "BAR.;1"
  'EXTENSION "DCOM")
=> "BAR.;1"

(PACKFILENAME.STRING 'BODY "BAR;1"
  'EXTENSION "DCOM")
=> "BAR.DCOM;1"
```

In the last two examples, note that in one case the extension is explicitly present in the body (as indicated by the preceding period), while in the other there is no indication of an extension, so the default is used.

(PACKFILENAME FIELD CONTENTS ... FIELD CONTENTS) [NoSpread Function]

The same as PACKFILENAME.STRING, except that it returns the file name as a symbol, instead of a string.

Incomplete File Names

In general, it is not necessary to pass a complete file name (one containing all the fields listed above) to functions that take a file name as an argument. Interlisp supplies suitable defaults for certain fields (below). Functions that return names of actual files, however, always return the full file name.

INTERLISP-D REFERENCE MANUAL

If the version field is omitted from a file name, Interlisp performs version recognition, as described below.

If the host, device and/or directory field are omitted from a file name, Interlisp uses the currently connected directory. You can change the currently connected directory by calling `CNDIR` (below) or using the programmer's assistant command `CONN`.

Defaults are added to the partially specified name "left to right" until a host, device or directory field is encountered. Thus, if the connected directory is `{TWENTY}PS:<FRED>`, then

```
BAR.DCOM means
{TWENTY}PS:<FRED>BAR.DCOM

<GRANOLA>BAR.DCOM means
{TWENTY}PS:<GRANOLA>BAR.DCOM

MTAO:<GRANOLA>BAR.DCOM means
{TWENTY}MTAO:<GRANOLA>BAR.DCOM

{THIRTY}<GRANOLA>BAR.DCOM means
{THIRTY}<GRANOLA>BAR.DCOM
```

In addition, if the partially specified name contains a subdirectory, but no principal directory, then the subdirectory is appended to the connected directory. For example,

```
ISO>BAR.DCOM means
{TWENTY}PS:<FRED>ISO>BAR.DCOM
```

Or, if the connected directory is the Unix directory `{UNIX}/usr/fred/`, then `iso/bar.dcom` means `{UNIX}/usr/fred/iso/bar.dcom`, but `/other/bar.dcom` means `{UNIX}/other/bar.dcom`.

(`CNDIR` *HOST/DIR*)

[Function]

Connects to the directory *HOST/DIR*, which can either be a directory name or a full directory specification including host and/or device. If the specification includes just a host, and the host supports directories, the directory is defaulted to the value of `(USERNAME)`; if the host is omitted, connection is made to another directory on the same host as before. If *HOST/DIR* is `NIL`, connects to the value of `LOGINHOST/DIR`.

`CNDIR` returns the full name of the now-connected directory. Causes an error, `Non-existent directory`, if *HOST/DIR* is not a valid directory.

Note that `CNDIR` does not necessarily require or provide any directory access privileges. Access privileges are checked when a file is opened.

`CONN` *HOST/DIR*

[Prog. Asst. Command]

Command form of `CNDIR` for use at the executive. Connects to *HOST/DIR*, or to the value of `LOGINHOST/DIR` if *HOST/DIR* is omitted. This command is undoable. —Undoing it causes the system to connect to the previously connected directory.

`LOGINHOST/DIR`

[Variable]

`CONN` with no argument connects to the value of the variable `LOGINHOST/DIR`, initially `{DSK}`, but usually reset in your greeting file (Chapter 12).

STREAMS & FILES

(**DIRECTORYNAME** *DIRNAME STRPTR*)

[Function]

If *DIRNAME* is T, returns the full specification of the currently connected directory. If *DIRNAME* is NIL, returns the value of LOGINHOST/DIR. For any other value of *DIRNAME*, returns a full directory specification if *DIRNAME* designates an existing directory (satisfies **DIRECTORYNAMEP**), otherwise NIL.

If *STRPTR* is T, the value is returned as an atom, otherwise it is returned as a string.

(**DIRECTORYNAMEP** *DIRNAME HOSTNAME*)

[Function]

Returns T if *DIRNAME* is a valid directory on host *HOSTNAME*, or on the host of the currently connected directory if *HOSTNAME* is NIL. *DIRNAME* may be either a directory name or a full directory specification containing host and/or device.

If *DIRNAME* includes subdirectories, this function may or may not pass judgment on their validity. Some hosts support "true" subdirectories, distinct entities manipulable by the file system, while others only provide them as a syntactic convenience.

(**HOSTNAMEP** *NAME*)

[Function]

Returns T if *NAME* is recognized as a valid host or file device name at the moment **HOSTNAMEP** is called.

Version Recognition

Most of the file devices in Interlisp support file version numbers. That is, you can have several files of the exact same name, differing only in their **VERSION** field, which is incremented for each new "version" of the file that is created. When the filesystem encounters a file name without a version number, it must figure out which version was intended. This process is known as *version recognition*.

When **OPENSTREAM** opens a file for input and no version number is given, the highest existing version number is used. Similarly, when a file is opened for output and no version number is given, a new file is created with a version number one higher than the highest one currently in use with that file name. You can change the version number defaulting for **OPENSTREAM** by specifying a different value for its *RECOG* argument (see **FULLNAME** below).

Other functions that accept file names as arguments generally perform default version recognition, which is newest version for existing files, or a new version if using the file name to create a new file. The one exception is **DELFILE**, which uses the oldest existing version of the file.

The functions below can be used to perform version recognition without actually calling **OPENSTREAM** to open the file. Note that these functions only tell the truth at the moment they are called, and thus cannot be used to anticipate the name of the file opened by a comparable **OPENSTREAM**. They are best used as helpful hints.

(**FULLNAME** *X RECOG*)

[Function]

If *X* is an open stream, simply returns the full file name of the stream. Otherwise, if *X* is a file name given as a string or symbol, performs version recognition, as follows:

INTERLISP-D REFERENCE MANUAL

If *X* is recognized in the recognition mode specified by *RECOG* as an abbreviation for some file, returns the file's full name, otherwise NIL. *RECOG* is one of the following:

- OLD Chooses the newest existing version of the file. Returns NIL if no file named *X* exists.
- OLDEST Chooses the oldest existing version of the file. Returns NIL if no file named *X* exists.
- NEW Chooses a new version of the file. If versions of *X* already exist, then chooses a version number one higher than highest existing version; otherwise chooses version 1. For some file systems, FULLNAME returns NIL if you do not have the access rights necessary to create a new file named *X*.
- OLD/NEW Tries OLD, then NEW. Choose the newest existing version of the file, if any; otherwise chooses version 1. This usually only makes sense if you intend to open *X* for access BOTH.

RECOG = NIL defaults to OLD. For all other values of *RECOG*, generates an error ILLEGAL ARG.

If *X* already contains a version number, the *RECOG* argument will never change it. In particular, *RECOG* = NEW does not require that the file actually be new. For example, (FULLNAME 'FOO.;2 'NEW) may return {ERIS}<LISP>FOO.;2 if that file already exists, even though (FULLNAME 'FOO 'NEW) would default the version to a new number, perhaps returning {ERIS}<LISP>FOO.;5.

(**INFILEP** *FILE*) [Function]

Equivalent to (FULLNAME *FILE* 'OLD). Returns the full file name of the newest version of *FILE* if *FILE* is the name of an existing file that can be opened for input, NIL otherwise.

(**OUTFILEP** *FILE*) [Function]

Equivalent to (FULLNAME *FILE* 'NEW).

Note that INFILEP, OUTFILEP and FULLNAME do not open any files; they are pure predicates. They are also only hints, as they do not imply that the caller has access rights to the file. For example, INFILEP might return non-NIL, but OPENSTREAM might fail for the same file because you don't have read access to it, or the file is open for output by another user. Similarly, OUTFILEP could return non-NIL, but OPENSTREAM could fail with a FILE SYSTEM RESOURCES EXCEEDED error.

Note also that in a shared file system, such as a remote file server, intervening file operations by another user could contradict the information returned by recognition. For example, a file that was INFILEP might be deleted, or between an OUTFILEP and the subsequent OPENSTREAM, another user might create a new version or delete the highest version, causing OPENSTREAM to open a different version of the file than the one returned by OUTFILEP. In addition, some file servers do not support recognition of files in output context. Thus, the "truth" about a file can only be obtained by actually opening the file; creators of files should rely on the name of the stream opened by OPENSTREAM, not

STREAMS & FILES

the value returned from these recognition functions. In particular, programmers are discouraged from using `OUTFILEP` or `(FULLNAME NAME 'NEW)`.

Using File Names Instead of Streams

In earlier implementations of Interlisp, from the days of Interlisp-10 onward, the "handle" used to refer to an open file was not a stream, but rather the file's full name, represented as a symbol. When the file name was passed to any I/O function, it was mapped to a stream by looking it up in a list of open files. This scheme was sometimes convenient for typing in file commands at the executive, but was poor for serious programming in two ways. First, mapping from file name to stream on every input/output operation is inefficient. Second, and more importantly, using the file name as the handle on an open stream means that it is not possible to have more than one stream open on a given file at once.

As of this writing, Medley is in a transition period, where it still supports the use of symbol file names as synonymous with open streams, but this use is not recommended. The remainder of this section discusses this usage of file names for the benefit of those reading older programs and wishing to convert them to work properly when this compatibility feature is removed.

File Name Efficiency Considerations

It is possible for a program to be seriously inefficient using a file name as a stream if the program is not using the name returned by `OPENFILE` (below). Any time that an input/output function is called with a file name other than the full file name, Interlisp must perform recognition on the partial file name to determine which open file is intended. Thus if repeated operations are to be performed, it is considerably more efficient to use the full file name returned from `OPENFILE`.

There is a more subtle problem with partial file names, in that recognition is performed on your entire directory, not just the open files. It is possible for a file name that previously denoted one file to suddenly denote a different file. For example, suppose a program performs `(INFILE 'FOO)`, opening `FOO. ; 1`, and reads several expressions from `FOO`. Then you interrupt the program, create a `FOO. ; 2` and resume the program (or a user at another workstation creates a `FOO. ; 2`). Now a call to `READ` giving it `FOO` as its *FILE* argument will generate a `FILE NOT OPEN` error, because `FOO` will be recognized as `FOO. ; 2`.

Obsolete File Opening Functions

The following functions are now obsolete, but are provided for backwards compatibility:

`(OPENFILE FILE ACCESS RECOG PARAMETERS)` [Function]

Opens *FILE* with access rights as specified by *ACCESS*, and recognition mode *RECOG*, and returns the full name of the resulting stream. Equivalent to `(FULLNAME (OPENSTREAM FILE ACCESS RECOG PARAMETERS))`.

`(INFILE FILE)` [Function]

Opens *FILE* for input, and sets it as the primary input stream. Equivalent to `(INPUT (OPENSTREAM FILE 'INPUT 'OLD))`

INTERLISP-D REFERENCE MANUAL

(**OUTFILE** *FILE*) [Function]

Opens *FILE* for output, and sets it as the primary output stream. Equivalent to (OUTPUT (OPENSTREAM *FILE* 'OUTPUT 'NEW)).

(**IOFILE** *FILE*) [Function]

Opens *FILE* for both input and output. Equivalent to (OPENFILE *FILE* 'BOTH 'OLD). Does not affect the primary input or output stream.

Converting Old Programs

At some point in the future, the Medley file system will change so that each call to OPENSTREAM returns a distinct stream, even if a stream is already open to the specified file. This change is required in order to deal with files in a multiprocessing environment.

This change will produce the following incompatibilities:

1. The functions OPENFILE, INPUT, and OUTPUT will return a stream, not a full file name. To make this less confusing in interactive situations, streams will have a print format that reveals the underlying file's actual name.
2. Passing anything other than the object returned from OPENFILE to I/O operations will cause problems. Passing the file's name will be significantly slower than passing the stream (even when passing the "full" file name), and in the case where there is more than one stream open on the file it might even act on the wrong one.
3. OPENP will return NIL when passed the name of a file rather than the value of OPENFILE or OPENSTREAM.

You should consider the following advice when writing new programs and editing existing programs, so your programs will behave properly when the change occurs:

Because of the efficiency and ambiguity considerations described earlier, users have long been encouraged to use only full file names as *FILE* arguments to I/O operations. The "proper" way to have done this was to bind a variable to the value returned from OPENFILE and pass that variable to all I/O operations; such code will continue to work. A less proper way to obtain the full file name, but one which has to date not incurred any obvious penalty, is that which binds a variable to the result of an INFILEP and passes that to OPENFILE and all I/O operations. This has worked because INFILEP and OPENFILE both return a full file name, an invalid assumption in this future world. Such code should be changed to pass around the value of the OPENFILE, not the INFILEP.

Code that calls OPENP to test whether a possibly incomplete file name is already open should be recoded to pass to OPENP only the value returned from OPENFILE or OPENSTREAM.

Code that uses ordinary string functions to manipulate file names, and in particular the value returned from OPENFILE, should be changed to use the the functions UNPACKFILENAME.STRING and PACKFILENAME.STRING. Those functions work both on file names (strings) and streams (coercing the stream to the name of its file).

Code that tests the value of OUTPUT for equality to some known file name or T should be examined carefully and, if possible, recoded.

STREAMS & FILES

To see more directly the effects of passing around streams instead of file names, replace your calls to `OPENFILE` with calls to `OPENSTREAM`. `OPENSTREAM` is called in exactly the same way, but returns a `STREAM`. Streams can be passed to `READ`, `PRINT`, `CLOSEF`, etc just as the file's full name can be currently, but using them is more efficient. The function `FULLNAME`, when applied to a stream, returns its full file name.

Using Files with Processes

Because Medley does not yet support multiple streams per file, problems can arise if different processes attempt to access the same file. You have to be careful not to have two processes manipulating the same file at the same time, since the two processes will be sharing a single input stream and file pointer. For example, you can't have one process `TCOMPL` a file while another process is running `LISTFILES` on it.

File Attributes

Any file has a number of "file attributes", such as the read date, protection, and bytesize. The exact attributes that a file can have is dependent on the file device. The functions `GETFILEINFO` and `SETFILEINFO` allow you to access file attributes:

(**GETFILEINFO** *FILE ATTRIB*) [Function]

Returns the current setting of the *ATTRIB* attribute of *FILE*.

(**SETFILEINFO** *FILE ATTRIB VALUE*) [Function]

Sets the attribute *ATTRIB* of *FILE* to be *VALUE*. `SETFILEINFO` returns `T` if it is able to change the attribute *ATTRIB*, and `NIL` if unsuccessful, either because the file device does not recognize *ATTRIB* or because the file device does not permit the attribute to be modified.

The *FILE* argument to `GETFILEINFO` and `SETFILEINFO` can be an open stream (or an argument designating an open stream, see Chapter 25), or the name of a closed file. `SETFILEINFO` in general requires write access to the file.

The attributes recognized by `GETFILEINFO` and `SETFILEINFO` fall into two categories: *permanent* attributes, which are properties of the file, and *temporary* attributes, which are properties only of an open stream to the file. The temporary attributes are only recognized when *FILE* designates an open stream; the permanent attributes are usually equally accessible for open and closed files. However, some devices are willing to change the value of certain attributes of an open stream only when specified in the `PARAMETERS` argument to `OPENSTREAM` (see above), not on a later call to `SETFILEINFO`.

The following are permanent attributes of a file:

- BYTESIZE** The byte size of the file. Medley currently only supports byte size 8.
- LENGTH** The number of bytes in the file. Alternatively, the byte position of the end-of-file. Like (`GETEOFPTR FILE`), but *FILE* does not have to be open.

INTERLISP-D REFERENCE MANUAL

SIZE	The size of <i>FILE</i> in pages.
CREATIONDATE	The date and time, as a string, that the content of <i>FILE</i> was "created". The creation date changes whenever the content of the file is modified, but remains unchanged when a file is transported, unmodified, across file systems. Specifically, <i>COPYFILE</i> and <i>RENAMEFILE</i> (see below) preserve the file's creation date. Note that this is different from the concept of "creation date" used by some operating systems (e.g., <i>Tops20</i>).
WRITEDATE	The date and time, as a string, that the content of <i>FILE</i> was last written to this particular file system. When a file is copied, its creation date does not change, but its write date becomes the time at which the copy is made.
READDATE	The date and time, as a string, that <i>FILE</i> was last read, or <i>NIL</i> if it has never been read.
ICREATIONDATE	
IWRITEDATE	
IREADDATE	The <i>CREATIONDATE</i> , <i>WRITEDATE</i> and <i>READDATE</i> , respectively, in integer form, as <i>IDATE</i> (Chapter 12) would return. This form is useful for comparing dates.
AUTHOR	The name of the user who last wrote the file.
TYPE	The "type" of the file, some indication of the nature of the file's content. The "types" of files allowed depends on the file device. Most devices recognize the symbol <i>TEXT</i> to mean that the file contains just characters, or <i>BINARY</i> to mean that the file contains arbitrary data.

Some devices support a wider range of file types that distinguish among the various sorts of files one might create whose content is "binary". All devices interpret any value of *TYPE* that they do not support to be *BINARY*. Thus, *GETFILEINFO* may return the more general value *BINARY* instead of the original type that was passed to *SETFILEINFO* or *OPENSTREAM*. Similarly, *COPYFILE*, while attempting to preserve the *TYPE* of the file it is copying, may turn, say, an *INTERPRESS* file into a mere *BINARY* file.

The way in which some file devices (e.g., Xerox file servers) support a wide range of file types is by representing the type as an integer, whose interpretation is known by the client. The variable *FILING.TYPES* is used to associate symbolic types with numbers for these devices. This list initially contains some of the well-known assignments of type name to number; you can add additional elements to handle any private file types. For example, suppose there existed an NS file type *MAZEFILE* with numeric value 5678. You could add the element (*MAZEFILE* 5678) to *FILING.TYPES* and then use *MAZEFILE* as a value for the *TYPE* attribute to *SETFILEINFO* or *OPENSTREAM*. Other devices are, of

STREAMS & FILES

course, free to store `TYPE` attributes in whatever manner they wish, be it numeric or symbolic. `FILING.TYPES` is merely considered the official registry for Xerox file types.

For most file devices, the `TYPE` of a newly created file, if not specified in the `PARAMETERS` argument to `OPENSTREAM`, defaults to the value of `DEFAULTFILETYPE`, initially `TEXT`.

The following are currently recognized as temporary attributes of an open stream:

- `ACCESS` The current access rights of the stream (see the beginning of this chapter). Can be one of `INPUT`, `OUTPUT`, `BOTH`, `APPEND`; or `NIL` if the stream is not open.
- `ENDOFSTREAMOP` The action to be taken when a stream is at "end of file" and an attempt is made to take input from it. The value of this attribute is a function of one argument, the stream. The function can examine the stream and its calling context and take any action it wishes. If the function returns normally, it should return either `T`, meaning to try the input operation again, or the byte that `BIN` would have returned had there been more bytes to read. Ordinarily, one should not let the `ENDOFSTREAMOP` function return unless one is only performing binary input from the file, since there is no way in general of knowing in what state the reader was at the time the end of file occurred, and hence how it will interpret a single byte returned to it.
- The default `ENDOFSTREAMOP` is a system function that causes the error `END OF FILE`. The behavior of that error can be further modified for a particular stream by using the `EOF` option of `WHENCLOSE` (see below).
- `EOL` The end-of-line convention for the stream. This can be `CR`, `LF`, or `CRLF`, indicating with what byte or sequence of bytes the "End Of Line" character is represented on the stream. On input, that sequence of bytes on the stream is read as `(CHARCODE EOL)` by `READCCODE` or the string reader. On output, `(TERPRI)` and `(PRINTCCODE (CHARCODE EOL))` cause that sequence of bytes to be placed on the stream.
- The end of line convention is usually not apparent to you. The file system is usually aware of the convention used by a particular remote operating system, and sets this attribute accordingly. If you believe a file actually is stored with a different convention than the default, it is possible to modify the default behavior by including the `EOL` attribute in the `PARAMETERS` argument to `OPENSTREAM`.
- `BUFFERS` Value is the number of 512-byte buffers that the stream maintains at one time. This attribute is only used by certain random-access devices (currently, the local disk, floppy, and Leaf servers); all others ignore it.

Streams open to files generally maintain some portion of the file buffered in memory, so that each call to an I/O function does not

INTERLISP-D REFERENCE MANUAL

require accessing the actual file on disk or a file server. For files being read or written sequentially, not much buffer space is needed, since once a byte is read or written, it will never need to be seen again. In the case of random access streams, buffering is more complicated, since a program may jump around in the file, using `SETFILEPTR` (Chapter 25). In this case, the more buffer space the stream has, the more likely it is that after a `SETFILEPTR` to a place in the file that has already been accessed, the stream still has that part of the file buffered and need not go out to the device again. This benefit must, of course, be traded off against the amount of memory consumed by the buffers.

NS servers implement the following additional attributes for `GETFILEINFO` (neither of these attributes are settable with `SETFILEINFO`):

READER	The name of the user who last read the file.
PROTECTION	A list specifying the access rights to the file. Each element of the list is of the form (name nametype . rights). Name is the name of a user or group or a name pattern. Rights is one or more of the symbols ALL READ WRITE DELETE CREATE or MODIFY. For servers running services 10.0 or later, nametype is the symbol "--". , In earlier releases it is one of the symbols INDIVIDUAL or GROUP

Closing and Reopening Files

The function `WHENCLOSE` permits you to associate certain operations with open streams that govern how and when the stream will be closed. You can specify that certain functions will be executed before `CLOSEF` closes the stream and/or after `CLOSEF` closes the stream. You can make a particular stream be invisible to `CLOSEALL`, so that it will remain open across user invocations of `CLOSEALL`.

(**WHENCLOSE** *FILE* *PROP VAL* ... *PROP VAL*) [NoSpread Function]

FILE must designate an open stream other than T (NIL defaults to the primary input stream, if other than T, or primary output stream if other than T). The remaining arguments specify properties to be associated with the full name of *FILE*. `WHENCLOSE` returns the full name of *FILE* as its value.

`WHENCLOSE` recognizes the following property names:

BEFORE	<i>VAL</i> is a function that <code>CLOSEF</code> will apply to the stream just before it is closed. This might be used, for example, to copy information about the file from an in-core data structure to the file just before it is closed.
AFTER	<i>VAL</i> is a function that <code>CLOSEF</code> will apply to the stream just after it is closed. This capability permits in-core data structures that know about the stream to be cleaned up when the stream is closed.
CLOSEALL	<i>VAL</i> is either YES or NO and determines whether <i>FILE</i> will be closed by <code>CLOSEALL</code> (YES) or whether <code>CLOSEALL</code> will ignore it (NO). <code>CLOSEALL</code>

STREAMS & FILES

uses `CLOSEF`, so that any `AFTER` functions will be executed if the stream is in fact closed. Files are initialized with `CLOSEALL` set to `YES`.

EOF `VAL` is a function that will be applied to the stream when an end-of-file error occurs, and the `ERRORTYPELAST` entry for that error, if any, returns `NIL`. The function can examine the context of the error, and can decide whether to close the stream, `RETFROM` some function, or perform some other computation. If the function supplied returns normally (i.e., does not `RETFROM` some function), the normal error machinery will be invoked.

The default EOF behavior, unless overridden by this `WHENCLOSE` option, is to call the value of `DEFAULTEOFCLOSE` (below).

For some applications, the `ENDOFSTREAMOP` attribute (see above) is a more useful way to intercept the end-of-file error. The `ENDOFSTREAMOP` attribute comes into effect before the error machinery is ever activated.

Multiple `AFTER` and `BEFORE` functions may be associated with a file; they are executed in sequence with the most recently associated function executed first. The `CLOSEALL` and `EOF` values, however, will override earlier values, so only the last value specified will have an effect.

DEFAULTEOFCLOSE

[Variable]

Value is the name of a function that is called by default when an end of file error occurs and no EOF option has been specified for the stream by `WHENCLOSE`. The initial value of `DEFAULTEOFCLOSE` is `NILL`, meaning take no special action (go ahead and cause the error). Setting it to `CLOSEF` would cause the stream to be closed before the rest of the error machinery is invoked.

I/O Operations to and from Strings

It is possible to treat a string as if it were the contents of a file by using the following function:

(OPENSTRINGSTREAM STR ACCESS)

[Function]

Returns a stream that can be used to access the characters of the string `STR`. `ACCESS` may be either `INPUT`, `OUTPUT`, or `BOTH`; `NIL` defaults to `INPUT`. The stream returned may be used exactly like a file opened with the same access, except that output operations may not extend past the end of the original string. Also, string streams do not appear in the value of `(OPENP)`.

For example, after performing

```
(SETQ STRM (OPENSTRINGSTREAM "THIS 2 (IS A LIST)"))
```

the following succession of reads could occur:

INTERLISP-D REFERENCE MANUAL

```
(READ STRM) => THIS
(RATOM STRM) => 2
(READ STRM) => (IS A LIST)
(EOFP STRM) => T
```

Compatibility Note: In Interlisp-10 it was possible to take input from a string simply by passing the string as the *FILE* argument to an input function. In order to maintain compatibility with this feature, Medley provides the same capability. This not terribly clean feature persists in the present implementation to give users time to convert old code. This means that strings are *not* equivalent to symbols when specifying a file name as a stream argument. In a future release, the old Interlisp-10 string-reading feature will be decommissioned, and *OPENSTRINGSTREAM* will be the only way to perform I/O on a string.

Temporary Files and the CORE Device

Many operating systems have a notion of "scratch file", a file typically used as temporary storage for data most naturally maintained in the form of a file, rather than some other data structure. A scratch file can be used as a normal file in most respects, but is automatically deleted from the file system after its useful life is up, e.g., when the job terminates, or you log out. In normal operation, you need never explicitly delete such files, since they are guaranteed to disappear soon.

A similar functionality is provided in Medley by core-resident files. Core-resident files are on the device *CORE*. The directory structure for this device and all files on it are represented completely within your virtual memory. These files are treated as ordinary files by all file operations; their only distinguishing feature is that all trace of them disappears when the Medley image is abandoned.

Core files are opened and closed by name the same as any other file, e.g., (*OPENSTREAM* ' {*CORE*} <*FOO*> *FIE.DCOM* ' *OUTPUT*). Directory names are completely optional, so files can also have names of the form {*CORE*}*NAME.EXT*. Core files can be enumerated by *DIRECTORY* (see below). While open, they are registered in (*OPENP*). They do consume virtual memory space, which is only reclaimed when the file is deleted. Some caution should thus be used when creating large *CORE* files. Since the virtual memory of an Medley workstation usually persists far longer than the typical process on a mainframe computer, it is still important to delete *CORE* files after they are no longer in use.

For many applications, the name of the scratch file is irrelevant, and there is no need for anyone to have access to the file independent of the program that created it. For such applications, *NODIRCORE* files are preferable. Files created on the device lisp *NODIRCORE* are core-resident files that have no name and are registered in no directory. These files "disappear", and the resources they consume are reclaimed, when all pointers to the file are dropped. Hence, such files need never be explicitly deleted or, for that matter, closed. The "name" of such a file is simply the stream object returned from (*OPENSTREAM* ' {*NODIRCORE*} ' *OUTPUT*), and it is this stream object that must be passed to all input/output operations, including *CLOSEF* and any calls to *OPENSTREAM* to reopen the file.

(**COREDEVICE** *NAME* *NODIRFLG*)

[Function]

Creates a new device for core-resident files and assigns *NAME* as its device name. Thus, after performing (*COREDEVICE* ' *FOO*), one can execute (*OPENSTREAM* ' {*FOO*} *BAR* ' *OUTPUT*) to open a file on that device. Medley is initialized with the single core-resident device named *CORE*, but *COREDEVICE* may be used to create any number of logically distinct core devices.

STREAMS & FILES

If *NODIRFLG* is non-NIL, a core device that acts like {NODIRCORE} is created.

Compatibility note: In Interlisp-10, it was possible to create scratch files by using file names with suffixes ;S or ;T. In Medley, these suffixes in file names are simply ignored when output is directed to a particular host or device. However, the function *PACKFILENAME.STRING* is defined to default the device name to *CORE* if the file has the *TEMPORARY* attribute and no explicit host is provided.

NULL Device

The NULL device provides a source of content-free "files". (*OPENSTREAM* ' {NULL} 'OUTPUT) creates a stream that discards all output directed at it. (*OPENSTREAM* ' {NULL} 'INPUT) creates a stream that is perpetually at end-of-file (i.e., has no input).

Deleting, Copying, and Renaming Files

(*DELFILE FILE*)

[Function]

Deletes *FILE* if possible. The file must be closed. Returns the full name of the file if deleted, else NIL. Recognition mode for *FILE* is *OLDEST*, i.e., if *FILE* does not have a version number specified, then *DELFILE* deletes the oldest version of the file.

(*COPYFILE FROMFILE TOFILE*)

[Function]

Copies *FROMFILE* to a new file named *TOFILE*. The source and destination may be on any combination of hosts/devices. *COPYFILE* attempts to preserve the *TYPE* and *CREATIONDATE* where possible. If the original file's file type is unknown, *COPYFILE* attempts to infer the type (file type is *BINARY* if any of its 8-bit bytes have their high bit on).

COPYFILE uses *COPYCHARS* (Chapter 25) if the source and destination hosts have different EOL conventions. Thus, it is possible for the source and destination files to be of different lengths.

(*RENAMEFILE OLDFILE NEWFILE*)

[Function]

Renames *OLDFILE* to be *NEWFILE*. Causes an error, *FILE NOT FOUND* if *FILE* does not exist. Returns the full name of the new file, if successful, else NIL if the rename cannot be performed.

If *OLDFILE* and *NEWFILE* are on the same host/device, and the device implements a renaming primitive, *RENAMEFILE* can be very fast. However, if the device does not know how to rename files in place, or if *OLDFILE* and *NEWFILE* are on different devices, *RENAMEFILE* works by copying *OLDFILE* to *NEWFILE* and then deleting *OLDFILE*.

Searching File Directories

DIRECTORIES [Variable]

Global variable containing the list of directories searched (in order) by SPELLFILE and FINDFILE (below) when not given an explicit *DIRLIST* argument. In this list, the atom NIL stands for the login directory (the value of LOGINHOST/DIR), and the atom T stands for the currently connected directory. Other elements should be *full* directory specifications, e.g., {TWENTY}PS:<LISPUSERS>, not merely LISPUSERS.

LISPUSERSDIRECTORIES [Variable]

Global variable containing a list of directories to search for "library" package files. Used by the FILES file package command (Chapter 17).

(SPELLFILE FILE NOPRINTFLG NSFLG DIRLIST) [Function]

Searches for the file name *FILE*, possibly performing spelling correction (see Chapter 20). Returns the corrected file name, if any, otherwise NIL.

If *FILE* has a directory field, SPELLFILE attempts spelling correction against the files in that particular directory. Otherwise, SPELLFILE searches for the file on the directory list *DIRLIST* before attempting any spelling correction.

If *NOPRINTFLG* is NIL, SPELLFILE asks you to confirm any spelling correction done, and prints out any files found, even if spelling correction is not done. If *NOPRINTFLG* = T, SPELLFILE does not do any printing, nor ask for approval.

If *NSFLG* = T (or *NOSPELLFLG* = T, see Chapter 20), no spelling correction is attempted, though searching through *DIRLIST* still occurs.

DIRLIST is the list of directories searched if *FILE* does not have a directory field. If *DIRLIST* is NIL, the value of the variable DIRECTORIES is used.

Note: If *DIRLIST* is NIL, and *FILE* is not found by searching the directories on DIRECTORIES, but the root name of *FILE* has a FILEDATES property (Chapter 17) indicating that a file by that name has been loaded, then the directory indicated in the FILEDATES property is searched, too. This additional search is not done if *DIRLIST* is non-NIL.

ERRORTYPELIST (Chapter 14) initially contains the entry ((23 (SPELLFILE (CADR ERRORMESS) NIL NOFILESPELLFLG))), which causes SPELLFILE to be called in case of a FILE NOT FOUND error. If the variable NOFILESPELLFLG is T (its initial value), then spelling correction is not done on the file name, but DIRECTORIES is still searched. If SPELLFILE is successful, the operation will be reexecuted with the new (corrected) file name.

STREAMS & FILES

(**FINDFILE** *FILE* *NSFLG* *DIRLST*)

[Function]

Uses *SPELLFILE* to search for a file named *FILE*. If it finds one, returns its full name, with no user interaction. Specifically, it calls (*SPELLFILE FILE T NSFLG DIRLST*), after first performing two simple checks: If *FILE* has an explicit directory, it checks to see if a file so named exists, and if so returns that file. If *DIRLST* is *NIL*, it looks for *FILE* on the connected directory before calling *SPELLFILE*.

Listing File Directories

The function *DIRECTORY* allows you to conveniently specify and/or program a variety of directory operations:

(**DIRECTORY** *FILES* *COMMANDS* *DEFAULTTEXT* *DEFAULTVERS*)

[Function]

Returns, lists, or performs arbitrary operations on all files specified by the "file group" *FILES*. A file group has the form of a regular file name, except that the character *** can be used to match any number of characters, including zero, in the file name. For example, the file group *A*B* matches all file names beginning with the character *A* and ending with the character *B*. The file group **.DCOM* matches all files with an extension of *DCOM*.

If *FILES* does not contain an explicit extension, it is defaulted to *DEFAULTTEXT*; if *FILES* does not contain an explicit version, it is defaulted to *DEFAULTVERS*. *DEFAULTTEXT* and *DEFAULTVERS* themselves default to ***. If the period or semicolon preceding the omitted extension or version, respectively, is present, the field is explicitly empty and no default is used. All other unspecified fields default to ***. Null version is interpreted as "highest". Thus *FILES* = *** or **.** or **.*;* enumerates all files on the connected directory; *FILES* = **.* or **.*;* enumerates all versions of files with null extension; *FILES* = **.*;* enumerates the highest version of files with null extension; and *FILES* = **.*;* enumerates the highest version of all files. If *FILES* is *NIL*, it defaults to **.*;*.

Note: Some hosts/devices are not capable of supporting "highest version" in enumeration. Such hosts instead enumerate *all* versions.

For each file that matches the file group *FILES*, the "file commands" in *COMMANDS* are executed in order. Some of the file commands allow aborting the command processing for a given file, effectively filtering the list of files. The interpretation of the different file commands is described below. If *COMMANDS* is *NIL*, it defaults to (*COLLECT*), which collects the matching file names in a list and returns it as the value of *DIRECTORY*.

The "file commands" in *COMMANDS* are interpreted as follows:

P Prints the file's name. For readability, *DIRECTORY* strips the directory from the name, printing it once as a header in front of each set of consecutive files on the same directory.

PP Prints the file's name without a version number.

a string Prints the string.

READDATE, WRITEDATE

INTERLISP-D REFERENCE MANUAL

CREATIONDATE, SIZE
LENGTH, BYTESIZE
PROTECTION, AUTHOR

TYPE	Prints the appropriate information returned by GETFILEINFO (see above).
COLLECT	Adds the full name of this file to an accumulating list, which will be returned as the value of DIRECTORY.
COUNTSIZE	Adds the size of this file to an accumulating sum, which will be returned as the value of DIRECTORY.
DELETE	Deletes the file.
DELVER	If this file is not the highest version of files by its name, delete it.
PAUSE	Waits until you type any character before proceeding with the rest of the commands (good for display if you want to ponder).

The following commands are predicates to filter the list. If the predicate is not satisfied, then processing for this file is aborted and no further commands (such as those above) are executed for this file.

Note: if the P and PP commands appear in *COMMANDS* ahead of any of the filtering commands below except PROMPT, they are postponed until after the filters. Thus, assuming the caller has placed the attribute options after the filters as well, no printing occurs for a file that is filtered out. This is principally so that functions like DIR (below) can both request printing and pass arbitrary commands through to DIRECTORY, and have the printing happen in the appropriate place.

PROMPT MESS	Prompts with the yes/no question <i>MESS</i> ; if user responds with No, abort command processing for this file.
OLDERTHAN N	Continue command processing if the file hasn't been referenced (read or written) in <i>N</i> days. <i>N</i> can also be a string naming an explicit date and time since which the file must not have been referenced.
NEWERTHAN N	Continue command processing if the file has been written within the last <i>N</i> days. <i>N</i> can also be a string naming an explicit date and time. Note that this is not quite the complement of OLDERTHAN, since it ignores the read date.
BY USER	Continue command processing if the file was last written by the given user, i.e., its AUTHOR attribute matches (case insensitively) <i>USER</i> .
@ X	<i>X</i> is either a function of one argument (<i>FILENAME</i>), or an arbitrary expression which uses the variable <i>FILENAME</i> freely. If <i>X</i> returns NIL, abort command processing for this file.

The following two commands apply not to any particular file, but globally to the manner in which directory information is printed.

STREAMS & FILES

OUT *FILE* Directs output to *FILE*.

COLUMNS *N* Attempts to format output in *N* columns (rather than just 1).

DIRECTORY uses the variable **DIRCOMMANDS** as a spelling list to correct spelling and define abbreviations and synonyms (see Chapter 20). Currently the following abbreviations are recognized:

AU	=>	AUTHOR
-	=>	PAUSE
COLLECT?	=>	PROMPT " ? " COLLECT
DA		
DATE	=>	CREATIONDATE
TI	=>	WRITEDATE
DEL	=>	DELETE
DEL?		
DELETE?	=>	PROMPT " delete? " DELETE
OLD	=>	OLDERTHAN 90
PR	=>	PROTECTION
SI	=>	SIZE
VERBOSE	=>	AUTHOR CREATIONDATE SIZE READDATE WRITEDATE

(*FILDIR FILEGROUP*) [Function]

Obsolete synonym of (**DIRECTORY *FILEGROUP***).

(*DIR FILEGROUP COM ... COM*) [NLambda NoSpread Function]

Convenient form of **DIRECTORY** for use in type-in at the executive. Performs (**DIRECTORY ' *FILEGROUP* ' (P *COM ... COM*)**).

(*NDIR FILEGROUP COM ... COM*) [NLambda NoSpread Function]

Version of **DIR** that lists the file names in a multi-column format. Also, by default only lists the most recent version of files (unless *FILEGROUP* contains an explicit version).

23. STREAMS AND FILES

Medley can perform input/output operations on a large variety of physical devices, including local disk drives, floppy disk drives, the keyboard and display screen, and remote file server computers accessed over a network. While the low-level details of how all these devices perform input/output vary considerably, the Interlisp-D language provides the programmer a small, common set of abstract operations whose use is largely independent of the physical input/output medium involved—operations such as *read*, *print*, *change font*, or *go to a new line*. By merely changing the targeted I/O device, a single program can be used to produce output on the display, a file, or a printer.

The underlying data abstraction that permits this flexibility is the *stream*. A stream is a data object (an instance of the data type `STREAM`) that encapsulates all of the information about an input/output connection to a particular I/O device. Each of Medley's general-purpose I/O functions takes a stream as one of its arguments. The general-purpose function then performs action specific to the stream's device to carry out the requested operation. Not every device is capable of implementing every I/O operation, while some devices offer additional functionality by way of special functions for that device alone. Such restrictions and extensions are noted in the documentation of each device.

The vast majority of the streams commonly used in Medley fall into two interesting categories: the *file stream* and the *image stream*.

A file is an ordered collection of data, usually a sequence of characters or bytes, stored on a file device in a manner that allows the data to be retrieved at a later time. Floppy disks, hard disks, and remote file servers are among the devices used to store files. Files are identified by a "file name", which specifies the device on which the file resides and a name unique to a specific file on that device. Input or output to a file is performed by obtaining a stream to the file, using `OPENSTREAM` (see below). In addition, there are functions that manipulate the files themselves, rather than their data content.

An image stream is an output stream to a display device, such as the display screen or a printer. In addition to the standard output operations, such as `print`, an image stream implements a variety of graphics operations, such as drawing lines and displaying characters in multiple fonts. Unlike a file, the "content" of an image stream cannot be retrieved. Image streams are described in Chapter 26.

The creation of other kinds of streams, such as network byte-stream connections, is described in the chapters peculiar to those kinds of streams. The operations common to streams in general are described in Chapter 24. This chapter describes operations specific to file devices: how to name files, how to open streams to files, and how to manipulate files on their devices.

Opening and Closing File Streams

In order to perform input from or output to a file, it is necessary to create a stream to the file, using `OPENSTREAM`:

INTERLISP-D REFERENCE MANUAL

(OPENSTREAM *FILE ACCESS RECOG PARAMETERS* →)

[Function]

Opens and returns a stream for the file specified by *FILE*, a file name. *FILE* can be either a string or a symbol. The syntax and manipulation of file names is described at length in the *FILENAMES* section below. Incomplete file names are interpreted with respect to the connected directory (below).

RECOG specifies the recognition mode of *FILE*, as described in a later section of this chapter. If *RECOG* = *NIL*, it defaults according to the value of *ACCESS*.

ACCESS specifies the "access rights" to be used when opening the file, one of the following:

- INPUT Only input operations are permitted on the file. The file must already exist. Starts reading at the beginning of the file. *RECOG* defaults to *OLD*.
- OUTPUT Only output operations are permitted on the file. Starts writing at the beginning of the file, which is initially empty. While the file is open, other users or processes are unable to open the file for either input or output. *RECOG* defaults to *NEW*.
- BOTH Both input and output operations are permitted on the file. Starts reading or writing at the beginning of the file. *RECOG* defaults to *OLD/NEW*. *ACCESS* = *BOTH* implies random accessibility (Chapter 25), and thus may not be possible for files on some devices.
- APPEND Only sequential output operations are permitted on the file. Starts writing at the *end* of the file. *RECOG* defaults to *OLD/NEW*. *ACCESS* = *APPEND* may not be allowed for files on some devices.

Note: *ACCESS* = *OUTPUT* implies that one intends to write a new or different file, even if a version number was specified and the corresponding file already exists. Thus any previous contents of the file are discarded, and the file is empty immediately after the *OPENSTREAM*. If it is desired to write on an already existing file while preserving the old contents, the file must be opened for access *BOTH* or *APPEND*.

PARAMETERS is a list of pairs (*ATTRIB VALUE*), where *ATTRIB* is any file attribute that the file system is willing to allow you to set (see *SETFILEINFO* below). A non-list *ATTRIB* in *PARAMETERS* is treated as the pair (*ATTRIB T*). Generally speaking, attributes that belong to the permanent file (e.g., *TYPE*) can only be set when creating a new file, while attributes that belong only to a particular opening of a file (e.g., *ENDOFSTREAMOP*) can be set on any call to *OPENSTREAM*. Not all devices honor all attributes; those not recognized by a particular device are simply ignored.

In addition to the attributes permitted by *SETFILEINFO*, the following tokens are accepted by *OPENSTREAM* as values of *ATTRIB* in its *PARAMETERS* argument:

STREAMS & FILES

DON'T.CHANGE.DATE	If <i>VALUE</i> is non-NIL, the file's creation date is not changed when the file is opened. This option is meaningful only for old files being opened for access BOTH. This should be used only for specialized applications in which the caller does not want the file system to believe the file's content has been changed.
SEQUENTIAL	If <i>VALUE</i> is non-NIL, this opening of the file need support only sequential access; i.e., the caller intends never to use SETFILEPTR. For some devices, sequential access to files is much more efficient than random access. Note that the device may choose to ignore this attribute and still open the file in a manner that permits random access. Also note that this attribute does not make sense with <i>ACCESS</i> = BOTH.

If *FILE* is not recognized by the file system, OPENSTREAM causes the error FILE NOT FOUND. Ordinarily, this error is intercepted via an entry on ERRORTYPELIST (Chapter 24), which causes SPELLFILE (see the Searching File Directories section of this chapter) to be called. SPELLFILE searches alternate directories and possibly attempts spelling correction on the file name. Only if SPELLFILE is unsuccessful will the FILE NOT FOUND error actually occur.

If *FILE* exists but cannot be opened, OPENSTREAM causes one of several other errors: FILE WON'T OPEN if the file is already opened for conflicting access by someone else; PROTECTION VIOLATION if the file is protected against the operation; FILE SYSTEM RESOURCES EXCEEDED if there is no more room in the file system.

(CLOSEF *FILE*)

[Function]

Closes *FILE*, and returns its full file name. Generates an error, FILE NOT OPEN, if *FILE* does not designate an open stream. After closing a stream, no further input/output operations are permitted on it.

If *FILE* is NIL, it is defaulted to the primary input stream if that is not the terminal stream, or else the primary output stream if that is not the terminal stream. If both primary input and output streams are the terminal input/output streams, CLOSEF returns NIL. If CLOSEF closes either the primary input stream or the primary output stream (either explicitly or in the *FILE* = NIL case), it resets the primary stream for that direction to be the corresponding terminal stream. See Chapter 25 for information on the primary input/output streams.

WHENCLOSE (see below) allows you to "advise" CLOSEF to perform various operations when a file is closed.

Because of buffering, the contents of a file open for output are not guaranteed to be written to the actual physical file device until CLOSEF is called. Buffered data can be

INTERLISP-D REFERENCE MANUAL

forced out to a file without closing the file by using the function `FORCEOUTPUT` (Chapter 25).

Some network file devices perform their transactions in the background. As a result, it is possible for a file to be closed by `CLOSEF` and yet not be "fully" closed for some small period of time afterward, during which time the file appears to still be busy, and cannot be opened for conflicting access by other users.

(`CLOSEF? FILE`) [Function]

Closes *FILE* if it is open, returning the value of `CLOSEF`; otherwise does nothing and returns `NIL`.

In the present implementation of Medley, all streams to files are kept, while open, in a registry of "open files". This registry does not include nameless streams, such as string streams (see below), display streams (Chapter 28), and the terminal input and output streams; nor streams explicitly hidden from you, such as dribble streams (Chapter 30). This registry may not persist in future implementations of Medley, but at the present time it is accessible by the following two functions:

(`OPENP FILE ACCESS`) [Function]

ACCESS is an access mode for a stream opening (one of `INPUT`, `OUTPUT`, `BOTH`, or `APPEND`), or `NIL`, meaning any access.

If *FILE* is a stream, returns its full name if it is open for the specified access, else `NIL`.

If *FILE* is a file name (a symbol), *FILE* is processed according to the rules of file recognition (see below). If a stream open to a file by that name is registered and open for the specified access, then the file's full name is returned. If the file name is not recognized, or no stream is open to the file with the specified access, `NIL` is returned.

If *FILE* is `NIL`, returns a list of the full names of all registered streams that are open for the specified access.

(`CLOSEALL ALLFLG`) [Function]

Closes all streams in the value of (`OPENP`). Returns a list of the files closed.

`WHENCLOSE` (see below) allows certain files to be "protected" from `CLOSEALL`. If *ALLFLG* is `T`, all files, including those protected by `WHENCLOSE`, are closed.

File Names

A file name in Medley is a string or symbol whose characters specify a "path" to the actual file: on what host or device the file resides, in which directory, and so forth. Because Medley supports a variety of non-local file devices, parts of the path could be very device-dependent. However, it is desirable for programs to be able to manipulate file names in a device-independent manner. To this end, Medley specifies a uniform file name syntax over all devices; the functions that perform the

STREAMS & FILES

actual file manipulation for a particular device are responsible for any translation to that device's naming conventions.

A file name is composed of a collection of *fields*, some of which have specific semantic interpretations. The functions described below refer to each field by a *field name*, a literal atom from among the following: HOST, DEVICE, DIRECTORY, NAME, EXTENSION, and VERSION. The standard syntax for a file name that contains all of those fields is {HOST}DEVICE:<DIRECTORY>NAME.EXTENSION;VERSION. Some host's file systems do not use all of those fields in their file names.

HOST	Specifies the host whose file system contains the file. In the case of local file devices, the "host" is the name of the device, e.g., DSK or FLOPPY.
DEVICE	Specifies, for those hosts that divide their file system's name space among multiple physical devices, the device or logical structure on which the file resides. This should not be confused with Medley's abstract "file device", which denotes either a host or a local physical device and is specified by the HOST field.
DIRECTORY	Specifies the "directory" containing the file. A directory usually is a grouping of a possibly large set of loosely related files, e.g., the personal files of a particular user, or the files belonging to some project. The DIRECTORY field usually consists of a principal directory and zero or more subdirectories that together describe a path through a file system's hierarchy. Each subdirectory name is set off from the previous directory or subdirectory by the character ">"; e.g., "LISP>LIBRARY>NEW".
NAME	This field carries no specific meaning, but generally names a set of files thought of as being different renditions of the "same" abstract file.
EXTENSION	This field also carries no specific meaning, but generally distinguishes the form of files having the same name. Most files systems have some "conventional" extensions that denote something about the content of the file. For example, in Medley, the extension DCOM standardly denotes a file containing compiled function definitions.
VERSION	A number used to distinguish the versions or "generations" of the files having a common name and extension. The version number is incremented each time a new file by the same name is created.

Most functions that take as input "a directory" accept either a directory name (the contents of the DIRECTORY field of a file name) or a "full" directory specification—a file name fragment consisting of

INTERLISP-D REFERENCE MANUAL

only the fields HOST, DEVICE, and DIRECTORY. In particular, the "connected directory" (see below) consists, in general, of all three fields.

For convenience in dealing with certain operating systems, Medley also recognizes [] and () as host delimiters (synonymous with {}), and / as a directory delimiter (synonymous with < at the beginning of a directory specification and > to terminate directory or subdirectory specification). For example, a file on a Unix file server UNX with the name /usr/foo/bar/stuff.tedit, whose DIRECTORY field is thus usr/foo/bar, could be specified as {UNX}/usr/foo/bar/stuff.tedit, or (UNX)<usr/foo/bar>stuff.tedit, or several other variations. Note that when using [] or () as host delimiters, they usually must be escaped with the reader's % escape character if the file name is expressed as a symbol rather than a string.

Different hosts have different requirements regarding which characters are valid in file names. From Medley's point of view, any characters are valid. However, in order to be able to parse a file name into its component fields, it is necessary that those characters that are conventionally used as file name delimiters be quoted when they appear inside of fields where there could be ambiguity. The file name quoting character is "'" (single quote). Thus, the following characters must be quoted when not used as delimiters: :, >, ;, /, and ' itself. The character . (period) need only be quoted if it is to be considered a part of the EXTENSION field. The characters },], and) need only be quoted in a file name when the host field of the name is introduced by {, [, and (, respectively. The characters {, [, (, and < need only be quoted if they appear as the first character of a file name fragment, where they would otherwise be assumed to introduce the HOST or DIRECTORY fields.

The following functions are the standard way to manipulate file names in Interlisp. Their operation is purely syntactic—they perform no file system operations themselves.

(UNPACKFILENAME.STRING *FILENAME*) [Function]

Parses *FILENAME*, returning a list in property list format of alternating field names and field contents. The field contents are returned as strings. If *FILENAME* is a stream, its full name is used.

Only those fields actually present in *FILENAME* are returned. A field is considered present if its delimiting punctuation (in the case of EXTENSION and VERSION, the preceding period or semicolon, respectively) is present, even if the field itself is empty. Empty fields are denoted by "" (the empty string).

Examples:

```
(UNPACKFILENAME.STRING "FOO.BAR") =>
  (NAME "FOO" EXTENSION "BAR")
(UNPACKFILENAME.STRING "FOO.;2") =>
  (NAME "FOO" EXTENSION "" VERSION "2")
(UNPACKFILENAME.STRING "FOO;") =>
  (NAME "FOO" VERSION "")
(UNPACKFILENAME.STRING
  "{ERIS}<LISP>CURRENT>IMTRAN.DCOM;21")
=> (HOST "ERIS" DIRECTORY "LISP>CURRENT"
    NAME "IMTRAN" EXTENSION "DCOM"
    VERSION "21")
```

STREAMS & FILES

(UNPACKFILENAME *FILE*)

[Function]

Old version of UNPACKFILENAME.STRING that returns the field values as atoms, rather than as strings. UNPACKFILENAME.STRING is now considered the "correct" way of unpacking file names, because it does not lose information when the contents of a field are numeric. For example,

```
(UNPACKFILENAME 'STUFF.TXT) =>
(NAME STUFF EXTENSION TXT)
```

but

```
(UNPACKFILENAME 'STUFF.029) =>
(NAME STUFF EXTENSION 29)
```

Explicitly omitted fields are denoted by the atom NIL, rather than the empty string.

Note: Both UNPACKFILENAME and UNPACKFILENAME.STRING leave the trailing colon on the device field, so that the Tenex device NIL: can be distinguished from the absence of a device. Although UNPACKFILENAME.STRING is capable of making the distinction, it retains this behavior for backward compatibility. Thus,

```
(UNPACKFILENAME.STRING '{TOAST}DSK:FOO) =>
(HOST "TOAST" DEVICE "DSK:" NAME "FOO")
```

(FILENAMEFIELD *FILENAME FIELDNAME*)

[Function]

Returns, as an atom, the contents of the *FIELDNAME* field of *FILENAME*. If *FILENAME* is a stream, its full name is used.

(PACKFILENAME.STRING *FIELD CONTENTS ... FIELD CONTENTS*)
Function]

[NoSpread

Takes a sequence of alternating field names and field contents (atoms or strings), and returns the corresponding file name, as a string.

If PACKFILENAME.STRING is given a single argument, it is interpreted as a list of alternating field names and field contents. Thus PACKFILENAME.STRING and UNPACKFILENAME.STRING operate as inverses.

If the same field name is given twice, the *first* occurrence is used.

The contents of the field name DIRECTORY may be either a directory name or a full directory specification as described above.

PACKFILENAME.STRING also accepts the "field name" BODY to mean that its contents should itself be unpacked and spliced into the argument list at that point. This feature, in conjunction with the rule that fields early in the argument list override later duplicates, is useful for altering existing file names. For example, to provide a default field, place BODY first in the argument list, then the default fields. To override a field, place the new fields first and BODY last.

If the value of the BODY field is a stream, its full name is used.

INTERLISP-D REFERENCE MANUAL

Examples:

```
(PACKFILENAME.STRING 'DIRECTORY "LISP"
  'NAME "NET")
=> "<LISP>NET"

(PACKFILENAME.STRING 'NAME "NET"
  'DIRECTORY "{DSK}<LISPFILES>")
=> "{DSK}<LISPFILES>NET"

(PACKFILENAME.STRING 'DIRECTORY "{DSK}"
  'BODY "{TOAST}<FOO>BAR")
=> "{DSK}BAR"

(PACKFILENAME.STRING 'DIRECTORY "FRED"
  'BODY "{TOAST}<FOO>BAR")
=> "{TOAST}<FRED>BAR"

(PACKFILENAME.STRING 'BODY "{TOAST}<FOO>BAR"
  'DIRECTORY "FRED")
=> "{TOAST}<FOO>BAR"

(PACKFILENAME.STRING 'VERSION NIL
  'BODY "{TOAST}<FOO>BAR.DCOM;2")
=> "{TOAST}<FOO>BAR.DCOM"

(PACKFILENAME.STRING 'BODY "{TOAST}<FOO>BAR.DCOM"
  'VERSION 1)
=> "{TOAST}<FOO>BAR.DCOM;1"

(PACKFILENAME.STRING 'BODY "{TOAST}<FOO>BAR.DCOM;"
  'VERSION 1)
=> "{TOAST}<FOO>BAR.DCOM;"

(PACKFILENAME.STRING 'BODY "BAR.;1"
  'EXTENSION "DCOM")
=> "BAR.;1"

(PACKFILENAME.STRING 'BODY "BAR;1"
  'EXTENSION "DCOM")
=> "BAR.DCOM;1"
```

In the last two examples, note that in one case the extension is explicitly present in the body (as indicated by the preceding period), while in the other there is no indication of an extension, so the default is used.

(PACKFILENAME *FIELD CONTENTS* ... *FIELD CONTENTS*) [NoSpread Function]

The same as PACKFILENAME.STRING, except that it returns the file name as a symbol, instead of a string.

Incomplete File Names

In general, it is not necessary to pass a complete file name (one containing all the fields listed above) to functions that take a file name as argument. Interlisp supplies suitable defaults for certain fields, as described below. Functions that return names of actual files, however, always return the fully specified name.

STREAMS & FILES

If the version field is omitted from a file name, Interlisp performs version recognition, as described below.

If the host, device and/or directory field are omitted from a file name, Interlisp defaults them with respect to the currently connected directory. The connected directory is changed by calling the function CNDIR or using the programmer's assistant command CONN.

Defaults are added to the partially specified name "left to right" until a host, device or directory field is encountered. Thus, if the connected directory is {TWENTY}PS:<FRED>, then

```
BAR.DCOM means
{TWENTY}PS:<FRED>BAR.DCOM
<GRANOLA>BAR.DCOM means
{TWENTY}PS:<GRANOLA>BAR.DCOM
MTA0:<GRANOLA>BAR.DCOM means
{TWENTY}MTA0:<GRANOLA>BAR.DCOM
{THIRTY}<GRANOLA>BAR.DCOM means
{THIRTY}<GRANOLA>BAR.DCOM
```

In addition, if the partially specified name contains a subdirectory, but no principal directory, then the subdirectory is appended to the connected directory. For example,

```
ISO>BAR.DCOM means
{TWENTY}PS:<FRED>ISO>BAR.DCOM
```

Or, if the connected directory is the Unix directory {UNIX}/usr/fred/, then iso/bar.dcom means {UNIX}/usr/fred/iso/bar.dcom, but /other/bar.dcom means {UNIX}/other/bar.dcom.

(CNDIR *HOST/DIR*)

[Function]

Connects to the directory *HOST/DIR*, which can either be a directory name or a full directory specification including host and/or device. If the specification includes just a host, and the host supports directories, the directory is defaulted to the value of (USERNAME); if the host is omitted, connection is made to another directory on the same host as before. If *HOST/DIR* is NIL, connects to the value of LOGINHOST/DIR.

CNDIR returns the full name of the now-connected directory. Causes an error, Non-existent directory, if *HOST/DIR* is not recognized as a valid directory.

Note that CNDIR does not necessarily require or provide any directory access privileges. Access privileges are checked when a file is opened.

CONN *HOST/DIR*

[Prog. Asst. Command]

Convenient command form of CNDIR for use at the executive. Connects to *HOST/DIR*, or to the value of LOGINHOST/DIR if *HOST/DIR* is omitted. This command is undoable—undoing it causes the system to connect to the previously connected directory.

INTERLISP-D REFERENCE MANUAL

LOGINHOST/DIR [Variable]

CONN with no argument connects to the value of the variable LOGINHOST/DIR, initially {DSK}, but usually reset in your greeting file (Chapter 12).

(DIRECTORYNAME DIRNAME STRPTR) [Function]

If *DIRNAME* is T, returns the full specification of the currently connected directory. If *DIRNAME* is NIL, returns the "login" directory specification (the value of LOGINHOST/DIR). For any other value of *DIRNAME*, returns a full directory specification if *DIRNAME* designates an existing directory (satisfies DIRECTORYNAMEP), otherwise NIL.

If *STRPTR* is T, the value is returned as an atom, otherwise it is returned as a string.

(DIRECTORYNAMEP DIRNAME HOSTNAME) [Function]

Returns T if *DIRNAME* is recognized as a valid directory on host *HOSTNAME*, or on the host of the currently connected directory if *HOSTNAME* is NIL. *DIRNAME* may be either a directory name or a full directory specification containing host and/or device as well.

If *DIRNAME* includes subdirectories, this function may or may not pass judgment on their validity. Some hosts support "true" subdirectories, distinct entities manipulable by the file system, while others only provide them as a syntactic convenience.

(HOSTNAMEP NAME) [Function]

Returns T if *NAME* is recognized as a valid host or file device name at the moment HOSTNAMEP is called.

Version Recognition

Most of the file devices in Interlisp support file version numbers. That is, it is possible to have several files of the exact same name, differing only in their VERSION field, which is incremented for each new "version" of the file that is created. When a file name lacking a version number is presented to the file system, it is necessary to determine which version number is intended. This process is known as *version recognition*.

When OPENSTREAM opens a file for input and no version number is given, the highest existing version number is used. Similarly, when a file is opened for output and no version number is given, a new file is created with a version number one higher than the highest one currently in use with that file name. The version number defaulting for OPENSTREAM can be changed by specifying a different value for its RECOG argument, as described under FULLNAME, below.

Other functions that accept file names as arguments generally perform the default version recognition, which is newest version for existing files, or a new version if using the file name to create a new file. The one exception is DELFILE, which defaults to the oldest existing version of the file.

STREAMS & FILES

The functions below can be used to perform version recognition without actually calling `OPENSTREAM` to open the file. Note that these functions only tell the truth about the moment at which they are called, and thus cannot in general be used to anticipate the name of the file opened by a comparable `OPENSTREAM`. They are sometimes, however, helpful hints.

(FULLNAME *X* *RECOG*) [Function]

If *X* is an open stream, simply returns the full file name of the stream. Otherwise, if *X* is a file name given as a string or symbol, performs version recognition, as follows:

If *X* is recognized in the recognition mode specified by *RECOG* as an abbreviation for some file, returns the file's full name, otherwise `NIL`. *RECOG* is one of the following:

- OLD Choose the newest existing version of the file. Return `NIL` if no file named *X* exists.
- OLDEST Choose the oldest existing version of the file. Return `NIL` if no file named *X* exists.
- NEW Choose a new (not yet existing) version of the file. That is, if versions of *X* already exist, then choose a version number one higher than highest existing version; else choose version 1. For some file systems, `FULLNAME` returns `NIL` if you do not have the access rights necessary for creating a new file named *X*.
- OLD/NEW Try `OLD`, then `NEW`. That is, choose the newest existing version of the file, if any; else choose version 1. This usually only makes sense if you are intending to open *X* for access `BOTH`.

RECOG = `NIL` defaults to `OLD`. For all other values of *RECOG*, generates an error `ILLEGAL ARG`.

If *X* already contains a version number, the *RECOG* argument will never change it. In particular, *RECOG* = `NEW` does not require that the file actually be new. For example, `(FULLNAME 'FOO. ;2 'NEW)` may return `{ERIS}<LISP>FOO. ;2` if that file already exists, even though `(FULLNAME 'FOO 'NEW)` would default the version to a new number, perhaps returning `{ERIS}<LISP>FOO. ;5`.

(INFILEP *FILE*) [Function]

Equivalent to `(FULLNAME FILE 'OLD)`. That is, returns the full file name of the newest version of *FILE* if *FILE* is recognized as specifying the name of an existing file that could potentially be opened for input, `NIL` otherwise.

INTERLISP-D REFERENCE MANUAL

(OUTFILEP *FILE*)

[Function]

Equivalent to (FULLNAME *FILE* 'NEW).

Note that INFILEP, OUTFILEP and FULLNAME do not open any files; they are pure predicates. In general they are also only hints, as they do not necessarily imply that the caller has access rights to the file. For example, INFILEP might return non-NIL, but OPENSTREAM might fail for the same file because the file is read-protected against you, or the file happens to be open for output by another user at the time. Similarly, OUTFILEP could return non-NIL, but OPENSTREAM could fail with a FILE SYSTEM RESOURCES EXCEEDED error.

Note also that in a shared file system, such as a remote file server, intervening file operations by another user could contradict the information returned by recognition. For example, a file that was INFILEP might be deleted, or between an OUTFILEP and the subsequent OPENSTREAM, another user might create a new version or delete the highest version, causing OPENSTREAM to open a different version of the file than the one returned by OUTFILEP. In addition, some file servers do not well support recognition of files in output context. Thus, in general, the "truth" about a file can only be obtained by actually opening the file; creators of files should rely on the name of the stream opened by OPENSTREAM, not the value returned from these recognition functions. In particular, for the reasons described earlier, programmers are discouraged from using OUTFILEP or (FULLNAME *NAME* 'NEW).

Using File Names Instead of Streams

In earlier implementations of Interlisp, from the days of Interlisp-10 onward, the "handle" used to refer to an open file was not a stream, but rather the file's full name, represented as a symbol. When the file name was passed to any I/O function, it was mapped to a stream by looking it up in a list of open files. This scheme was sometimes convenient for typing in file commands at the executive, but was very poor for serious programming in two major ways. First, the mapping from file name to stream on every input/output operation is inefficient. Second, and more importantly, using the file name as the handle on an open stream means that it is not possible to have more than one stream open on a given file at once.

As of this writing, Medley is in a transition period, where it still supports the use of symbol file names as synonymous with open streams, but this use is not recommended. The remainder of this section discusses this usage of file names for the benefit of those reading older programs and wishing to convert them as necessary to work properly when this compatibility feature is removed.

File Name Efficiency Considerations

It is possible for a program to be seriously inefficient using a file name as a stream if the program is not using the file's full name, the name returned by OPENFILE (below). Any time that an input/output function is called with a file name other than the full file name, Interlisp must perform recognition on the partial file name in order to determine which open file is intended. Thus if repeated operations are to be performed, it is considerably more efficient to use the full file name

STREAMS & FILES

returned from `OPENFILE` than to repeatedly use the possibly incomplete name that was used to open the file.

There is a more subtle problem with partial file names, in that recognition is performed on your entire directory, not just the open files. It is possible for a file name that was previously recognized to denote one file to suddenly denote a different file. For example, suppose a program performs `(INFILE 'FOO)`, opening `FOO. ; 1`, and reads several expressions from `FOO`. Then you interrupt the program, create a `FOO. ; 2` and resume the program (or a user at another workstation creates a `FOO. ; 2`). Now a call to `READ` giving it `FOO` as its *FILE* argument will generate a `FILE NOT OPEN` error, because `FOO` will be recognized as `FOO. ; 2`.

Obsolete File Opening Functions

The following functions are now considered obsolete, but are provided for backwards compatibility:

`(OPENFILE FILE ACCESS RECOG PARAMETERS)` [Function]

Opens *FILE* with access rights as specified by *ACCESS*, and recognition mode *RECOG*, and returns the full name of the resulting stream. Equivalent to `(FULLNAME (OPENSTREAM FILE ACCESS RECOG PARAMETERS))`.

`(INFILE FILE)` [Function]

Opens *FILE* for input, and sets it as the primary input stream. Equivalent to `(INPUT (OPENSTREAM FILE 'INPUT 'OLD))`

`(OUTFILE FILE)` [Function]

Opens *FILE* for output, and sets it as the primary output stream. Equivalent to `(OUTPUT (OPENSTREAM FILE 'OUTPUT 'NEW))`.

`(IOFILE FILE)` [Function]

Equivalent to `(OPENFILE FILE 'BOTH 'OLD)`; opens *FILE* for both input and output. Does not affect the primary input or output stream.

Converting Old Programs

At some point in the future, the Medley file system will change so that each call to `OPENSTREAM` returns a distinct stream, even if a stream is already open to the specified file. This change is required in order to deal rationally with files in a multiprocessing environment.

This change will of necessity produce the following incompatibilities:

1. The functions `OPENFILE`, `INPUT`, and `OUTPUT` will return a `STREAM`, not a full file name. To make this less confusing in interactive situations, `STREAMs` will have a print format that reveals the underlying file's actual name,

INTERLISP-D REFERENCE MANUAL

2. A greater penalty will ensue for passing as the *FILE* argument to I/O operations anything other than the object returned from `OPENFILE`. Passing the file's name will be significantly slower than passing the stream (even when passing the "full" file name), and in the case where there is more than one stream open on the file it might even act on the wrong one.
3. `OPENP` will return `NIL` when passed the name of a file rather than a stream (the value of `OPENFILE` or `OPENSTREAM`).

Users should consider the following advice when writing new programs and editing existing programs, in order that they will continue to operate well when this change is made:

Because of the efficiency and ambiguity considerations described earlier, users have long been encouraged to use only full file names as *FILE* arguments to I/O operations. The "proper" way to have done this was to bind a variable to the value returned from `OPENFILE` and pass that variable to all I/O operations; such code will continue to work. A less proper way to obtain the full file name, but one which has to date not incurred any obvious penalty, is that which binds a variable to the result of an `INFILEP` and passes that to `OPENFILE` and all I/O operations. This has worked because `INFILEP` and `OPENFILE` both return a full file name, an invalid assumption in this future world. Such code should be changed to pass around the value of the `OPENFILE`, not the `INFILEP`.

Code that calls `OPENP` to test whether a possibly incomplete file name is already open should be recoded to pass to `OPENP` only the value returned from `OPENFILE` or `OPENSTREAM`.

Code that uses ordinary string functions to manipulate file names, and in particular the value returned from `OPENFILE`, should be changed to use the the functions `UNPACKFILENAME.STRING` and `PACKFILENAME.STRING`. Those functions work both on file names (strings) and streams (coercing the stream to the name of its file).

Code that tests the value of `OUTPUT` for equality to some known file name or `T` should be examined carefully and, if possible, recoded.

To see more directly the effects of passing around `STREAMs` instead of file names, replace your calls to `OPENFILE` with calls to `OPENSTREAM`. `OPENSTREAM` is called in exactly the same way, but returns a `STREAM`. Streams can be passed to `READ`, `PRINT`, `CLOSEF`, etc just as the file's full name can be currently, but using them is more efficient. The function `FULLNAME`, when applied to a stream, returns its full file name.

Using Files with Processes

Because Medley does not yet support multiple streams per file, problems can arise if different processes attempt to access the same file. You have to be careful not to have two processes manipulating the same file at the same time, since the two processes will be sharing a single input

stream and file pointer. For example, it will not work to have one process TCOMPL a file while another process is running LISTFILES on it.

File Attributes

Any file has a number of "file attributes", such as the read date, protection, and bytesize. The exact attributes that a file can have is dependent on the file device. The functions GETFILEINFO and SETFILEINFO allow you to conveniently access file attributes:

(GETFILEINFO *FILE ATTRIB*) [Function]

Returns the current setting of the *ATTRIB* attribute of *FILE*.

(SETFILEINFO *FILE ATTRIB VALUE*) [Function]

Sets the attribute *ATTRIB* of *FILE* to be *VALUE*. SETFILEINFO returns T if it is able to change the attribute *ATTRIB*, and NIL if unsuccessful, either because the file device does not recognize *ATTRIB* or because the file device does not permit the attribute to be modified.

The *FILE* argument to GETFILEINFO and SETFILEINFO can be an open stream (or an argument designating an open stream, see Chapter 25), or the name of a closed file. SETFILEINFO in general requires write access to the file.

The attributes recognized by GETFILEINFO and SETFILEINFO fall into two categories: *permanent* attributes, which are properties of the file, and *temporary* attributes, which are properties only of an open stream to the file. The temporary attributes are only recognized when *FILE* designates an open stream; the permanent attributes are usually equally accessible for open and closed files. However, some devices are willing to change the value of certain attributes of an open stream only when specified in the *PARAMETERS* argument to OPENSTREAM (see above), not on a later call to SETFILEINFO.

The following are currently recognized as permanent attributes of a file:

BYTESIZE	The byte size of the file. Medley currently only supports byte size 8.
LENGTH	The number of bytes in the file. Alternatively, the byte position of the end-of-file. Like (GETEOFPTR <i>FILE</i>), but <i>FILE</i> does not have to be open.
SIZE	The size of <i>FILE</i> in pages.
CREATIONDATE	The date and time, as a string, that the content of <i>FILE</i> was "created". The creation date changes whenever the content of the file is modified, but remains unchanged when a file is transported,

INTERLISP-D REFERENCE MANUAL

unmodified, across file systems. Specifically, COPYFILE and RENAMEFILE (see below) preserve the file's creation date. Note that this is different from the concept of "creation date" used by some operating systems (e.g., Tops20).

WRITEDATE The date and time, as a string, that the content of *FILE* was last written to this particular file system. When a file is copied, its creation date does not change, but its write date becomes the time at which the copy is made.

READDATE The date and time, as a string, that *FILE* was last read, or NIL if it has never been read.

ICREATIONDATE

IWRITEDATE

IREADDATE

The CREATIONDATE, WRITEDATE and READDATE, respectively, in integer form, as IDATE (Chapter 12) would return. This form is useful for comparing dates.

AUTHOR The name of the user who last wrote the file.

TYPE The "type" of the file, some indication of the nature of the file's content. The "types" of files allowed depends on the file device. Most devices recognize the symbol TEXT to mean that the file contains just characters, or BINARY to mean that the file contains arbitrary data.

Some devices support a wider range of file types that distinguish among the various sorts of files one might create whose content is "binary". All devices interpret any value of TYPE that they do not support to be BINARY. Thus, GETFILEINFO may return the more general value BINARY instead of the original type that was passed to SETFILEINFO or OPENSTREAM. Similarly, COPYFILE, while attempting to preserve the TYPE of the file it is copying, may turn, say, an INTERPRESS file into a mere BINARY file.

The way in which some file devices (e.g., Xerox file servers) support a wide range of file types is by representing the type as an integer, whose interpretation is known by the client. The variable FILING.TYPES is used to associate symbolic types with numbers for these devices. This list initially contains some of the well-known assignments of

STREAMS & FILES

type name to number; you can add additional elements to handle any private file types. For example, suppose there existed an NS file type `MAZEFILE` with numeric value 5678. You could add the element `(MAZEFILE 5678)` to `FILING.TYPES` and then use `MAZEFILE` as a value for the `TYPE` attribute to `SETFILEINFO` or `OPENSTREAM`. Other devices are, of course, free to store `TYPE` attributes in whatever manner they wish, be it numeric or symbolic. `FILING.TYPES` is merely considered the official registry for Xerox file types.

For most file devices, the `TYPE` of a newly created file, if not specified in the `PARAMETERS` argument to `OPENSTREAM`, defaults to the value of `DEFAULTFILETYPE`, initially `TEXT`.

The following are currently recognized as temporary attributes of an open stream:

ACCESS	The current access rights of the stream (see the beginning of this chapter). Can be one of <code>INPUT</code> , <code>OUTPUT</code> , <code>BOTH</code> , <code>APPEND</code> ; or <code>NIL</code> if the stream is not open.
ENDOFSTREAMOP	The action to be taken when a stream is at "end of file" and an attempt is made to take input from it. The value of this attribute is a function of one argument, the stream. The function can examine the stream and its calling context and take any action it wishes. If the function returns normally, it should return either <code>T</code> , meaning to try the input operation again, or the byte that <code>BIN</code> would have returned had there been more bytes to read. Ordinarily, one should not let the <code>ENDOFSTREAMOP</code> function return unless one is only performing binary input from the file, since there is no way in general of knowing in what state the reader was at the time the end of file occurred, and hence how it will interpret a single byte returned to it.
The default <code>ENDOFSTREAMOP</code> is a system function that causes the error <code>END OF FILE</code> . The behavior of that error can be further modified for a particular stream by using the <code>EOF</code> option of <code>WHENCLOSE</code> (see below).	
EOL	The end-of-line convention for the stream. This can be <code>CR</code> , <code>LF</code> , or <code>CRLF</code> , indicating with what byte or sequence of bytes the "End Of Line" character is

represented on the stream. On input, that sequence of bytes on the stream is read as (CHARCODE EOL) by READCCODE or the string reader. On output, (TERPRI) and (PRINTCCODE (CHARCODE EOL)) cause that sequence of bytes to be placed on the stream.

The end of line convention is usually not apparent to you. The file system is usually aware of the convention used by a particular remote operating system, and sets this attribute accordingly. If you believe a file actually is stored with a different convention than the default, it is possible to modify the default behavior by including the EOL attribute in the *PARAMETERS* argument to OPENSTREAM.

BUFFERS Value is the number of 512-byte buffers that the stream maintains at one time. This attribute is only used by certain random-access devices (currently, the local disk, floppy, and Leaf servers); all others ignore it.

Streams open to files generally maintain some portion of the file buffered in memory, so that each call to an I/O function does not require accessing the actual file on disk or a file server. For files being read or written sequentially, not much buffer space is needed, since once a byte is read or written, it will never need to be seen again. In the case of random access streams, buffering is more complicated, since a program may jump around in the file, using SETFILEPTR (Chapter 25). In this case, the more buffer space the stream has, the more likely it is that after a SETFILEPTR to a place in the file that has already been accessed, the stream still has that part of the file buffered and need not go out to the device again. This benefit must, of course, be traded off against the amount of memory consumed by the buffers.

Closing and Reopening Files

The function WHENCLOSE permits you to associate certain operations with open streams that govern how and when the stream will be closed. You can specify that certain functions will be executed before CLOSEF closes the stream and/or after CLOSEF closes the stream. You can make a particular stream be invisible to CLOSEALL, so that it will remain open across user invocations of CLOSEALL.

STREAMS & FILES

(WHENCLOSE *FILE* PROP VAL ... PROP VAL)

[NoSpread Function]

FILE must designate an open stream other than T (NIL defaults to the primary input stream, if other than T, or primary output stream if other than T). The remaining arguments specify properties to be associated with the full name of *FILE*. WHENCLOSE returns the full name of *FILE* as its value.

WHENCLOSE recognizes the following property names:

- BEFORE VAL is a function that CLOSEF will apply to the stream just before it is closed. This might be used, for example, to copy information about the file from an in-core data structure to the file just before it is closed.
- AFTER VAL is a function that CLOSEF will apply to the stream just after it is closed. This capability permits in-core data structures that know about the stream to be cleaned up when the stream is closed.
- CLOSEALL VAL is either YES or NO and determines whether *FILE* will be closed by CLOSEALL (YES) or whether CLOSEALL will ignore it (NO). CLOSEALL uses CLOSEF, so that any AFTER functions will be executed if the stream is in fact closed. Files are initialized with CLOSEALL set to YES.
- EOF VAL is a function that will be applied to the stream when an end-of-file error occurs, and the ERRORTYPELST entry for that error, if any, returns NIL. The function can examine the context of the error, and can decide whether to close the stream, RETFROM some function, or perform some other computation. If the function supplied returns normally (i.e., does not RETFROM some function), the normal error machinery will be invoked.

The default EOF behavior, unless overridden by this WHENCLOSE option, is to call the value of DEFAULTEOFDCLOSE (below).

For some applications, the ENDOFSTREAMOP attribute (see above) is a more useful way to intercept the end-of-file error. The ENDOFSTREAMOP attribute comes into effect before the error machinery is ever activated.

Multiple AFTER and BEFORE functions may be associated with a file; they are executed in sequence with the most recently associated function executed first. The CLOSEALL and EOF values, however, will

INTERLISP-D REFERENCE MANUAL

override earlier values, so only the last value specified will have an effect.

DEFAULTEOFCLOSE

[Variable]

Value is the name of a function that is called by default when an end of file error occurs and no EOF option has been specified for the stream by WHENCLOSE. The initial value of DEFAULTEOFCLOSE is NIL, meaning take no special action (go ahead and cause the error). Setting it to CLOSEF would cause the stream to be closed before the rest of the error machinery is invoked.

Local Hard Disk Device

Warning: This section describes the Medley functions that control the local hard disk drive available on some computers. All of these functions may not work on all computers running Medley. For more information on using the local hard disk facilities, see the users guide for your computer.

This section describes the local file system currently supported on the Xerox 1108 and 1186 computers. The Xerox 1132 supports a simpler local file system. The functions below are no-ops on the Xerox 1132, except for DISKPARTITION (which returns a disk partition number), and DISKFREEPAGES. On the Xerox 1132, different numbered partitions are referenced by using devices such as {DSK1}, {DSK2}, etc. {DSK} always refers to the disk partition that Interlisp is running on. The 1132 local file system does not support the use of directories.

The hard disk used with the Xerox 1108 or 1186 may be partitioned into a number of named "logical volumes." Logical volumes may be used to hold the Interlisp virtual memory file (see Chapter 12), or Interlisp files. For information on initializing and partitioning the hard disk, see the users guide for your computer. In order to store Interlisp files on a logical volume, it is necessary to create a lisp file directory on that volume (see CREATEDSKDIRECTORY, below).

So long as there exists a logical volume with a Lisp directory on it, files on this volume can be accessed by using the file device called {DSK}. Medley can be used to read, write, and otherwise interact with files on local disk disks through standard Interlisp input/output functions. All I/O functions such as LOAD, OPENSTREAM, READ, PRINT, GETFILEINFO, COPYFILE, etc., work with files on the local disk.

If you do not have a logical volume with a Lisp directory on it, Interlisp emulates the {DSK} device by a core device, a file device whose backing store is entirely within the Lisp virtual memory. However, this is not recommended because the core device only provides limited scratch space, and since the core device is contained in virtual memory, it (and the files stored on it) will be erased when the virtual memory file is reloaded.

Each logical volume with a Lisp directory on it serves as a directory of the device {DSK}. Files are referred to by forms such as

STREAMS & FILES

`{DSK}<VOLUMENAME>FILENAME`

Thus, the file `INIT.LISP` on the volume `LISPFILES` would be called `{DSK}<LISPFILES>INIT.LISP`.

Subdirectories within a logical volume are supported, using the `>` character in file names to delimit subdirectory names. For example, the file name `{DSK}<LISPFILES>DOC>DESIGN.TEDIT` designates the file names `DESIGN.TEDIT` on the subdirectory `DOC` on the logical volume `LISPFILES`.

If a logical volume name is not specified, it defaults in an unusual but simple way: the logical volume defaults to the next logical volume that has a lisp file directory on it including or after the volume containing the currently running virtual memory. For example, if the local disk has the logical volumes `LISP`, `TEMP`, and `LISPFILES`, the `LISP` volume contains the running virtual memory, and only the `LISP` volume has a Lisp file directory on it, then `{DSK}INIT.LISP` refers to the file `{DSK}<LispFiles>INIT.LISP`. All the functions below default logical volume names in a similar way, except for those such as `CREATEDSKDIRECTORY`. To determine the current default lisp file directory, evaluate `(DIRECTORYNAME ' {DSK})`.

`(CREATEDSKDIRECTORY VOLUMENAME)` [Function]

Creates a Lisp file directory on the logical volume `VOLUMENAME`, and returns the name of the directory created. It is only necessary to create a Lisp file directory the first time the logical volume is used. After that, the system automatically recognizes and opens access to the logical volumes that have Lisp file directories on them.

`(PURGEDSKDIRECTORY VOLUMENAME)` [Function]

Erases all Lisp files on the volume `VOLUMENAME`, and deletes the Lisp file directory.

`(LISPDIRECTORYP VOLUMENAME)` [Function]

Returns `T` if the logical volume `VOLUMENAME` has a lisp file directory on it.

`(VOLUMES)` [Function]

Returns a list of the names of all of the logical volumes on the local hard disk (whether they have lisp file directories or not).

`(VOLUMESIZE VOLUMENAME)` [Function]

Returns the total size of the logical volume `VOLUMENAME` in disk pages.

`(DISKFREEPAGES VOLUMENAME)` [Function]

Returns the total number of free disk pages left on the logical volume `VOLUMENAME`.

`(DISKPARTITION)` [Function]

Returns the name of the logical volume containing the virtual memory file that Interlisp is currently running in (see Chapter 12).

INTERLISP-D REFERENCE MANUAL

(DSKDISPLAY *NEWSTATE*)

[Function]

Controls a display window that displays information about the logical volumes on the local hard disk (logical volume names, sizes, free pages, etc.). DSKDISPLAY opens or closes this display window depending on the value of *NEWSTATE* (one of ON, OFF, or CLOSED), and returns the previous state of the display window.

If *NEWSTATE* is ON, the display window is opened, and it is automatically updated whenever the file system state changes (this can slow file operations significantly). If *NEWSTATE* is OFF, the display window is opened, but it is not automatically updated. If *NEWSTATE* is CLOSED, the display window is closed. The display mode is initially set to CLOSED.

Once the display window is open, you can update it or change its state with the mouse. Left-buttoning the display window updates it, and middle-buttoning the window brings up a menu that allows you to change the display state.

Note: DSKDISPLAY uses the value of the variable DSKDISPLAY.POSITION for the position of the lower-left corner of the disk display window when it is opened. This variable is changed if the disk display window is moved.

(SCAVENGEDSKDIRECTORY *VOLUMENAME SILENT*)

[Function]

Rebuilds the lisp file directory for the logical volume *VOLUMENAME*. This may repair damage in the unlikely event of file system failure, signified by symptoms such as infinite looping or other strange behavior while the system is doing a directory search. Calling SCAVENGEDSKDIRECTORY will not harm an intact volume.

Normally, SCAVENGEDSKDIRECTORY prints out messages as it scavenges the directory. If *SILENT* is non-NIL, these messages are not printed.

Note: Some low-level disk failures may cause "HARD DISK ERROR" errors to occur. To fix such a failure, it may be necessary to log out of Interlisp, scavenge the logical volume in question using Pilot tools, and then call SCAVENGEDSKDIRECTORY from within Interlisp. See the users guide for your computer for more information.

Floppy Disk Device

Warning: This section describes the Medley functions that control the floppy disk drive available on some computers. All of these functions may not work on all computers running Medley. For more information on using the floppy disk facilities, see the users guide for your computer.

The floppy disk drive is accessed through the device {FLOPPY}. Medley can be used to read, write, and otherwise interact with files on floppy disks through standard Interlisp input/output functions. All I/O functions such as LOAD, OPENSTREAM, READ, PRINT, GETFILEINFO, COPYFILE, etc., work with files on floppies.

STREAMS & FILES

Note that floppy disks are a removable storage medium. Therefore, it is only meaningful to perform I/O operations to the floppy disk drive, rather than to a given floppy disk. In this section, the phrase "the floppy" is used to mean "the floppy that is currently in the floppy disk drive."

For example, the following sequence could be used to open a file XXX.TXT on the floppy, print "Hello" on it, and close it:

```
(SETQ XXX (OPENSTREAM '{FLOPPY}XXX.TXT 'OUTPUT 'NEW)
  (PRINT "Hello" XXX)
  (CLOSEF XXX)
```

(FLOPPY.MODE *MODE*)

[Function]

Medley can currently read and write files on floppies stored in a number of different formats. At any point, the floppy is considered to be in one of four "modes," which determines how it reads and writes files on the floppy. FLOPPY.MODE sets the floppy mode to the value of *MODE*, one of PILOT, HUGEPILOT, SYSOUT, or CPM, and returns the previous floppy mode. The floppy modes are interpreted as follows:

PILOT This is the normal floppy mode, using floppies in the Xerox Pilot floppy disk format. This file format allows all of the normal Medley I/O operations. This format also supports file names with arbitrary levels of subdirectories. For example, it is possible to create a file named {FLOPPY}<Lisp>Project>FOO.TXT.

HUGEPILOT This floppy mode is used to access files that are larger than a single floppy, stored on multiple floppies. There are some restrictions with using "huge" files. Some I/O operations are not meaningful for "huge" files. When a stream is created for output in this mode, the LENGTH file attribute must be specified when the file is opened, so that it is known how many floppies will be needed. When an output file is created, the floppy (or floppies) are automatically erased and reformatted (after confirmation from you).

HUGEPILOT mode is primarily useful for saving big files to and from floppies. For example, the following could be used to copy the file {ERIS}<Lisp>Bigfile.txt onto the huge Pilot file {FLOPPY}BigFile.save:

```
(FLOPPY.MODE 'HUGEPILOT)

(COPYFILE      '{ERIS}<Lisp>Bigfile.txt
 '{FLOPPY}BigFile.save)
```

INTERLISP-D REFERENCE MANUAL

and the following would restore the file:

```
(FLOPPY.MODE 'HUGEPILOT)

(COPYFILE      '{FLOPPY}BigFile.save
 '{ERIS}<Lisp>Bigfile.txt)
```

During each copying operation, you will be prompted to insert "the next floppy" if {ERIS}<Lisp>Bigfile.txt takes multiple floppies.

SYSOUT Similar to HUGEPILOT mode, SYSOUT mode is used for storing sysout files (Chapter 12) on multiple floppy disks. You are prompted to insert new floppies as they are needed.

This mode is set automatically when SYSOUT or MAKESYS is done to the floppy device: (SYSOUT '{FLOPPY}) or (MAKESYS '{FLOPPY}). Notice that the file name does not need to be specified in SYSOUT mode; unlike HUGEPILOT mode, the file name Lisp.sysout is always used.

Note: The procedure for loading sysout files from floppies depends on the particular computer being used. For information on loading sysout files from floppies, see the users guide for your computer.

Explicitly setting the mode to SYSOUT is useful when copying a sysout file to or from floppies. For example, the following can be used to copy the sysout file {ERIS}<Lisp>Lisp.sysout onto floppies (it is important to set the floppy mode back when done):

```
(FLOPPY.MODE 'SYSOUT)
(COPYFILE      '{ERIS}<Lisp>Lisp.sysout
 '{FLOPPY})
(FLOPPY.MODE 'PILOT)
```

CPM Medley supports the single-density single-sided (SDSS) CPM floppy format (a standard used by many computers). CPM-formatted floppies are totally different than Pilot floppies, so you should call FLOPPY.MODE to switch to CPM mode when planning to use CPM floppies. After switching to CPM mode, FLOPPY.FORMAT can be used to create CPM-formatted floppies, and the usual input/output operations work with CPM floppy files.

STREAMS & FILES

Note: There are a few limitations on CPM floppy format files: (1) CPM file names are limited to eight or fewer characters, with extensions of three or fewer characters; (2) CPM floppies do not have directories or version numbers; and (3) CPM files are padded out with blanks to make the file lengths multiples of 128.

(FLOPPY .FORMAT *NAME* *AUTOCONFIRMFLG* *SLOWFLG*) [Function]

FLOPPY .FORMAT erases and initializes the track information on a floppy disk. This must be done when new floppy disks are to be used for the first time. This can also be used to erase the information on used floppy disks.

NAME should be a string that is used as the name of the floppy (106 characters max). This name can be read and set using FLOPPY .NAME (below).

If *AUTOCONFIRMFLG* is NIL, you will be prompted to confirm erasing the floppy, if it appears to contain valid information. If *AUTOCONFIRMFLG* is T, you are not prompted to confirm.

If *SLOWFLG* is NIL, only the Pilot records needed to give your floppy an empty directory are written. If *SLOWFLG* is T, FLOPPY .FORMAT will completely erase the floppy, writing track information and critical Pilot records on it. *SLOWFLG* should be set to T when formatting a brand-new floppy.

Note: Formatting a floppy is a very compute-intensive operation for the I/O hardware. Therefore, the cursor may stop tracking the mouse and keystrokes may be lost while formatting a floppy. This behavior goes away when the formatting is finished.

Warning: The floppy mode set by FLOPPY .MODE (above) affects how FLOPPY .FORMAT formats the floppy. If the floppy is going to be used in Pilot mode, it should be formatted under (FLOPPY .MODE 'PILOT). If it is to be used as a CPM floppy, it should be formatted under (FLOPPY .MODE 'CPM). The two types of formatting are incompatible.

(FLOPPY .NAME *NAME*) [Function]

If *NAME* is NIL, returns the name stored on the floppy disk. If *NAME* is non-NIL, then the name of the floppy disk is set to *NAME*.

(FLOPPY .FREE .PAGES) [Function]

Returns the number of unallocated free pages on the floppy disk in the floppy disk drive.

Note: Pilot floppy files are represented by contiguous pages on a floppy disk. If you are creating and deleting a lot of files on a floppy, it is advisable to keep such a floppy less than 75 percent full.

(FLOPPY .CAN .READP) [Function]

Returns non-NIL if there is a floppy in the floppy drive.

INTERLISP-D REFERENCE MANUAL

Note: `FLOPPY.CAN.READP` does not provide any debouncing (protection against not fully closing the floppy drive door). It may be more useful to use `FLOPPY.WAIT.FOR.FLOPPY` (below).

(`FLOPPY.CAN.WRITEP`) [Function]

Returns non-NIL if there is a floppy in the floppy drive and the floppy drive can write on this floppy.

It is not possible to write on a floppy disk if the "write-protect notch" on the floppy disk is punched out.

(`FLOPPY.WAIT.FOR.FLOPPY NEWFLG`) [Function]

If `NEWFLG` is NIL, waits until a floppy is in the floppy drive before returning.

If `NEWFLG` is T, waits until the existing floppy in the floppy drive, if any, is removed, then waits for a floppy to be inserted into the drive before returning.

(`FLOPPY.SCAVENGE`) [Function]

Attempts to repair a floppy whose critical records have become confused (causing errors when file operations are attempted). May also retrieve accidentally-deleted files, provided they haven't been overwritten by new files.

(`FLOPPY.TO.FILE TOFILE`) [Function]

Copies the entire contents of the floppy to the "floppy image" file `TOFILE`, which can be on a file server, local disk, etc. This can be used to create a centralized copy of a floppy, that different users can copy to their own floppy disks (using `FLOPPY.FROM.FILE`).

Note: A floppy image file for an 8-inch floppy is about 2500 pages long, regardless of the number of pages in use on the floppy.

(`FLOPPY.FROM.FILE FROMFILE`) [Function]

Copies the "floppy image" file `FROMFILE` to the floppy. `FROMFILE` must be a file produced by `FLOPPY.TO.FILE`.

(`FLOPPY.ARCHIVE FILES NAME`) [Function]

`FLOPPY.ARCHIVE` formats a floppy inserted into the floppy drive, giving the floppy the name `NAME#1`. `FLOPPY.ARCHIVE` then copies each file in `FILES` to the freshly formatted floppy. If the first floppy fills up, `FLOPPY.ARCHIVE` uses multiple floppies (named `NAME#2`, `NAME#3`, etc.), each time prompting you to insert a new floppy.

The function `DIRECTORY` (see below) is convenient for generating a list of files to archive. For example,

```
(FLOPPY.ARCHIVE
  (DIRECTORY ' {ERIS} <Lisp>Project >*)
  ' Project)
```

STREAMS & FILES

will archive all files on the directory {ERIS}<Lisp>Project> to floppies (named Project#1, Project#2, etc.).

(FLOPPY.UNARCHIVE *HOST/DIRECTORY*)

[Function]

FLOPPY.UNARCHIVE copies all files on the current floppy to the directory *HOST/DIRECTORY*. For example, (FLOPPY.UNARCHIVE '{ERIS}<Lisp>Project>) will copy each file on the current floppy to the directory {ERIS}<Lisp>Project>. If there is more than one floppy to restore from archive, FLOPPY.UNARCHIVE should be called on each floppy disk.

I/O Operations to and from Strings

It is possible to treat a string as if it were the contents of a file by using the following function:

(OPENSTRINGSTREAM *STR ACCESS*)

[Function]

Returns a stream that can be used to access the characters of the string *STR*. *ACCESS* may be either INPUT, OUTPUT, or BOTH; NIL defaults to INPUT. The stream returned may be used exactly like a file opened with the same access, except that output operations may not extend past the end of the original string. Also, string streams do not appear in the value of (OPENP).

For example, after performing

```
(SETQ STRM (OPENSTRINGSTREAM "THIS 2 (IS A LIST)"))
```

the following succession of reads could occur:

```
(READ STRM) => THIS
(RATOM STRM) => 2
(READ STRM) => (IS A LIST)
(EOFP STRM) => T
```

Compatibility Note: In Interlisp-10 it was possible to take input from a string simply by passing the string as the *FILE* argument to an input function. In order to maintain compatibility with this feature, Medley provides the same capability. This not terribly clean feature persists in the present implementation to give users time to convert old code. This means that strings are *not* equivalent to symbols when specifying a file name as a stream argument. In a future release, the old Interlisp-10 string-reading feature will be decommissioned, and OPENSTRINGSTREAM will be the only way to perform I/O on a string.

Temporary Files and the CORE Device

Many operating systems have a notion of "scratch file", a file typically used as temporary storage for data most naturally maintained in the form of a file, rather than some other data structure. A scratch

INTERLISP-D REFERENCE MANUAL

file can be used as a normal file in most respects, but is automatically deleted from the file system after its useful life is up, e.g., when the job terminates, or you log out. In normal operation, you need never explicitly delete such files, since they are guaranteed to disappear soon.

A similar functionality is provided in Medley by core-resident files. Core-resident files are on the device CORE. The directory structure for this device and all files on it are represented completely within your virtual memory. These files are treated as ordinary files by all file operations; their only distinguishing feature is that all trace of them disappears when the virtual memory is abandoned.

Core files are opened and closed by name the same as any other file, e.g., (OPENSTREAM ' {CORE} <FOO>FIE.DCOM ' OUTPUT). Directory names are completely optional, so files can also have names of the form {CORE}NAME.EXT. Core files can be enumerated by DIRECTORY (see below). While open, they are registered in (OPENP). They do consume virtual memory space, which is only reclaimed when the file is deleted. Some caution should thus be used when creating large CORE files. Since the virtual memory of an Medley workstation usually persists far longer than the typical process on a mainframe computer, it is still important to delete CORE files after they are no longer in use.

For many applications, the name of the scratch file is irrelevant, and there is no need for anyone to have access to the file independent of the program that created it. For such applications, NODIRCORE files are preferable. Files created on the device lisp NODIRCORE are core-resident files that have no name and are registered in no directory. These files "disappear", and the resources they consume are reclaimed, when all pointers to the file are dropped. Hence, such files need never be explicitly deleted or, for that matter, closed. The "name" of such a file is simply the stream object returned from (OPENSTREAM ' {NODIRCORE} ' OUTPUT), and it is this stream object that must be passed to all input/output operations, including CLOSEP and any calls to OPENSTREAM to reopen the file.

(COREDEVICE NAME NODIRFLG)

[Function]

Creates a new device for core-resident files and assigns NAME as its device name. Thus, after performing (COREDEVICE ' FOO), one can execute (OPENSTREAM ' {FOO}BAR ' OUTPUT) to open a file on that device. Medley is initialized with the single core-resident device named CORE, but COREDEVICE may be used to create any number of logically distinct core devices.

If NODIRFLG is non-NIL, a core device that acts like {NODIRCORE} is created.

Compatibility note: In Interlisp-10, it was possible to create scratch files by using file names with suffixes ;S or ;T. In Medley, these suffixes in file names are simply ignored when output is directed to a particular host or device. However, the function PACKFILENAME.STRING is defined to default the device name to CORE if the file has the TEMPORARY attribute and no explicit host is provided.

NULL Device

The NULL device provides a source of content-free "files". (OPENSTREAM ' {NULL} 'OUTPUT) creates a stream that discards all output directed at it. (OPENSTREAM ' {NULL} 'INPUT) creates a stream that is perpetually at end-of-file (i.e., has no input).

Deleting, Copying, and Renaming Files(DELFILE *FILE*)

[Function]

Deletes *FILE* if possible. The file must be closed. Returns the full name of the file if deleted, else NIL. Recognition mode for *FILE* is OLDEST, i.e., if *FILE* does not have a version number specified, then DELFILE deletes the oldest version of the file.

(COPYFILE *FROMFILE TOFILE*)

[Function]

Copies *FROMFILE* to a new file named *TOFILE*. The source and destination may be on any combination of hosts/devices. COPYFILE attempts to preserve the TYPE and CREATIONDATE where possible. If the original file's file type is unknown, COPYFILE attempts to infer the type (file type is BINARY if any of its 8-bit bytes have their high bit on).

COPYFILE uses COPYCHARS (Chapter 25) if the source and destination hosts have different EOL conventions. Thus, it is possible for the source and destination files to be of different lengths.

(RENAMEFILE *OLDFILE NEWFILE*)

[Function]

Renames *OLDFILE* to be *NEWFILE*. Causes an error, FILE NOT FOUND if *FILE* does not exist. Returns the full name of the new file, if successful, else NIL if the rename cannot be performed.

If *OLDFILE* and *NEWFILE* are on the same host/device, and the device implements a renaming primitive, RENAMEFILE can be very fast. However, if the device does not know how to rename files in place, or if *OLDFILE* and *NEWFILE* are on different devices, RENAMEFILE works by copying *OLDFILE* to *NEWFILE* and then deleting *OLDFILE*.

Searching File Directories

DIRECTORIES

[Variable]

Global variable containing the list of directories searched (in order) by SPELLFILE and FINDFILE (below) when not given an explicit *DIRLIST* argument. In this list, the atom NIL stands for the login directory (the value of LOGINHOST/DIR), and the atom T stands for the currently connected directory. Other elements should be *full* directory specifications, e.g., {TWENTY}PS:<LISPUSERS>, not merely LISPUSERS.

INTERLISP-D REFERENCE MANUAL

LISPUSERSDIRECTORIES [Variable]

Global variable containing a list of directories to search for "library" package files. Used by the FILES file package command (Chapter 17).

(SPELLFILE *FILE* *NOPRINTFLG* *NSFLG* *DIRLST*) [Function]

Searches for the file name *FILE*, possibly performing spelling correction (see Chapter 20). Returns the corrected file name, if any, otherwise NIL.

If *FILE* has a directory field, SPELLFILE attempts spelling correction against the files in that particular directory. Otherwise, SPELLFILE searches for the file on the directory list *DIRLST* before attempting any spelling correction.

If *NOPRINTFLG* is NIL, SPELLFILE asks you to confirm any spelling correction done, and prints out any files found, even if spelling correction is not done. If *NOPRINTFLG* = T, SPELLFILE does not do any printing, nor ask for approval.

If *NSFLG* = T (or *NOSPELLFLG* = T, see Chapter 20), no spelling correction is attempted, though searching through *DIRLST* still occurs.

DIRLST is the list of directories searched if *FILE* does not have a directory field. If *DIRLST* is NIL, the value of the variable DIRECTORIES is used.

Note: If *DIRLST* is NIL, and *FILE* is not found by searching the directories on DIRECTORIES, but the root name of *FILE* has a FILEDATES property (Chapter 17) indicating that a file by that name has been loaded, then the directory indicated in the FILEDATES property is searched, too. This additional search is not done if *DIRLST* is non-NIL.

ERRORTYPELIST (Chapter 14) initially contains the entry ((23 (SPELLFILE (CADR ERRORMESS) NIL NOFILESPELLFLG))), which causes SPELLFILE to be called in case of a FILE NOT FOUND error. If the variable NOFILESPELLFLG is T (its initial value), then spelling correction is not done on the file name, but DIRECTORIES is still searched. If SPELLFILE is successful, the operation will be reexecuted with the new (corrected) file name.

(FINDFILE *FILE* *NSFLG* *DIRLST*) [Function]

Uses SPELLFILE to search for a file named *FILE*. If it finds one, returns its full name, with no user interaction. Specifically, it calls (SPELLFILE *FILE* T *NSFLG* *DIRLST*), after first performing two simple checks: If *FILE* has an explicit directory, it checks to see if a file so named exists, and if so returns that file. If *DIRLST* is NIL, it looks for *FILE* on the connected directory before calling SPELLFILE.

Listing File Directories

The function `DIRECTORY` allows you to conveniently specify and/or program a variety of directory operations:

(`DIRECTORY FILES COMMANDS DEFAULTTEXT DEFAULTVERS`) [Function]

Returns, lists, or performs arbitrary operations on all files specified by the "file group" *FILES*. A file group has the form of a regular file name, except that the character `*` can be used to match any number of characters, including zero, in the file name. For example, the file group *A*B* matches all file names beginning with the character *A* and ending with the character *B*. The file group **.DCOM* matches all files with an extension of *DCOM*.

If *FILES* does not contain an explicit extension, it is defaulted to *DEFAULTTEXT*; if *FILES* does not contain an explicit version, it is defaulted to *DEFAULTVERS*. *DEFAULTTEXT* and *DEFAULTVERS* themselves default to `*`. If the period or semicolon preceding the omitted extension or version, respectively, is present, the field is explicitly empty and no default is used. All other unspecified fields default to `*`. Null version is interpreted as "highest". Thus *FILES* = `*` or `*.*` or `*.*.*` enumerates all files on the connected directory; *FILES* = `*.` or `*.*` enumerates all versions of files with null extension; *FILES* = `*.*` enumerates the highest version of files with null extension; and *FILES* = `*.*.*` enumerates the highest version of all files. If *FILES* is *NIL*, it defaults to `*.*.*`.

Note: Some hosts/devices are not capable of supporting "highest version" in enumeration. Such hosts instead enumerate *all* versions.

For each file that matches the file group *FILES*, the "file commands" in *COMMANDS* are executed in order. Some of the file commands allow aborting the command processing for a given file, effectively filtering the list of files. The interpretation of the different file commands is described below. If *COMMANDS* is *NIL*, it defaults to (*COLLECT*), which collects the matching file names in a list and returns it as the value of `DIRECTORY`.

The "file commands" in *COMMANDS* are interpreted as follows:

P	Prints the file's name. For readability, <code>DIRECTORY</code> strips the directory from the name, printing it once as a header in front of each set of consecutive files on the same directory.
PP	Prints the file's name without a version number.
a string	Prints the string.
READDATE, WRITEDATE CREATIONDATE, SIZE LENGTH, BYTESIZE PROTECTION, AUTHOR	
TYPE	Prints the appropriate information returned by <code>GETFILEINFO</code> (see above).

INTERLISP-D REFERENCE MANUAL

COLLECT	Adds the full name of this file to an accumulating list, which will be returned as the value of DIRECTORY.
COUNTSIZE	Adds the size of this file to an accumulating sum, which will be returned as the value of DIRECTORY.
DELETE	Deletes the file.
DELVER	If this file is not the highest version of files by its name, delete it.
PAUSE	Waits until you type any character before proceeding with the rest of the commands (good for display if you want to ponder).

The following commands are predicates to filter the list. If the predicate is not satisfied, then processing for this file is aborted and no further commands (such as those above) are executed for this file.

Note: if the P and PP commands appear in *COMMANDS* ahead of any of the filtering commands below except PROMPT, they are postponed until after the filters. Thus, assuming the caller has placed the attribute options after the filters as well, no printing occurs for a file that is filtered out. This is principally so that functions like DIR (below) can both request printing and pass arbitrary commands through to DIRECTORY, and have the printing happen in the appropriate place.

PROMPT MESS	Prompts with the yes/no question <i>MESS</i> ; if user responds with No, abort command processing for this file.
OLDERTHAN <i>N</i>	Continue command processing if the file hasn't been referenced (read or written) in <i>N</i> days. <i>N</i> can also be a string naming an explicit date and time since which the file must not have been referenced.
NEWERTHAN <i>N</i>	Continue command processing if the file has been written within the last <i>N</i> days. <i>N</i> can also be a string naming an explicit date and time. Note that this is not quite the complement of OLDERTHAN, since it ignores the read date.
BY USER	Continue command processing if the file was last written by the given user, i.e., its AUTHOR attribute matches (case insensitively) <i>USER</i> .
@ <i>X</i>	<i>X</i> is either a function of one argument (<i>FILENAME</i>), or an arbitrary expression which uses the variable <i>FILENAME</i> freely. If <i>X</i> returns NIL, abort command processing for this file.

STREAMS & FILES

The following two commands apply not to any particular file, but globally to the manner in which directory information is printed.

`OUT FILE` Directs output to *FILE*.

`COLUMNS N` Attempts to format output in *N* columns (rather than just 1).

`DIRECTORY` uses the variable `DIRCOMMANDS` as a spelling list to correct spelling and define abbreviations and synonyms (see Chapter 20). Currently the following abbreviations are recognized:

<code>AU</code>	<code>=></code>	<code>AUTHOR</code>	
<code>-</code>	<code>=></code>	<code>PAUSE</code>	
<code>COLLECT?</code>	<code>=></code>	<code>PROMPT " ? " COLLECT</code>	
<code>DA</code>			
<code>DATE</code>	<code>=></code>	<code>CREATIONDATE</code>	
<code>TI</code>	<code>=></code>	<code>WRITEDATE</code>	
<code>DEL</code>	<code>=></code>	<code>DELETE</code>	
<code>DEL?</code>			
<code>DELETE?</code>	<code>=></code>	<code>PROMPT " delete? " DELETE</code>	
<code>OLD</code>	<code>=></code>	<code>OLDERTHAN 90</code>	
<code>PR</code>	<code>=></code>	<code>PROTECTION</code>	
<code>SI</code>	<code>=></code>	<code>SIZE</code>	
<code>VERBOSE</code>	<code>=></code>	<code>AUTHOR CREATIONDATE SIZE</code>	<code>READDATA</code>
		<code>TE WRITEDATE</code>	

`(FILDIR FILEGROUP)` [Function]

Obsolete synonym of `(DIRECTORY FILEGROUP)`.

`(DIR FILEGROUP COM ... COM)` [NLambda NoSpread Function]

Convenient form of `DIRECTORY` for use in type-in at the executive. Performs `(DIRECTORY 'FILEGROUP' (P COM ... COM))`.

`(NDIR FILEGROUP COM ... COM)` [NLambda NoSpread Function]

Version of `DIR` that lists the file names in a multi-column format. Also, by default only lists the most recent version of files (unless *FILEGROUP* contains an explicit version).

File Servers

A file server is a shared resource on a local communications network which provides large amounts of file storage. Different file servers honor a variety of access protocols. Medley supports the following

INTERLISP-D REFERENCE MANUAL

protocols: PUP-FTP, PUP-Leaf, and NS Filing. In addition, there are library packages available that support other communications protocols, such as TCP/IP and RS232.

With the exception of the RS232-based protocols, which exist only for file transfer, these network protocols are integrated into the Medley file system to allow files on a file server to be treated in much the same way files are accessed on local devices, such as the disk. Thus, it is possible to call `OPENSTREAM` on the file `{ERIS}<LISP>FOO.DCOM;3` and read from it or write to it just as if the file had been on the local disk (`{DSK}<LISP>FOO.DCOM;3`), rather than on a remote server named ERIS. However, the protocols vary in how much control they give the workstation over file system operations. Hence, some restrictions apply, as described in the following sections.

PUP File Server Protocols

There are two file server protocols in the family of PUP protocols: Leaf and FTP. Some servers support both, while others support only one of them. Medley uses whichever protocol is more appropriate for the requested operation.

Leaf is a random access protocol, so files opened using these protocols are `RANDACCESSP`, and thus most normal I/O operations can be performed. However, Leaf does not support directory enumeration. Hence, `DIRECTORY` cannot be used on a Leaf file server unless the server also supports FTP. In addition, Leaf does not supply easy access to a file's attributes. `INFILEP` and `GETFILEINFO` have to open the file for input in order to obtain their information, and hence the file's read date will change, even though the semantics of these functions do not imply it.

FTP is a file transfer protocol that only permits sequential access to files. However, most implementations of it are considerably more efficient than Leaf. Medley uses FTP in preference to Leaf whenever the call to `OPENSTREAM` requests sequential access only. In particular, the functions `SYSOUP` and `COPYFILE` open their files for sequential access. If a file server supports FTP but for some reason it is undesirable for Lisp to use it, one can set the internal variable `\FTPAVAILABLE` to `NIL`.

The system normally maintains a Leaf connection to a host in the background. This connection can be broken by calling `(BREAKCONNECTION HOST)`. Any subsequent reference to files on that host will re-establish the connection. The principal use for this function arises when you interrupt a file operation in such a way that the file server thinks the file is open but Lisp thinks it is closed (or not yet open). As a result, the next time Lisp tries to open the file, it gets a file busy error.

Xerox NS File Server Protocols

Interlisp supports file access to Xerox 803x file servers, using the Filing Protocol built on Xerox Network Systems protocols. Medley determines that a host is an NS File Server by the presence of a colon in its name, e.g., `{PHYLEX:}`. The general format of NS fileserver device names is `{SERVERNAME:DOMAIN:ORGANIZATION}`; the device specification for an 8000-series product in general includes the ClearingHouse domain and organization. If domain and organization are not supplied directly, then they are obtained from the defaults, which themselves are found by consulting

STREAMS & FILES

the nearest ClearingHouse if you have not defined them in an init file. However, note that the server name must still have a colon in it to distinguish it from other types of host names (e.g., PUP server names).

NS file servers in general permit arbitrary characters in file names. You should be cognizant of file name quoting conventions, and the fact that any file name presented as a symbol needs to have characters of significance to the reader, such as space, escaped with a %. Of course, one can always present the file name as a string, in which case only the quoting conventions are important.

NS file servers support a true hierarchical file system, where subdirectories are just another kind of file, which needs to be explicitly created. In Interlisp, subdirectories are created automatically as needed: A call to `OPENFILE` to create a file in a non-existent subdirectory automatically creates the subdirectory. `CONN` to a non-existent subdirectory asks you whether to create the directory. For those using Star software, a directory corresponds to a "File Drawer," while a subdirectory corresponds to a "File Folder."

Because of their hierarchical structure, NS directories can be enumerated to arbitrary levels. The default is to enumerate all the files (the leaves of the tree), omitting the subdirectory nodes themselves. This default can be changed by the following variable:

`FILING.ENUMERATION.DEPTH` [Variable]

This variable is either a number, specifying the number of levels deep to enumerate, or `T`, meaning enumerate to all levels. In the former case, when the enumeration reaches the specified depth, only the subdirectory name rooted at that level is listed, and none of its descendants is listed. When `FILING.ENUMERATION.DEPTH` is `T`, all files are listed, and no subdirectory names are listed. `FILING.ENUMERATION.DEPTH` is initially `T`.

Independent of `FILING.ENUMERATION.DEPTH`, a request to enumerate the top-level of a file server's hierarchy lists only the top level, i.e., assumes a depth of 1. For example, `(DIRECTORY ' { PHYLEX: } )` lists exactly the top-level directories of the server `PHYLEX:`.

NS file servers do not currently support random access. Therefore, `SETFILEPTR` of an NS file generally causes an error. However, `GETFILEPTR` returns the correct character position for open files on NS file servers. In addition, `SETFILEPTR` works in the special case where the file is open for input, and the file pointer is being set forward. In this case, the intervening characters are automatically read.

Even while Interlisp has no file open on an NS Server, the system maintains a "session" with the server for a while in order to improve the speed of subsequent requests to the server. While this session is open, it is possible for some nodes of the server's file system to appear "busy" or inaccessible to certain clients on other workstations (such as Star). If this happens, the following function can be used to terminate any open sessions immediately.

`(BREAK.NSFILING.CONNECTION HOST)` [Function]

Closes any open connections to NS file server *HOST*.

INTERLISP-D REFERENCE MANUAL

Operating System Designations

Some of the network server protocols are implemented on more than one kind of foreign host. Such hosts vary in their conventions for logging in, naming files, representing end-of-line, etc. In order for Interlisp to communicate gracefully with all these hosts, it is necessary that the variable `NETWORKOSTYPES` be set correctly. The following functions are now considered obsolete, but are provided for backwards compatibility:

`NETWORKOSTYPES` [Variable]

An association-list that associates a host name with its operating system type. Elements in this list are of the form `(HOSTNAME . TYPE)`. For example, `(MAXC2 . TENEX)`. The operating system types currently known to Lisp are TENEX, TOPS20, UNIX, and VMS. The host names in this list should be the "canonical" host name, represented as an uppercase atom. For PUP and NS hosts, the function `CANONICAL.HOSTNAME` (below) can be used to determine which of several aliases of a server is the canonical name.

`(CANONICAL.HOSTNAME HOSTNAME)` [Function]

Returns the "canonical" name of the server `HOSTNAME`, or `NIL` if `HOSTNAME` is not the name of a server.

Logging In

Most file servers require a user name and password for access. Medley maintains an ephemeral database of user names and passwords for each host accessed recently. The database vanishes when `LOGOUT`, `SAVEVM`, `SYSOUT`, or `MAKESYS` is executed, so that the passwords remain secure from any subsequent user of the same virtual memory image. Medley also maintains a notion of the "default" user name and password, which are generally those with which you initially log in.

When a file server for which the system does not yet have an entry in its password database requests a name and password, the system first tries the default user name and password. If the file server does not recognize that name/password, the system prompts you for a name and password to use for that host. It suggests a default name:

```
{ERIS} Login: Green
```

which you can accept by pressing [Return], or replace the name by typing a new name or backspacing over it. Following the name, you are prompted for a password:

```
{ERIS} Login: Verdi (password)
```

which is not echoed, terminated by another [Return]. This information is stored in the password database so that you are prompted only once, until the database is again cleared.

STREAMS & FILES

Medley also prompts for password information when a protection violation occurs on accessing a directory on certain kinds of servers that support password-protected directories. Some such servers allow one to protect a file in a way that is inaccessible to even its owner until the file's protection is changed. In such cases, no password would help, and the system causes the normal PROTECTION VIOLATION error.

You can abort a password interaction by typing the ERROR interrupt, initially Cosntrol-E. This generally either causes a PROTECTION VIOLATION error, if the password was requested in order to gain access to a protected file on an otherwise accessible server; or to act as though the server did not exist, in the case where the password was needed to gain any access to the server.

(LOGIN *HOSTNAME* *FLG* *DIRECTORY* *MSG*)

[Function]

Forces Medley to ask for your name and password to be used when accessing host *HOSTNAME*. Any previous login information for *HOSTNAME* is overridden. If *HOSTNAME* is NIL, it overrides login information for all hosts and resets the default user name and password to be those typed in by you. The special value *HOSTNAME* = NS:: is used to obtain the default user name and password for all logins for NS Servers.

If *FLG* is the atom QUIET, only prompts you if there is no cached information for *HOSTNAME*.

If *DIRECTORY* is specified, it is the name of a directory on *HOSTNAME*. In this case, the information requested is the "connect" password for that directory. Connect passwords for any number of different directories on a host can be maintained.

If *MSG* is non-NIL, it is a message (a string) to be printed before the name and password information is requested.

LOGIN returns the user name with which you completed the login.

(SETPASSWORD *HOST* *USER* *PASSWORD* *DIRECTORY*)

[Function]

Sets the values in the internal password database exactly as if the strings *USER* and *PASSWORD* were typed in via (LOGIN *HOST* NIL *DIRECTORY*).

(SETUSERNAME *NAME*)

[Function]

Sets the default uer name to *NAME*.

(USERNAME *FLG* *STRPTR* *PRESERVECASE*)

[Function]

If *FLG* = NIL, returns the default user name. This is the only value of *FLG* that is meaningful in Medley.

USERNAME returns the value as a string, unless *STRPTR* is T, in which case USERNAME returns the value as an atom. The name is returned in uppercase, unless *PRESERVECASE* is true.

INTERLISP-D REFERENCE MANUAL

Abnormal Conditions

If Medley tries to access a file and does not get a response from the file server in a reasonable period of time, it prints a message that the file server is not responding, and keeps trying. If the file server has actually crashed, this may continue indefinitely. A Control-E or similar interrupt aborts out of this state.

If the file server crashes but is restarted before you attempt to do anything, file operations will usually proceed normally, except for a brief pause while Medley tries to re-establish any connections it had open before the crash. However, this is not always possible. For example, when a file is open for sequential output and the server crashes, there is no way to recover the output already written, since it vanished with the crash. In such cases, the system will cause an error such as `Connection Lost`.

`LOGOUT` closes any file server connections that are currently open. On return, it attempts to re-establish connections for any files that were open before logging out. If a file has disappeared or been modified, Medley reports this fact. Files that were open for sequential access generally cannot be reopened after `LOGOUT`.

Interlisp supports simultaneous access to the same server from different processes and permits overlapping of Lisp computation with file server operations, allowing for improved performance. However, as a corollary of this, a file is not closed the instant that `CLOSEF` returns; Interlisp closes the file "in the background". It is therefore very important that you exit Interlisp via `(LOGOUT)` or `(LOGOUT T)`, rather than boot the machine.

On rare occasions, the Ethernet may appear completely unresponsive, due to Interlisp having gotten into a bad state. Type `(RESTART.ETHER)` to reinitialize Lisp's Ethernet driver(s), just as when the Lisp system is started up following a `LOGOUT`, `SYSOUT`, etc.

24. INPUT/OUTPUT FUNCTIONS

This chapter describes the standard I/O functions used for reading and printing characters and Interlisp expressions on files and other streams. First, the primitive input functions are presented, then the output functions, then functions for random-access operations (such as searching a file for a given stream, or changing the "next-character" pointer to a position in a file). Next, the `PRINTOUT` statement is documented (see below), which provides an easy way to write complex output operations. Finally, read tables, used to parse characters as Interlisp expressions, are documented.

Specifying Streams for Input/Output Functions

Most of the input/output functions in Interlisp-D have an argument named *STREAM* or *FILE*, specifying on which open stream the function's action should occur (the name *FILE* is used in older functions that predate the concept of stream; the two should, however, be treated synonymously). The value of this argument should be one of the following:

- a stream An object of type *STREAM*, as returned by `OPENSTREAM` (Chapter 23) or other stream-producing functions, is always the most precise and efficient way to designate a stream argument.
- T The litatom `T` designates the terminal input or output stream of the currently running process, controlling input from the keyboard and output to the display screen. For functions where the direction (input or output) is ambiguous, `T` is taken to designate the terminal output stream. The `T` streams are always open; they cannot be closed.

The terminal output stream can be set to a given window or display stream by using `TTYDISPLAYSTREAM` (Chapter 28). The terminal input stream cannot be changed. For more information on terminal I/O, see Chapter 30.
- NIL The litatom `NIL` designates the "primary" input or output stream. These streams are initially the same as the terminal input/output streams, but they can be changed by using the functions `INPUT` and `OUTPUT`.

For functions where the direction (input or output) is ambiguous, e.g., `GETFILEPTR`, the argument `NIL` is taken to mean the primary input stream, if that stream is not identical to the terminal input stream, else the primary output stream.
- a window Uses the display stream of the window . Valid for output only.
- a file name As of this writing, the name of an open file (as a litatom) can be used as a stream argument. However, there are inefficiencies and possible

future incompatibilities associated with doing so. See Chapter 24 for details.

(**GETSTREAM** *FILE ACCESS*) [Function]

Coerces the argument *FILE* to a stream by the above rules. If *ACCESS* is INPUT, OUTPUT, or BOTH, produces the stream designated by *FILE* that is open for *ACCESS*. If *ACCESS*=NIL, returns a stream for *FILE* open for any kind of input/output (see the list above for the ambiguous cases). If *FILE* does not designate a stream open in the specified mode, causes an error, FILE NOT OPEN.

(**STREAMP** *X*) [Function]

Returns *X* if *X* is a STREAM, otherwise NIL.

Input Functions

While the functions described below can take input from any stream, some special actions occur when the input is from the terminal (the T input stream, see above). When reading from the terminal, the input is buffered a line at a time, unless buffering has been inhibited by CONTROL (Chapter 30) or the input is being read by READC or PEEKC. Using specified editing characters, you can erase a character at a time, a word at a time, or the whole line. The keys that perform these editing functions are assignable via SETSYNTAX, with the initial settings chosen to be those most natural for the given operating system. In Interlisp-D, the initial settings are as follows: characters are deleted one at a time by Backspace; words are erased by control-W; the whole line is erased by Control-Q.

On the Interlisp-D display, deleting a character or a line causes the characters to be physically erased from the screen. In Interlisp-10, the deleting action can be modified for various types of display terminals by using DELETECONTROL (Chapter 30).

Unless otherwise indicated, when the end of file is encountered while reading from a file, all input functions generate an error, END OF FILE. Note that this does not close the input file. The ENDOFSTREAMOP stream attribute (Chapter 24) is useful for changing the behavior at end of file.

Most input functions have a *RDTBL* argument, which specifies the read table to be used for input. Unless otherwise specified, if *RDTBL* is NIL, the primary read table is used.

If the *FILE* or *STREAM* argument to an input function is NIL, the primary input stream is used.

(**INPUT** *FILE*) [Function]

Sets *FILE* as the primary input stream; returns the old primary input stream. *FILE* must be open for input.

(INPUT) returns the current primary input stream, which is not changed.

Note: If the primary input stream is set to a file, the file's full name, rather than the stream itself, is returned. See discussion in Chapter 24.

(**READ** *FILE* *RD_TBL* *FLG*)

[Function]

Reads one expression from *FILE*. Atoms are delimited by the break and separator characters as defined in *RD_TBL*. To include a break or separator character in an atom, the character must be preceded by the character %, e.g., *AB%(C* is the atom *AB(C*, %% is the atom %, %*control-K* is the atom *Control-K*. For input from the terminal, an atom containing an interrupt character can be input by typing instead the corresponding alphabetic character preceded by Control-V, e.g., ^VD for Control-D.

Strings are delimited by double quotes. To input a string containing a double quote or a %, precede it by %, e.g., "AB%"C" is the string *AB"C*. Note that % can always be typed even if next character is not "special", e.g., %A%B%C is read as *ABC*.

If an atom is interpretable as a number, *READ* creates a number, e.g., 1E3 reads as a floating point number, 1D3 as a literal atom, 1.0 as a number, 1,0 as a literal atom, etc. An integer can be input in a non-decimal radix by using syntax such as 123Q, |b10101, |5r1234 (see Chapter 7). The function *RADIX*, sets the radix used to print integers.

When reading from the terminal, all input is line-buffered to enable the action of the backspacing control characters, unless inhibited by *CONTROL* (Chapter 30). Thus no characters are actually seen by the program until a carriage-return (actually the character with terminal syntax class *EOL*, see Chapter 30), is typed. However, for reading by *READ*, when a matching right parenthesis is encountered, the effect is the same as though a carriage-return were typed, i.e., the characters are transmitted. To indicate this, Interlisp also prints a carriage-return line-feed on the terminal. The line buffer is also transmitted to *READ* whenever an *IMMEDIATE* read macro character is typed (see below).

FLG=T suppresses the carriage-return normally typed by *READ* following a matching right parenthesis. (However, the characters are still given to *READ*; i.e., you do not have to type the carriage-return.)

(**RATOM** *FILE* *RD_TBL*)

[Function]

Reads in one atom from *FILE*. Separation of atoms is defined by *RD_TBL*. % is also defined for *RATOM*, and the remarks concerning line-buffering and editing control characters also apply.

If the characters comprising the atom would normally be interpreted as a number by *READ*, that number is returned by *RATOM*. Note however that *RATOM* takes no special action for " whether or not it is a break character, i.e., *RATOM* never makes a string.

(**RSTRING** *FILE* *RD_TBL*)

[Function]

Reads characters from *FILE* up to, but not including, the next break or separator character, and returns them as a string. Backspace, Control-W, Control-Q, Control-V, and % have the same effect as with *READ*.

Note that the break or separator character that terminates a call to `RATOM` or `RSTRING` is *not* read by that call, but remains in the buffer to become the first character seen by the next reading function that is called. If that function is `RSTRING`, it will return the null string. This is a common source of program bugs.

(**RATOMS** *A FILE RDTBL*) [Function]

Calls `RATOM` repeatedly until the atom *A* is read. Returns a list of the atoms read, not including *A*.

(**RATEST** *FLG*) [Function]

If *FLG* = `T`, `RATEST` returns `T` if a separator was encountered immediately prior to the atom returned by the last `RATOM` or `READ`, `NIL` otherwise.

If *FLG* = `NIL`, `RATEST` returns `T` if last atom read by `RATOM` or `READ` was a break character, `NIL` otherwise.

If *FLG* = `1`, `RATEST` returns `T` if last atom read (by `READ` or `RATOM`) contained a `%` used to quote the next character (as in `% [` or `%A%B%C`), `NIL` otherwise.

(**READC** *FILE RDTBL*) [Function]

Reads and returns the next character, including `%`, `"`, etc, i.e., is not affected by break or separator characters. The action of `READC` is subject to line-buffering, i.e., `READC` does not return a value until the line has been terminated even if a character has been typed. Thus, the editing control characters have their usual effect. *RD_TBL* does not directly affect the value returned, but is used as usual in line-buffering, e.g., determining when input has been terminated. If `(CONTROL T)` has been executed (Chapter 30), defeating line-buffering, the *RD_TBL* argument is irrelevant, and `READC` returns a value as soon as a character is typed (even if the character typed is one of the editing characters, which ordinarily would never be seen in the input buffer).

(**PEEKC** *FILE*) [Function]

Returns the next character, but does not actually read it and remove it from the buffer. If reading from the terminal, the character is echoed as soon as `PEEKC` reads it, even though it is then "put back" into the system buffer, where Backspace, Control-W, etc. could change it. Thus it is possible for the value returned by `PEEKC` to "disagree" in the first character with a subsequent `READ`.

(**LASTC** *FILE*) [Function]

Returns the last character read from *FILE*. `LASTC` can return an incorrect result when called immediately following a `PEEKC` on a file that contains run-coded NS characters.

(**READCCODE** *FILE RDTBL*) [Function]

Returns the next character *code* from *STREAM*; thus, this operation is equivalent to, but more efficient than, `(CHCON1 (READC FILE RDTBL))`.

(**PEEKCCODE** *FILE*) [Function]

Returns, without consuming, the next character *code* from *STREAM*; thus, this operation is equivalent to, but more efficient than, (CHCON1 (PEEK *FILE*)).

(**BIN** *STREAM*) [Function]

Returns the next byte from *STREAM*. This operation is useful for reading streams of binary, rather than character, data.

Note: BIN is similar to READCCODE, except that BIN always reads a single byte, whereas READCCODE reads a "character" that can consist of more than one byte, depending on the character and its encoding.

READ, RATOM, RATOMS, PEEKC, READC all wait for input if there is none. The only way to test whether or not there is input is to use READP:

(**READP** *FILE* *FLG*) [Function]

Returns T if there is anything in the input buffer of *FILE*, NIL otherwise. This operation is only interesting for streams whose source of data is dynamic, e.g., the terminal or a byte stream over a network; for other streams, such as to files, (READP *FILE*) is equivalent to (NOT (EOFP *FILE*)).

Note that because of line-buffering, READP may return T, indicating there is input in the buffer, but READ may still have to wait.

Frequently, the terminal's input buffer contains a single EOL character left over from a previous input. For most applications, this situation wants to be treated as though the buffer were empty, and so READP returns NIL in this case. However, if *FLG*=T, READP returns T if there is *any* character in the input buffer, including a single EOL. *FLG* is ignored for streams other than the terminal.

(**EOFP** *FILE*) [Function]

Returns true if *FILE* is at "end of file", i.e., the next call to an input function would cause an END OF FILE error; NIL otherwise. For randomly accessible files, this can also be thought of as the file pointer pointing beyond the last byte of the file. *FILE* must be open for (at least) input, or an error is generated, FILE NOT OPEN.

Note that EOFP can return NIL and yet the next call to READ might still cause an END OF FILE error, because the only characters remaining in the input were separators or otherwise constituted an incomplete expression. The function SKIPSEPRS is sometimes more useful as a way of detecting end of file when it is known that all the expressions in the file are well formed.

(**WAITFORINPUT** *FILE*) [Function]

Waits until input is available from *FILE* or from the terminal, i.e. from T. WAITFORINPUT is functionally equivalent to (until (OR (READP T) (READP *FILE*)) do NIL),

except that it does not use up machine cycles while waiting. Returns the device for which input is now available, i.e. *FILE* or T.

FILE can also be an integer, in which case WAITFORINPUT waits until there is input available from the terminal, or until *FILE* milliseconds have elapsed. Value is T if input is now available, NIL in the case that WAITFORINPUT timed out.

(**SKREAD** *FILE* *REREADSTRING* *RD_TBL*) [Function]

"Skip Read". SKREAD consumes characters from *FILE* as if one call to READ had been performed, without paying the storage and compute cost to really read in the structure. *REREADSTRING* is for the case where the caller has already performed some READC's and RATOM's before deciding to skip this expression. In this case, *REREADSTRING* should be the material already read (as a string), and SKREAD operates as though it had seen that material first, thus setting up its parenthesis count, double-quote count, etc.

The read table *RD_TBL* is used for reading from *FILE*. If *RD_TBL* is NIL, it defaults to the value of *FILERD_TBL*. SKREAD may have difficulties if unusual read macros are defined in *RD_TBL*. SKREAD does not recognize read macro characters in *REREADSTRING*, nor SPLICE or INFIX read macros. This is only a problem if the read macros are defined to parse subsequent input in the stream that does not follow the normal parenthesis and string-quote conventions.

SKREAD returns %) if the read terminated on an unbalanced closing parenthesis; %] if the read terminated on an unbalanced %], i.e., one which also would have closed any extant open left parentheses; otherwise NIL.

(**SKIPSEPRS** *FILE* *RD_TBL*) [Function]

Consumes characters from *FILE* until it encounters a non-separator character (as defined by *RD_TBL*). SKIPSEPRS returns, but does not consume, the terminating character, so that the next call to READC would return the same character. If no non-separator character is found before the end of file is reached, SKIPSEPRS returns NIL and leaves the stream at end of file. This function is useful for skipping over "white space" when scanning a stream character by character, or for detecting end of file when reading expressions from a stream with no pre-arranged terminating expression.

Output Functions

Unless otherwise specified by DEFPRINT, pointers other than lists, strings, atoms, or numbers, are printed in the form {*DATATYPE*} followed by the octal representation of the address of the pointer (regardless of radix). For example, an array pointer might print as {ARRAYP}#43,2760. This printed representation is for compactness of display on your terminal, and will *not* read back in correctly; if the form above is read, it will produce the listatom {ARRAYP}#43,2760.

Note: The term "end-of-line" appearing in the description of an output function means the character or characters used to terminate a line in the file system being used

by the given implementation of Interlisp. For example, in Interlisp-D end-of-line is indicated by the character carriage-return.

Some of the functions described below have a *RD_TBL* argument, which specifies the read table to be used for output. If *RD_TBL* is *NIL*, the primary read table is used.

Most of the functions described below have an argument *FILE*, which specifies the stream on which the operation is to take place. If *FILE* is *NIL*, the primary output stream is used .

(**OUTPUT** *FILE*) [Function]

Sets *FILE* as the primary output stream; returns the old primary output stream. *FILE* must be open for output.

(**OUTPUT**) returns the current primary output stream, which is not changed.

Note: If the primary output stream is set to a file, the file's full name, rather than the stream itself, is returned. See the discussion in Chapter 24.

(**PRIN1** *X FILE*) [Function]

Prints *X* on *FILE*.

(**PRIN2** *X FILE RD_TBL*) [Function]

Prints *X* on *FILE* with %'s and "'s inserted where required for it to read back in properly by **READ**, using *RD_TBL*.

Both **PRIN1** and **PRIN2** print any kind of Lisp expression, including lists, atoms, numbers, and strings. **PRIN1** is generally used for printing expressions where human readability, rather than machine readability, is important, e.g., when printing text rather than program fragments. **PRIN1** does not print double quotes around strings, or % in front of special characters. **PRIN2** is used for printing Interlisp expressions which can then be read back into Interlisp with **READ**; i.e., break and separator characters in atoms will be preceded by %'s. For example, the atom "()" is printed as % (%) by **PRIN2**. If the integer output radix (as set by **RADIX**) is not 10, **PRIN2** prints the integer using the input syntax for non-decimal integers (see Chapter 7) but **PRIN1** does not (but both print the integer in the output radix).

(**PRIN3** *X FILE*) [Function]

(**PRIN4** *X FILE RD_TBL*) [Function]

PRIN3 and **PRIN4** are the same as **PRIN1** and **PRIN2** respectively, except that they do not increment the horizontal position counter nor perform any linelength checks. They are useful primarily for printing control characters.

(**PRINT** *X FILE RD_TBL*) [Function]

Prints the expression *X* using **PRIN2** followed by an end-of-line. Returns *X*.

(**PRINTCCODE** *CHARCODE FILE*) [Function]

Outputs a single character whose code is *CHARCODE* to *FILE*. This is similar to (**PRIN1** (*CHARACTER CHARCODE*)), except that numeric characters are guaranteed to print "correctly"; e.g., (**PRINTCCODE** (*CHARCODE 9*)) always prints "9", independent of the setting of *RADIX*.

PRINTCCODE may actually print more than one byte on *FILE*, due to character encoding and end of line conventions; thus, no assumptions should be made about the relative motion of the file pointer (see **GETFILEPTR**) during this operation.

(**BOUT** *STREAM BYTE*) [Function]

Outputs a single 8-bit byte to *STREAM*. This is similar to **PRINTCCODE**, but for binary streams the character position in *STREAM* is not updated (as with **PRIN3**), and end of line conventions are ignored.

Note: **BOUT** is similar to **PRINTCCODE**, except that **BOUT** always writes a single byte, whereas **PRINTCCODE** writes a "character" that can consist of more than one byte, depending on the character and its encoding.

(**SPACES** *N FILE*) [Function]

Prints *N* spaces. Returns **NIL**.

(**TERPRI** *FILE*) [Function]

Prints an end-of-line character. Returns **NIL**.

(**FRESHLINE** *STREAM*) [Function]

Equivalent to **TERPRI**, except it does nothing if it is already at the beginning of the line. Returns **T** if it prints an end-of-line, **NIL** otherwise.

(**TAB** *POS MINSPACES FILE*) [Function]

Prints the appropriate number of spaces to move to position *POS*. *MINSPACES* indicates how many spaces must be printed (if **NIL**, 1 is used). If the current position plus *MINSPACES* is greater than *POS*, **TAB** does a **TERPRI** and then (**SPACES** *POS*). If *MINSPACES* is **T**, and the current position is greater than *POS*, then **TAB** does nothing.

Note: A sequence of **PRINT**, **PRIN2**, **SPACES**, and **TERPRI** expressions can often be more conveniently coded with a single **PRINTOUT** statement.

(**SHOWPRIN2** *X FILE RDTBL*) [Function]

Like **PRIN2** except if **SYSPRETTYFLG**=**T**, prettyprints *X* instead. Returns *X*.

(**SHOWPRINT** *X FILE RDTBL*) [Function]

Like **PRINT** except if **SYSPRETTYFLG**=T, prettyprints *X* instead, followed by an end-of-line. Returns *X*.

SHOWPRINT and **SHOWPRIN2** are used by the programmer's assistant (Chapter 13) for printing the values of expressions and for printing the history list, by various commands of the beak package (Chapter 14), e.g. **?=** and **BT** commands, and various other system packages. The idea is that by simply setting or binding **SYSPRETTYFLG** to **T** (initially **NIL**), you instruct the system when interacting with you to **PRETTYPRINT** expressions (Chapter 26) instead of printing them.

(**PRINTBELLS**) [Function]

Used by **DWIM** (Chapter 19) to print a sequence of bells to alert you to stop typing. Can be advised or redefined for special applications, e.g., to flash the screen on a display terminal.

(**FORCEOUTPUT** *STREAM WAITFORFINISH*) [Function]

Forces any buffered output data in *STREAM* to be transmitted.

If *WAITFORFINISH* is non-**NIL**, this doesn't return until the data has been forced out.

(**POSITION** *FILE N*) [Function]

Returns the column number at which the next character will be read or printed. After a end of line, the column number is 0. If *N* is non-**NIL**, resets the column number to be *N*.

Note that resetting **POSITION** only changes Lisp's belief about the current column number; it does not cause any horizontal motion. Also note that (**POSITION** *FILE*) is not the same as (**GETFILEPTR** *FILE*) which gives the position in the *file*, not on the *line*.

(**LINELENGTH** *N FILE*) [Function]

Sets the length of the print line for the output file *FILE* to *N*; returns the former setting of the line length. *FILE* defaults to the primary output stream. (**LINELENGTH** **NIL** *FILE*) returns the current setting for *FILE*. When a file is first opened, its line length is set to the value of the variable **FILELINELENGTH**.

Whenever printing an atom or string would increase a file's position *beyond* the line length of the file, an end of line is automatically inserted first. This action can be defeated by using **PRIN3** and **PRIN4**.

(**SETLINELENGTH** *N*) [Function]

Sets the line length for the terminal by doing (**LINELENGTH** *N* **T**). If *N* is **NIL**, it determines *N* by consulting the operating system's belief about the terminal's characteristics. In Interlisp-D, this is a no-op.

PRINTLEVEL

When using Interlisp one often has to handle large, complicated lists, which are difficult to understand when printed out. `PRINTLEVEL` allows you to specify in how much detail lists should be printed. The print functions `PRINT`, `PRIN1`, and `PRIN2` are all affected by level parameters set by:

`(PRINTLEVEL CARVAL CDRVAL)` [Function]

Sets the CAR print level to *CARVAL*, and the CDR print level to *CDRVAL*. Returns a list cell whose CAR and CDR are the old settings. `PRINTLEVEL` is initialized with the value `(1000 . -1)`.

In order that `PRINTLEVEL` can be used with `RESETFORM` or `RESETSAVE`, if *CARVAL* is a list cell it is equivalent to `(PRINTLEVEL (CAR CARVAL) (CDR CARVAL))`.

`(PRINTLEVEL NNIL)` changes the CAR printlevel without affecting the CDR printlevel.

`(PRINTLEVEL NIL N)` changes the CDR printlevel with affecting the CAR printlevel.

`(PRINTLEVEL)` gives the current setting without changing either.

Note: Control-P (Chapter 30) can be used to change the `PRINTLEVEL` setting dynamically, even while Interlisp is printing.

The CAR printlevel specifies how "deep" to print a list. Specifically, it is the number of unpaired left parentheses which will be printed. Below that level, all lists will be printed as `&`. If the CAR printlevel is *negative*, the action is similar except that an end-of-line is inserted after each right parentheses that would be immediately followed by a left parenthesis.

The CDR printlevel specifies how "long" to print a list. It is the number of top level list elements that will be printed before the printing is terminated with `--`. For example, if *CDRVAL*=2, `(A B C D E)` will print as `(A B --)`. For sublists, the number of list elements printed is also affected by the depth of printing in the CAR direction: Whenever the *sum* of the depth of the sublist (i.e. the number of unmatched left parentheses) and the number of elements is greater than the CDR printlevel, `--` is printed. This gives a "triangular" effect in that less is printed the farther one goes in either CAR or CDR direction. If the CDR printlevel is negative, then it is the same as if the CDR printlevel were infinite.

Examples:

After:	<code>(A (B C (D (E F) G) H) K L)</code> prints as:
<code>(PRINTLEVEL 3 -1)</code>	<code>(A (B C (D & G) H) K L)</code>
<code>(PRINTLEVEL 2 -1)</code>	<code>(A (B C & H) K L)</code>
<code>(PRINTLEVEL 1 -1)</code>	<code>(A & K L)</code>
<code>(PRINTLEVEL 0 -1)</code>	<code>&</code>
<code>(PRINTLEVEL 1000 2)</code>	<code>(A (B --) --)</code>
<code>(PRINTLEVEL 1000 3)</code>	<code>(A (B C --) K --)</code>

(PRINTLEVEL 1 3) (A & K --)

PLVLFILEFLG

[Variable]

Normally, PRINTLEVEL only affects terminal output. Output to all other files acts as though the print level is infinite. However, if PLVLFILEFLG is T (initially NIL), then PRINTLEVEL affects output to files as well.

The following three functions are useful for printing isolated expressions at a specified print level without going to the overhead of resetting the global print level.

(**LVLPRINT** *X FILE CARLVL CDRLVL TAIL*)

[Function]

Performs PRINT of *X* to *FILE*, using as CAR and CDR print levels the values *CARLVL* and *CDRLVL*, respectively. Uses the T read table. If *TAIL* is specified, and *X* is a tail of it, then begins its printing with ". . .", rather than on open parenthesis.

(**LVLPRIN2** *X FILE CARLVL CDRLVL TAIL*)

[Function]

Similar to LVLPRIN2, but performs a PRIN2.

(**LVLPRIN1** *X FILE CARLVL CDRLVL TAIL*)

[Function]

Similar to LVLPRIN1, but performs a PRIN1.

Printing Numbers

How the ordinary printing functions (PRIN1, PRIN2, etc.) print numbers can be affected in several ways. RADIX influences the printing of integers, and FLTfmt influences the printing of floating point numbers. The setting of the variable PRXFLG determines how the symbol-manipulation functions handle numbers. The PRINTNUM package permits greater controls on the printed appearance of numbers, allowing such things as left-justification, suppression of trailing decimals, etc.

(**RADIX** *N*)

[Function]

Resets the output radix for integers to the absolute value of *N*. The value of RADIX is its previous setting. (RADIX) gives the current setting without changing it. The initial setting is 10.

Note that RADIX affects output *only*. There is no input radix; on input, numbers are interpreted as decimal unless they are entered in a non-decimal radix with syntax such as 123Q, |b10101, |5r1234 (see Chapter 7). RADIX does not affect the behavior of UNPACK, etc., unless the value of PRXFLG (below) is T. For example, if PRXFLG is NIL and the radix is set to 8 with (RADIX 8), the value of (UNPACK 9) is (9), not (1 1).

Using PRINTNUM (below) or the PRINTOUT command .I (below) is often a more convenient and appropriate way to print a single number in a specified radix than to globally change RADIX.

(**FLTFMT** *FORMAT*)

[Function]

Resets the output format for floating point numbers to the **FLOAT** format *FORMAT* (see **PRINTNUM** below for a description of **FLOAT** formats). *FORMAT*=**T** specifies the default "free" formatting: some number of significant digits (a function of the implementation) are printed, with trailing zeros suppressed; numbers with sufficiently large or small exponents are instead printed in exponent notation.

FLTFMT returns its current setting. (**FLTFMT**) returns the current setting without changing it. The initial setting is **T**.

Note: In Interlisp-D, **FLTFMT** ignores the *WIDTH* and *PAD* fields of the format (they are implemented only by **PRINTNUM**).

Whether print name manipulation functions (**UNPACK**, **NCHARS**, etc.) use the values of **RADIX** and **FLTFMT** is determined by the variable **PRXFLG**:

PRXFLG

[Variable]

If **PRXFLG**=**NIL** (the initial setting), then the "PRIN1" name used by **PACK**, **UNPACK**, **MKSTRING**, etc., is computed using base 10 for integers and the system default floating format for floating point numbers, independent of the current setting of **RADIX** or **FLTFMT**. If **PRXFLG**=**T**, then **RADIX** and **FLTFMT** do dictate the "PRIN1" name of numbers. Note that in this case, **PACK** and **UNPACK** are *not* inverses.

Examples with (**RADIX** 8), (**FLTFMT** '(**FLOAT** 4 2)):

With **PRXFLG**=**NIL**,

```
(UNPACK 13) => (1 3)
(PACK '(A 9)) => A9
(UNPACK 1.2345) => (1 %. 2 3 4 5)
```

With **PRXFLG**=**T**,

```
(UNPACK 13) => (1 5)
(PACK '(A 9)) => A11
(UNPACK 1.2345) => (1 %. 2 3)
```

Note that **PRXFLG** does not effect the radix of "PRIN2" names, so with (**RADIX** 8), (**NCHARS** 9 **T**), which uses **PRIN2** names, would return 3, (since 9 would print as 11Q) for either setting of **PRXFLG**.

Warning: Some system functions will not work correctly if **PRXFLG** is not **NIL**. Therefore, resetting the global value of **PRXFLG** is not recommended. It is much better to rebind **PRXFLG** as a **SPECVAR** for that part of a program where it needs to be non-**NIL**.

The basic function for printing numbers under format control is **PRINTNUM**. Its utility is considerably enhanced when used in conjunction with the **PRINTOUT** package, which implements a compact language for specifying complicated sequences of elementary printing operations, and makes fancy output formats easy to design and simple to program.

(**PRINTNUM** *FORMAT* *NUMBER* *FILE*)

[Function]

Prints *NUMBER* on *FILE* according to the format *FORMAT*. *FORMAT* is a list structure with one of the forms described below.

If *FORMAT* is a list of the form (*FIX* *WIDTH* *RADIX* *PAD0* *LEFTFLUSH*), this specifies a *FIX* format. *NUMBER* is rounded to the nearest integer, and then printed in a field *WIDTH* characters long with radix set to *RADIX* (or 10 if *RADIX*=NIL; note that the setting from the function *RADIX* is *not* used as the default). If *PAD0* and *LEFTFLUSH* are both NIL, the number is right-justified in the field, and the padding characters to the left of the leading digit are spaces. If *PAD0* is T, the character "0" is used for padding. If *LEFTFLUSH* is T, then the number is left-justified in the field, with trailing spaces to fill out *WIDTH* characters.

The following examples illustrate the effects of the *FIX* format options on the number 9 (the vertical bars indicate the field width):

```
FORMAT:      (PRINTNUM FORMAT 9) prints:
(FIX 2)      | 9 |
(FIX 2 NIL T) | 09 |
(FIX 12 8 T)  | 000000000011 |
(FIX 5 NIL NIL T) | 9      |
```

If *FORMAT* is a list of the form (*FLOAT* *WIDTH* *DECPART* *EXPPART* *PAD0* *ROUND*), this specifies a *FLOAT* format. *NUMBER* is printed as a decimal number in a field *WIDTH* characters wide, with *DECPART* digits to the right of the decimal point. If *EXPPART* is not 0 (or NIL), the number is printed in exponent notation, with the exponent occupying *EXPPART* characters in the field. *EXPPART* should allow for the character E and an optional sign to be printed before the exponent digits. As with *FIX* format, padding on the left is with spaces, unless *PAD0* is T. If *ROUND* is given, it indicates the digit position at which rounding is to take place, counting from the leading digit of the number.

Interlisp-D interprets *WIDTH*=NIL to mean no padding, i.e., to use however much space the number needs, and interprets *DECPART*=NIL to mean as many decimal places as needed.

The following examples illustrate the effects of the *FLOAT* format options on the number 27.689 (the vertical bars indicate the field width):

```
FORMAT:      (PRINTNUM FORMAT 27.689) prints:
(FLOAT 7 2)   | 27.69 |
(FLOAT 7 2 NIL 0) | 0027.69 |
(FLOAT 7 2 2)  | 2.77E1 |
(FLOAT 11 2 4) | 2.77E+01 |
(FLOAT 7 2 NIL NIL 1) | 30.00 |
(FLOAT 7 2 NIL NIL 2) | 28.00 |
```

NILNUMPRINTFLG

[Variable]

If PRINTNUM's *NUMBER* argument is not a number and not NIL, a NON-NUMERIC ARG error is generated. If *NUMBER* is NIL, the effect depends on the setting of the variable NILNUMPRINTFLG. If NILNUMPRINTFLG is NIL, then the error occurs as usual. If it is non-NIL, then no error occurs, and the value of NILNUMPRINTFLG is printed right-justified in the field described by *FORMAT*. This option facilitates the printing of numbers in aggregates with missing values coded as NIL.

User Defined Printing

Initially, Interlisp only knows how to print in an interesting way objects of type *litatom*, *number*, *string*, *list* and *stackp*. All other types of objects are printed in the form {*datatype*} followed by the octal representation of the address of the pointer, a format that cannot be read back in to produce an equivalent object. When defining user data types (using the *DATATYPE* record type, Chapter 8), it is often desirable to specify as well how objects of that type should be printed, so as to make their contents readable, or at least more informative to the viewer. The function *DEFPRINT* is used to specify the printing format of a data type.

(**DEFPRINT** *TYPE FN*)

[Function]

TYPE is a type name. Whenever a printing function (*PRINT*, *PRIN1*, *PRIN2*, etc.) or a function requiring a print name (*CHCON*, *NCHARS*, etc.) encounters an object of the indicated type, *FN* is called with two arguments: the item to be printed and the name of the stream, if any, to which the object is to be printed. The second argument is NIL on calls that request the print name of an object without actually printing it.

If *FN* returns a list of the form (*ITEM1* . *ITEM2*), *ITEM1* is printed using *PRIN1* (unless it is NIL), and then *ITEM2* is printed using *PRIN2* (unless it is NIL). No spaces are printed between the two items. Typically, *ITEM1* is a read macro character.

If *FN* returns NIL, the datum is printed in the system default manner.

If *FN* returns T, nothing further is printed; *FN* is assumed to have printed the object to the stream itself. Note that this case is permitted only when the second argument passed to *FN* is non-NIL; otherwise, there is no destination for *FN* to do its printing, so it must return as in one of the other two cases.

Printing Unusual Data Structures

HPRINT (for "Horrible Print") and *HREAD* provide a mechanism for printing and reading back in general data structures that cannot normally be dumped and loaded easily, such as (possibly re-entrant or circular) structures containing user datatypes, arrays, hash tables, as well as list structures. *HPRINT* will correctly print and read back in any structure containing any or all of the above, chasing all pointers down to the level of literal atoms, numbers or strings. *HPRINT* currently cannot handle compiled code arrays, stack positions, or arbitrary unboxed numbers.

HPRINT operates by simulating the Interlisp PRINT routine for normal list structures. When it encounters a user datatype (see Chapter 8), or an array or hash array, it prints the data contained therein, surrounded by special characters defined as read macro characters. While chasing the pointers of a structure, it also keeps a hash table of those items it encounters, and if any item is encountered a second time, another read macro character is inserted before the first occurrence (by resetting the file pointer with SETFILEPTR) and all subsequent occurrences are printed as a back reference using an appropriate macro character. Thus the inverse function, HREAD merely calls the Interlisp READ routine with the appropriate read table.

(HPRINT *EXPR* *FILE* *UNCIRCULAR* *DATATYPESEEN*) [Function]

Prints *EXPR* on *FILE*. If *UNCIRCULAR* is non-NIL, HPRINT does no checking for any circularities in *EXPR* (but is still useful for dumping arbitrary structures of arrays, hash arrays, lists, user data types, etc., that do not contain circularities). Specifying *UNCIRCULAR* as non-NIL results in a large speed and internal-storage advantage.

Normally, when HPRINT encounters a user data type for the first time, it outputs a summary of the data type's declaration. When this is read in, the data type is redeclared. If *DATATYPESEEN* is non-NIL, HPRINT assumes that the same data type declarations will be in force at read time as were at HPRINT time, and not output declarations.

HPRINT is intended primarily for output to random access files, since the algorithm depends on being able to reset the file pointer. If *FILE* is not a random access file (and *UNCIRCULAR* = NIL), a temporary file, HPRINT.SCRATCH, is opened, *EXPR* is HPRINTed on it, and then that file is copied to the final output file and the temporary file is deleted.

You can not use HPRINT to save things that contains pointers to raw storage. Fontdescriptors contain pointers to raw storage and windows contain pointers to fontdescriptors. Netiher can therefor be saved with HPRINT.

(HREAD *FILE*) [Function]

Reads and returns an HPRINT-ed expression from *FILE*.

(HCOPYALL *X*) [Function]

Copies data structure *X*. *X* may contain circular pointers as well as arbitrary structures.

Note: HORRIBLEVARS and UGLYVARS (Chapter 17) are two file package commands for dumping and reloading circular and re-entrant data structures. They provide a convenient interface to HPRINT and HREAD.

When HPRINT is dumping a data structure that contains an instance of an Interlisp datatype, the datatype declaration is also printed onto the file. Reading such a data structure with HREAD can cause problems if it redefines a system datatype. Redefining a system datatype will almost definitely cause serious errors. The Interlisp system datatypes do not change very often, but there is always a possibility when loading in old files created under an old Interlisp release.

To prevent accidental system crashes, HREAD will *not* redefine datatypes. Instead, it will cause an error "attempt to read DATATYPE with different field

specification than currently defined". Continuing from this error will redefine the datatype.

Random Access File Operations

For most applications, files are read starting at their beginning and proceeding sequentially, i.e., the next character read is the one immediately following the last character read. Similarly, files are written sequentially. However, for files on some devices, it is also possible to read / write characters at arbitrary positions in a file, essentially treating the file as a large block of auxiliary storage. For example, one application might involve writing an expression at the *beginning* of the file, and then reading an expression from a specified point in its *middle*. This particular example requires the file be open for *both* input and output. However, random file input or output can also be performed on files that have been opened for only input or only output.

Associated with each file is a "file pointer" that points to the location where the next character is to be read from or written to. The file position of a byte is the number of bytes that precede it in the file, i.e., 0 is the position of the beginning of the file. The file pointer to a file is automatically advanced after each input or output operation. This section describes functions which can be used to *reposition* the file pointer on those files that can be randomly accessed. A file used in this fashion is much like an array in that it has a certain number of addressable locations that characters can be put into or taken from. However, unlike arrays, files can be enlarged. For example, if the file pointer is positioned at the end of a file and anything is written, the file "grows." It is also possible to position the file pointer *beyond* the end of file and then to write. (If the program attempts to *read* beyond the end of file, an `END OF FILE` error occurs.) In this case, the file is enlarged, and a "hole" is created, which can later be written into. Note that this enlargement only takes place at the *end* of a file; it is not possible to make more room in the middle of a file. In other words, if expression A begins at position 1000, and expression B at 1100, and the program attempts to overwrite A with expression C, whose printed representation is 200 bytes long, part of B will be altered.

Warning: File positions are always in terms of bytes, not characters. You should thus be very careful about computing the space needed for an expression. In particular, NS characters may take multiple bytes (see below). Also, the end-of-line character (see Chapter 24) may be represented by a different number of characters in different implementations. Output functions may also introduce end-of-line's as a result of `LINELENGTH` considerations. Therefore `NCHARS` (see Chapter 2) does *not* specify how many bytes an expression takes to print, even ignoring line length considerations.

(**GETFILEPTR** *FILE*) [Function]

Returns the current position of the file pointer for *FILE*, i.e., the byte address at which the next input/output operation will commence.

(**SETFILEPTR** *FILE ADR*) [Function]

Sets the file pointer for *FILE* to the position *ADR*; returns *ADR*. The special value *ADR*=-1 is interpreted to mean the address of the end of file.

Note: If a file is opened for output only, the end of file is initially zero, even if an old file by the same name had existed (see `OPENSTREAM`, Chapter 24). If a file is opened for both input and output, the initial file pointer is the beginning of the file, but `(SETFILEPTR FILE -1)` sets it to the end of the file. If the file had been opened in append mode by `(OPENSTREAM FILE 'APPEND)`, the file pointer right after opening would be set to the end of the existing file, in which case a `SETFILEPTR` to position the file at the end would be unnecessary.

(GETEOFPTR FILE) [Function]

Returns the byte address of the end of file, i.e., the number of bytes in the file. Equivalent to performing `(SETFILEPTR FILE -1)` and returning `(GETFILEPTR FILE)` except that it does not change the current file pointer.

(RANDACCESSP FILE) [Function]

Returns `FILE` if `FILE` is randomly accessible, `NIL` otherwise. The file `T` is not randomly accessible, nor are certain network file connections in Interlisp-D. `FILE` must be open or an error is generated, `FILE NOT OPEN`.

(COPYBYTES SRCFIL DSTFIL START END) [Function]

Copies bytes from `SRCFIL` to `DSTFIL`, starting from position `START` and up to but not including position `END`. Both `SRCFIL` and `DSTFIL` must be open. Returns `T`.

If `END=NIL`, `START` is interpreted as the number of bytes to copy (starting at the current position). If `START` is also `NIL`, bytes are copied until the end of the file is reached.

Warning: `COPYBYTES` does not take any account of multi-byte NS characters (see Chapter 2). `COPYCHARS` (below) should be used whenever copying information that might include NS characters.

(COPYCHARS SRCFIL DSTFIL START END) [Function]

Like `COPYBYTES` except that it copies NS characters (see Chapter 2), and performs the proper conversion if the end-of-line conventions of `SRCFIL` and `DSTFIL` are not the same (see Chapter 24). `START` and `END` are interpreted the same as with `COPYBYTES`, i.e., as byte (not character) specifications in `SRCFIL`. The number of bytes actually output to `DSTFIL` might be more or less than the number of bytes specified by `START` and `END`, depending on what the end-of-line conventions are. In the case where the end-of-line conventions happen to be the same, `COPYCHARS` simply calls `COPYBYTES`.

(FILEPOS STR FILE START END SKIP TAIL CASEARRAY) [Function]

Analogous to `STRPOS` (see Chapter 4), but searches a file rather than a string. `FILEPOS` searches `FILE` for the string `STR`. Search begins at `START` (or the current position of the file pointer, if `START=NIL`), and goes to `END` (or the end of `FILE`, if `END=NIL`). Returns the address of the start of the match, or `NIL` if not found.

SKIP can be used to specify a character which matches any character in the file. If *TAIL* is T, and the search is successful, the value is the address of the first character *after* the sequence of characters corresponding to *STR*, instead of the starting address of the sequence. In either case, the file is left so that the next i/o operation begins at the address returned as the value of *FILEPOS*.

CASEARRAY should be a "case array" that specifies that certain characters should be transformed to other characters before matching. Case arrays are returned by *CASEARRAY* or *SEPRCASE* below. *CASEARRAY=NIL* means no transformation will be performed.

A case array is an implementation-dependent object that is logically an array of character codes with one entry for each possible character. *FILEPOS* maps each character in the file "through" *CASEARRAY* in the sense that each character code is transformed into the corresponding character code from *CASEARRAY* before matching. Thus if two characters map into the same value, they are treated as equivalent by *FILEPOS*. *CASEARRAY* and *SETCASEARRAY* provide an implementation-independent interface to case arrays.

For example, to search without regard to upper and lower case differences, *CASEARRAY* would be a case array where all characters map to themselves, except for lower case characters, whose corresponding elements would be the upper case characters. To search for a delimited atom, one could use " *ATOM* " as the pattern, and specify a case array in which all of the break and separator characters mapped into the same code as space.

For applications calling for extensive file searches, the function *FFILEPOS* is often faster than *FILEPOS*.

(**FFILEPOS** *PATTERN FILE START END SKIP TAIL CASEARRAY*) [Function]

Like *FILEPOS*, except much faster in most applications. *FFILEPOS* is an implementation of the Boyer-Moore fast string searching algorithm. This algorithm preprocesses the string being searched for and then scans through the file in steps usually equal to the length of the string. Thus, *FFILEPOS* speeds up roughly in proportion to the length of the string, e.g., a string of length 10 will be found twice as fast as a string of length 5 in the same position.

Because of certain fixed overheads, it is generally better to use *FILEPOS* for short searches or short strings.

(**CASEARRAY** *OLDARRAY*) [Function]

Creates and returns a new case array, with all elements set to themselves, to indicate the identity mapping. If *OLDARRAY* is given, it is reused.

(**SETCASEARRAY** *CASEARRAY FROMCODE TOCODE*) [Function]

Modifies the case array *CASEARRAY* so that character code *FROMCODE* is mapped to character code *TOCODE*.

(**GETCASEARRAY** CASEARRAY FROMCODE)

[Function]

Returns the character code that *FROMCODE* is mapped to in *CASEARRAY*.

(**SEPRCASE** CLFLG)

[Function]

Returns a new case array suitable for use by *FILEPOS* or *FFILEPOS* in which all of the break/separators of *FILERDTBL* are mapped into character code zero. If *CLFLG* is non-NIL, then all CLISP characters are mapped into this character as well. This is useful for finding a delimited atom in a file. For example, if *PATTERN* is " FOO ", and (*SEPRCASE* T) is used for *CASEARRAY*, then *FILEPOS* will find " (FOO_".

UPPERCASEARRAY

[Variable]

Value is a case array in which every lowercase character is mapped into the corresponding uppercase character. Useful for searching text files.

Input/Output Operations with Characters and Bytes

Interlisp-D supports the 16-bit NS character set (see Chapter 2). All of the standard string and print name functions accept litatoms and strings containing NS characters. In almost all cases, a program does not have to distinguish between NS characters or 8-bit characters. The exception to this rule is the handling of input/output operations.

Interlisp-D uses two ways of writing 16-bit NS characters on files. One way is to write the full 16-bits (two bytes) every time a character is output. The other way is to use "run-encoding." Each 16 NS character can be decoded into a character set (an integer from 0 to 254 inclusive) and a character number (also an integer from 0 to 254 inclusive). In run-encoding, the byte 255 (illegal as either a character set number or a character number) is used to signal a change to a given character set, and the following bytes are all assumed to come from the same character set (until the next change-character set sequence). Run-encoding can reduce the number of bytes required to encode a string of NS characters, as long as there are long sequences of characters from the same character set (usually the case).

Note that characters are not the same as bytes. A single character can take anywhere from one to four bytes, depending on whether it is in the same character set as the preceding character, and whether run-encoding is enabled. Programs which assume that characters are equal to bytes must be changed to work with NS characters.

The functions *BIN* and *BOUT* (see above) should only be used to read and write single eight-bit bytes. The functions *READCCODE* and *PRINTCCODE* (see above) should be used to read and write single character codes, interpreting run-encoded NS characters. *COPYBYTES* should only be used to copy blocks of 8-bit data; *COPYCHARS* should be used to copy characters. Most I/O functions (*READC*, *PRIN1*, etc.) read or write 16-bit NS characters.

The use of NS characters has serious consequences for any program that uses file pointers to access a file in a random access manner. At any point when a file is being read or written, it has a "current character set." If the file pointer is changed with `SETFILEPTR` to a part of the file with a different character set, any characters read or written may have the wrong character set. The current character set can be accessed with the following function:

(**CHARSET** *STREAM* *CHARACTERSET*) [Function]

Returns the current character set of the stream *STREAM*. If *CHARACTERSET* is non-NIL, the current character set for *STREAM* is set. Note that for output streams this may cause bytes to be written to the stream.

If *CHARACTERSET* is T, run encoding for *STREAM* is disabled: both the character set and the character number (two bytes total) will be written to the stream for each character printed.

PRINTOUT

Interlisp provides many facilities for controlling the format of printed output. By executing various sequences of `PRIN1`, `PRIN2`, `TAB`, `TERPRI`, `SPACES`, `PRINTNUM`, and `PRINTDEF`, almost any effect can be achieved. `PRINTOUT` implements a compact language for specifying complicated sequences of these elementary printing functions. It makes fancy output formats easy to design and simple to program.

`PRINTOUT` is a CLISP word (like `FOR` and `IF`) for interpreting a special printing language in which you can describe the kinds of printing desired. The description is translated by `DWIMIFY` to the appropriate sequence of `PRIN1`, `TAB`, etc., before it is evaluated or compiled. `PRINTOUT` printing descriptions have the following general form:

(`PRINTOUT` *STREAM* *PRINTCOM* ... *PRINTCOM*)

STREAM is evaluated to obtain the stream to which the output from this specification is directed. The `PRINTOUT` commands are strung together, one after the other without punctuation, after *STREAM*. Some commands occupy a single position in this list, but many commands expect to find arguments following the command name in the list. The commands fall into several logical groups: one set deals with horizontal and vertical spacing, another group provides controls for certain formatting capabilities (font changes and subscripting), while a third set is concerned with various ways of actually printing items. Finally, there is a command that permits escaping to a simple Lisp evaluation in the middle of a `PRINTOUT` form. The various commands are described below. The following examples give a general flavor of how `PRINTOUT` is used:

Example 1: Suppose you want to print out on the terminal the values of three variables, *X*, *Y*, and *Z*, separated by spaces and followed by a carriage return. This could be done by:

(`PRIN1` *X* *T*)

```
(SPACES 1 T)
(PRIN1 Y T)
(SPACES 1 T)
(PRIN1 Z T)
(TERPRI T)
```

or by the more concise PRINTOUT form:

```
(PRINTOUT T X , Y , Z T)
```

Here the first T specifies output to the terminal, the commas cause single spaces to be printed, and the final T specifies a TERPRI. The variable names are not recognized as special PRINTOUT commands, so they are printed using PRIN1 by default.

Example 2: Suppose the values of X and Y are to be pretty-printed lined up at position 10, preceded by identifying strings. If the output is to go to the primary output stream, you could write either:

```
(PRIN1 "X =")
(PRINTDEF X 10 T)
(TERPRI )
(PRIN1 "Y =")
(PRINTDEF Y 10 T)
(TERPRI)
```

or the equivalent:

```
(PRINTOUT NIL "X =" 10 .PPV X T
"Y =" 10 .PPV Y T)
```

Since strings are not recognized as special commands, "X =" is also printed with PRIN1 by default. The positive integer means TAB to position 10, where the .PPV command causes the value of X to be prettyprinted as a variable. By convention, special atoms used as PRINTOUT commands are prefixed with a period. The T causes a carriage return, so the Y information is printed on the next line.

Example 3. As a final example, suppose that the value of X is an integer and the value of Y is a floating-point number. X is to be printed right-flushed in a field of width 5 beginning at position 15, and Y is to be printed in a field of width 10 also starting at position 15 with 2 places to the right of the decimal point. Furthermore, suppose that the variable names are to appear in the font class named BOLDFONT and the values in font class SMALLFONT. The program in ordinary Interlisp that would accomplish these effects is too complicated to include here. With PRINTOUT, one could write:

```
(PRINTOUT NIL
.FONT BOLDFONT "X =" 15
.FONT SMALLFONT .I5 X T
```

```
.FONT BOLDFONT "Y =" 15
.FONT SMALLFONT .F10.2 Y T
.FONT BOLDFONT)
```

The `.FONT` commands do whatever is necessary to change the font on a multi-font output device. The `.I5` command sets up a `FIX` format for a call to the function `PRINTNUM` (see above) to print `X` in the desired format. The `.F10.2` specifies a `FLOAT` format for `PRINTNUM`.

Horizontal Spacing Commands

The horizontal spacing commands provide convenient ways of calling `TAB` and `SPACES`. In the following descriptions, *N* stands for a literal positive integer (*not* for a variable or expression whose value is an integer).

N (*N* a number) [PRINTOUT Command]

Used for absolute spacing. It results in a `TAB` to position *N* (literally, a `(TAB N)`). If the line is currently at position *N* or beyond, the file will be positioned at position *N* on the next line.

.TAB POS [PRINTOUT Command]

Specifies `TAB` to position (the value of) *POS*. This is one of several commands whose effect could be achieved by simply escaping to Lisp, and executing the corresponding form. It is provided as a separate command so that the `PRINTOUT` form is more concise and is prettyprinted more compactly. Note that `.TAB N` and *N*, where *N* is an integer, are equivalent.

.TAB0 POS [PRINTOUT Command]

Like `.TAB` except that it can result in zero spaces (i.e. the call to `TAB` specifies `MINSPACES=0`).

-N (*N* a number) [PRINTOUT Command]

Negative integers indicate relative (as opposed to absolute) spacing. Translates as `(SPACES |N|)`.

, [PRINTOUT Command]

” [PRINTOUT Command]

”” [PRINTOUT Command]

(1, 2 or 3 commas) Provides a short-hand way of specifying 1, 2 or 3 spaces, i.e., these commands are equivalent to `-1`, `-2`, and `-3`, respectively.

.SP DISTANCE [PRINTOUT Command]

Translates as `(SPACES DISTANCE)`. Note that `.SP N` and `-N`, where *N* is an integer, are equivalent.

Vertical Spacing Commands

Vertical spacing is obtained by calling TERPRI or printing form-feeds. The relevant commands are:

T [PRINTOUT Command]

Translates as (TERPRI), i.e., move to position 0 (the first column) of the next line. To print the letter T, use the string "T".

.SKIP LINES [PRINTOUT Command]

Equivalent to a sequence of *LINES* (TERPRI)'s. The .SKIP command allows for skipping large constant distances and for computing the distance to be skipped.

.PAGE [PRINTOUT Command]

Puts a form-feed (Control-L) out on the file. Care is taken to make sure that Interlisp's view of the current line position is correctly updated.

Special Formatting Controls

There are a small number of commands for invoking some of the formatting capabilities of multi-font output devices. The available commands are:

.FONT FONTSPEC [PRINTOUT Command]

Changes printing to the font *FONTSPEC*, which can be a font descriptor, a "font list" such as ' (MODERN 10), an image stream (coerced to its current font), or a windows (coerced to the current font of its display stream). The DSPFONT is changed permanently. See fonts (Chapter 27) for more information.

FONTSPEC may also be a positive integer *N*, which is taken as an abbreviated reference to the font class named FONTN (e.g. 1 => FONT1).

.SUP [PRINTOUT Command]

Specifies superscripting. All subsequent characters are printed above the base of the current line. Note that this is absolute, not relative: a .SUP following a .SUP is a no-op.

.SUB [PRINTOUT Command]

Specifies subscripting. Subsequent printing is below the base of the current line. As with superscripting, the effect is absolute.

.BASE [PRINTOUT Command]

Moves printing back to the base of the current line. Un-does a previous .SUP or .SUB; a no-op, if printing is currently at the base.

Printing Specifications

The value of any expression in a PRINTOUT form that is not recognized as a command itself or as a command argument is printed using PRIN1 by default. For example, title strings can be printed by simply including the string as a separate PRINTOUT command, and the values of variables and forms can be printed in much the same way. Note that a literal integer, say 51, cannot be printed by including it as a command, since it would be interpreted as a TAB; the desired effect can be obtained by using instead the string specification "51", or the form (QUOTE 51).

For those instances when PRIN1 is not appropriate, e.g., PRIN2 is required, or a list structures must be prettyprinted, the following commands are available:

.P2 *THING* [PRINTOUT Command]

Causes *THING* to be printed using PRIN2; translates as (PRIN2 *THING*).

.PPF *THING* [PRINTOUT Command]

Causes *THING* to be prettyprinted at the current line position via PRINTDEF (see Chapter 26). The call to PRINTDEF specifies that *THING* is to be printed as if it were part of a function definition. That is, SELECTQ, PROG, etc., receive special treatment.

.PPV *THING* [PRINTOUT Command]

Prettyprints *THING* as a variable; no special interpretation is given to SELECTQ, PROG, etc.

.PPFTL *THING* [PRINTOUT Command]

Like .PPF, but prettyprints *THING* as a *tail*, that is, without the initial and final parentheses if it is a list. Useful for prettyprinting sub-lists of a list whose other elements are formatted with other commands.

.PPVTL *THING* [PRINTOUT Command]

Like .PPV, but prettyprints *THING* as a tail.

Paragraph Format

Interlisp's prettyprint routines are designed to display the structure of expressions, but they are not really suitable for formatting unstructured text. If a list is to be printed as a textual paragraph, its internal structure is less important than controlling its left and right margins, and the indentation of its first line. The .PARA and .PARA2 commands allow these parameters to be conveniently specified.

.PARA *LMARG RMARG LIST* [PRINTOUT Command]

Prints *LIST* in paragraph format, using PRIN1. Translates as (PRINTPARA *LMARG RMARG LIST*) (see below).

Example: (PRINTOUT T 10 .PARA 5 -5 LST) will print the elements of LST as a paragraph with left margin at 5, right margin at (LINELENGTH)-5, and the first line indented to 10.

.PARA2 *LMARG RMARG LIST* [PRINTOUT Command]

Print as paragraph using PRIN2 instead of PRIN1. Translates as (PRINTPARA *LMARG RMARG LIST*).

Right-Flushing

Two commands are provided for printing simple expressions flushed-right against a specified line position, using the function FLUSHRIGHT (see below). They take into account the current position, the number of characters in the print-name of the expression, and the position the expression is to be flush against, and then print the appropriate number of spaces to achieve the desired effect. Note that this might entail going to a new line before printing. Note also that right-flushing of expressions longer than a line (e.g. a large list) makes little sense, and the appearance of the output is not guaranteed.

.FR *POS EXPR* [PRINTOUT Command]

Flush-right using PRIN1. The value of *POS* determines the position that the right end of *EXPR* will line up at. As with the horizontal spacing commands, a negative position number means $|POS|$ columns from the current position, a positive number specifies the position absolutely. *POS*=0 specifies the right-margin, i.e. is interpreted as (LINELENGTH).

.FR2 *POS EXPR* [PRINTOUT Command]

Flush-right using PRIN2 instead of PRIN1.

Centering

Commands for centering simple expressions between the current line position and another specified position are also available. As with right flushing, centering of large expressions is not guaranteed.

.CENTER *POS EXPR* [PRINTOUT Command]

Centers *EXPR* between the current line position and the position specified by the value of *POS*. A positive *POS* is an absolute position number, a negative *POS* specifies a position relative to the current position, and 0 indicates the right-margin. Uses PRIN1 for printing.

.CENTER2 *POS EXPR* [PRINTOUT Command]

Centers using PRIN2 instead of PRIN1.

Numbering

The following commands provide FORTRAN-like formatting capabilities for integer and floating-point numbers. Each command specifies a printing format and a number to be printed. The format specification translates into a format-list for the function `PRINTNUM`.

.I *FORMAT NUMBER* [PRINTOUT Command]

Specifies integer printing. Translates as a call to the function `PRINTNUM` with a `FIX` format-list constructed from *FORMAT*. The atomic format is broken apart at internal periods to form the format-list. For example, `.I5.8.T` yields the format-list `(FIX 5 8 T)`, and the command sequence `(PRINTOUT T .I5.8.T FOO)` translates as `(PRINTNUM '(FIX 5 8 T) FOO)`. This expression causes the value of `FOO` to be printed in radix 8 right-flushed in a field of width 5, with 0's used for padding on the left. Internal `NIL`'s in the format specification may be omitted, e.g., the commands `.I5..T` and `.I5.NIL.T` are equivalent.

The format specification `.I1` is often useful for forcing a number to be printed in radix 10 (but not otherwise specially formatted), independent of the current setting of `RADIX`.

.F *FORMAT NUMBER* [PRINTOUT Command]

Specifies floating-number printing. Like the `.I` format command, except translates with a `FLOAT` format-list.

.N *FORMAT NUMBER* [PRINTOUT Command]

The `.I` and `.F` commands specify calls to `PRINTNUM` with quoted format specifications. The `.N` command translates as `(PRINTNUM FORMAT NUMBER)`, i.e., it permits the format to be the value of some expression. Note that, unlike the `.I` and `.F` commands, *FORMAT* is a separate element in the command list, not part of an atom beginning with `.N`.

Escaping to Lisp

There are many reasons for taking control away from `PRINTOUT` in the middle of a long printing expression. Common situations involve temporary changes to system printing parameters (e.g. `LINELENGTH`), conditional printing (e.g. print `FOO` only if `FILE` is `T`), or lower-level iterative printing within a higher-level print specification.

*FORM* [PRINTOUT Command]

The escape command. *FORM* is an arbitrary Lisp expression that is evaluated within the context established by the `PRINTOUT` form, i.e., *FORM* can assume that the primary output stream has been set to be the *FILE* argument to `PRINTOUT`. Note that nothing is done with the *value* of *FORM*; any printing desired is accomplished by *FORM* itself, and the value is discarded.

Note: Although `PRINTOUT` logically encloses its translation in a `RESETFORM` (Chapter 14) to change the primary output file to the *FILE* argument (if non-`NIL`), in most

cases it can actually pass *FILE* (or a locally bound variable if *FILE* is a non-trivial expression) to each printing function. Thus, the RESETFORM is only generated when the # command is used, or user-defined commands (below) are used. If many such occur in repeated PRINTOUT forms, it may be more efficient to embed them all in a single RESETFORM which changes the primary output file, and then specify *FILE*=NIL in the PRINTOUT expressions themselves.

User-Defined Commands

The collection of commands and options outlined above is aimed at fulfilling all common printing needs. However, certain applications might have other, more specialized printing idioms, so a facility is provided whereby you can define new commands. This is done by adding entries to the global list PRINTOUTMACROS to define how the new commands are to be translated.

PRINTOUTMACROS

[Variable]

PRINTOUTMACROS is an association-list whose elements are of the form (COMM FN). Whenever COMM appears in command position in the sequence of PRINTOUT commands (as opposed to an argument position of another command), FN is applied to the tail of the command-list (including the command).

After inspecting as much of the tail as necessary, the function must return a list whose CAR is the translation of the user-defined command and its arguments, and whose CDR is the list of commands still remaining to be translated in the normal way.

For example, suppose you want to define a command "?", which will cause its single argument to be printed with PRIN1 only if it is not NIL. This can be done by entering (? ?TRAN) on PRINTOUTMACROS, and defining the function ?TRAN as follows:

```
(DEFINEQ (?TRAN (COMS)
  (CONS
    (SUBST (CADR COMS) 'ARG
      ' (PROG ((TEMP ARG))
        (COND (TEMP (PRIN1 TEMP))))))
    (CDDR COMS))]
```

Note that ?TRAN does not do any printing itself; it returns a form which, when evaluated in the proper context, will perform the desired action. This form should direct all printing to the primary output file.

Special Printing Functions

The paragraph printing commands are translated into calls on the function PRINTPARA, which may also be called directly:

(PRINTPARA LMARG RMARG LIST P2FLAG PARENFLAG FILE)

[Function]

Prints LIST on FILE in line-filled paragraph format with its first element beginning at the current line position and ending at or before RMARG, and with subsequent lines appearing

between *LMARG* and *RMARG*. If *P2FLAG* is non-NIL, prints elements using PRIN2, otherwise PRIN1. If *PARENFLAG* is non-NIL, then parentheses will be printed around the elements of *LIST*.

If *LMARG* is zero or positive, it is interpreted as an absolute column position. If it is negative, then the left margin will be at $|LMARG| + (POSITION)$. If *LMARG*=NIL, the left margin will be at $(POSITION)$, and the paragraph will appear in block format.

If *RMARG* is positive, it also is an absolute column position (which may be greater than the current $(LINELENGTH)$). Otherwise, it is interpreted as relative to $(LINELENGTH)$, i.e., the right margin will be at $(LINELENGTH) + |RMARG|$. Example: (TAB 10) (PRINTPARA 5 -5 LST T) will PRIN2 the elements of LST in a paragraph with the first line beginning at column 10, subsequent lines beginning at column 5, and all lines ending at or before $(LINELENGTH) - 5$.

The current $(LINELENGTH)$ is unaffected by PRINTPARA, and upon completion, *FILE* will be positioned immediately after the last character of the last item of *LIST*. PRINTPARA is a no-op if *LIST* is not a list.

The right-flushing and centering commands translate as calls to the function FLUSHRIGHT:

(**FLUSHRIGHT** *POS X MIN P2FLAG CENTERFLAG FILE*) [Function]

If *CENTERFLAG*=NIL, prints *X* right-flushed against position *POS* on *FILE*; otherwise, centers *X* between the current line position and *POS*. Makes sure that it spaces over at least *MIN* spaces before printing by doing a TERPRI if necessary; *MIN*=NIL is equivalent to *MIN*=1. A positive *POS* indicates an absolute position, while a negative *POS* signifies the position which is $|POS|$ to the right of the current line position. *POS*=0 is interpreted as $(LINELENGTH)$, the right margin.

READFILE and WRITEFILE

For those applications where you simply want to simply read all of the expressions on a file, and not evaluate them, the function READFILE is available:

(**READFILE** *FILE RDTBL ENDTOKEN*) [NoSpread Function]

Reads successive expressions from file using READ (with read table *RDTBL*) until the single litatom *ENDTOKEN* is read, or an end of file encountered. Returns a list of these expressions.

If *RDTBL* is not specified, it defaults to *FILERDTBL*. If *ENDTOKEN* is not specified, it defaults to the litatom STOP.

(**WRITEFILE** *X FILE*) [Function]

Writes a date expression onto *FILE*, followed by successive expressions from *X*, using *FILERDTBL* as a read table. If *X* is atomic, its value is used. If *FILE* is not open, it is

opened. If *FILE* is a list, (*CAR FILE*) is used and the file is left opened. Otherwise, when *X* is finished, the litatom *STOP* is printed on *FILE* and it is closed. Returns *FILE*.

(**ENDFILE** *FILE*)

[Function]

Prints *STOP* on *FILE* and closes it.

Read Tables

Many Interlisp input functions treat certain characters in special ways. For example, *READ* recognizes that the right and left parenthesis characters are used to specify list structures, and that the quote character is used to delimit text strings. The Interlisp input and (to a certain extent) output routines are table driven by read tables. Read tables are objects that specify the syntactic properties of characters for input routines. Since the input routines parse character sequences into objects, the read table in use determines which sequences are recognized as literal atoms, strings, list structures, etc.

Most Interlisp input functions take an optional read table argument, which specifies the read table to use when reading an expression. If *NIL* is given as the read table, the "primary read table" is used. If *T* is specified, the system terminal read table is used. Some functions will also accept the atom *ORIG* (not the value of *ORIG*) as indicating the "original" system read table. Some output functions also take a read table argument. For example, *PRIN2* prints an expression so that it would be read in correctly using a given read table.

The Interlisp-D system uses the following read tables: *T* for input/output from terminals, the value of *FILERDTBL* for input/output from files, the value of *EDITRDTBL* for input from terminals while in the tty-based editor, the value of *DEDITRDTBL* for input from terminals while in the display-based editor, and the value of *CODERDTBL* for input/output from compiled files. These five read tables are initially copies of the *ORIG* read table, with changes made to some of them to provide read macros that are specific to terminal input or file input. Using the functions described below, you may further change, reset, or copy these tables. However, in the case of *FILERDTBL* and *CODERDTBL*, you are cautioned that changing these tables may prevent the system from being able to read files made with the original tables, or prevent users possessing only the standard tables from reading files made using the modified tables.

You can also create new read tables, and either explicitly pass them to input/output functions as arguments, or install them as the primary read table, via *SETREADTABLE*, and then not specify a *RDTBL* argument, i.e., use *NIL*.

Read Table Functions

(**READTABLEP** *RDTBL*)

[Function]

Returns *RDTBL* if *RDTBL* is a real read table (not *T* or *ORIG*), otherwise *NIL*.

(**GETREADTABLE** *RD_TBL*) [Function]

If *RD_TBL*=NIL, returns the primary read table. If *RD_TBL*=T, returns the system terminal read table. If *RD_TBL* is a real read table, returns *RD_TBL*. Otherwise, generates an ILLEGAL READTABLE error.

(**SETREADTABLE** *RD_TBL FLG*) [Function]

Sets the primary read table to *RD_TBL*. If *FLG*=T, SETREADTABLE sets the system terminal read table, T. Note that you can reset the other system read tables with SETQ, e.g., (SETQ FILERD_TBL (GETREADTABLE)).

Generates an ILLEGAL READTABLE error if *RD_TBL* is not NIL, T, or a real read table. Returns the previous setting of the primary read table, so SETREADTABLE is suitable for use with RESETFORM (Chapter 14).

(**COPYREADTABLE** *RD_TBL*) [Function]

Returns a copy of *RD_TBL*. *RD_TBL* can be a real read table, NIL, T, or ORIG (in which case COPYREADTABLE returns a copy of the *original* system read table), otherwise COPYREADTABLE generates an ILLEGAL READTABLE error.

Note that COPYREADTABLE is the only function that *creates* a read table.

(**RESETREADTABLE** *RD_TBL FROM*) [Function]

Copies (smashes) *FROM* into *RD_TBL*. *FROM* and *RD_TBL* can be NIL, T, or a real read table. In addition, *FROM* can be ORIG, meaning use the system's original read table.

Syntax Classes

A read table is an object that contains information about the "syntax class" of each character. There are nine basic syntax classes: LEFTPAREN, RIGHTPAREN, LEFTBRACKET, RIGHTBRACKET, STRINGDELIM, ESCAPE, BREAKCHAR, SEPRCHAR, and OTHER, each associated with a primitive syntactic property. In addition, there is an unlimited assortment of user-defined syntax classes, known as "read macros". The basic syntax classes are interpreted as follows:

LEFTPAREN (normally left parenthesis) Begins list structure.

RIGHTPAREN (normally right parenthesis) Ends list structure.

LEFTBRACKET (normally left bracket) Begins list structure. Also matches RIGHTBRACKET characters.

RIGHTBRACKET (normally left bracket) Ends list structure. Can close an arbitrary numbers of LEFTPAREN lists, back to the last LEFTBRACKET.

STRINGDELIM (normally double quote) Begins and ends text strings. Within the string, all characters except for the one(s) with class ESCAPE are treated as ordinary, i.e., interpreted as if they were of syntax class OTHER. To include the string delimiter inside a string, prefix it with the ESCAPE character.

ESCAPE	(normally percent sign) Inhibits any special interpretation of the next character, i.e., the next character is interpreted to be of class OTHER, independent of its normal syntax class.
BREAKCHAR	(None initially) Is a break character, i.e., delimits atoms, but is otherwise an ordinary character.
SEPRCHAR	(space, carriage return, etc.) Delimits atoms, and is otherwise ignored.
OTHER	Characters that are not otherwise special belong to the class OTHER.

Characters of syntax class LEFTPAREN, RIGHTPAREN, LEFTBRACKET, RIGHTBRACKET, and STRINGDELIM are all *break* characters. That is, in addition to their interpretation as delimiting list or string structures, they also terminate the reading of an atom. Characters of class BREAKCHAR serve *only* to terminate atoms, with no other special meaning. In addition, if a break character is the first non-separator encountered by RATOM, it is read as a one-character atom. In order for a break character to be included in an atom, it must be preceded by the ESCAPE character.

Characters of class SEPRCHAR also terminate atoms, but are otherwise completely ignored; they can be thought of as logically spaces. As with break characters, they must be preceded by the ESCAPE character in order to appear in an atom.

For example, if \$ were a break character and * a separator character, the input stream ABC**DEF\$GH*\$ would be read by six calls to RATOM returning respectively ABC, DEF, \$, GH, \$, \$.

Although normally there is only one character in a read table having each of the list- and string-delimiting syntax classes (such as LEFTPAREN), it is perfectly acceptable for any character to have any syntax class, and for more than one to have the same class.

Note that a "syntax class" is an abstraction: there is no object referencing a collection of characters called a *syntax class*. Instead, a read table provides the *association* between a character and its syntax class, and the input/output routines enforce the abstraction by using read tables to drive the parsing.

The functions below are used to obtain and set the syntax class of a character in a read table. *CH* can either be a character code (a integer), or a character (a single-character atom). Single-digit integers are interpreted as character codes, rather than as characters. For example, 1 indicates Control-A, and 49 indicates the *character* 1. Note that *CH* can be a full sixteen-bit NS character (see Chapter 2).

Note: Terminal tables, described in Chapter 30, also associate characters with syntax classes, and they can also be manipulated with the functions below. The set of read table and terminal table syntax classes are disjoint, so there is never any ambiguity about which type of table is being referred to.

(GETSYNTAX *CH* *TABLE*)

[Function]

Returns the syntax class of *CH*, a character or a character code, with respect to *TABLE*. *TABLE* can be NIL, T, ORIG, or a real read table or terminal table.

CH can also be a syntax class, in which case GETSYNTAX returns a list of the character codes in *TABLE* that have that syntax class.

(**SETSYNTAX** *CHAR CLASS TABLE*) [Function]

Sets the syntax class of *CHAR*, a character or character code, in *TABLE*. *TABLE* can be either NIL, T, or a real read table or terminal table. SETSYNTAX returns the previous syntax class of *CHAR*. *CLASS* can be any one of the following:

- The name of one of the basic syntax classes.
- A list, which is interpreted as a read macro (see below).
- NIL, T, ORIG, or a real read table or terminal table, which means to give *CHAR* the syntax class it has in the table indicated by *CLASS*. For example, (SETSYNTAX '*%*(' 'ORIG *TABLE*) gives the left parenthesis character in *TABLE* the same syntax class that it has in the original system read table.
- A character code or character, which means to give *CHAR* the same syntax class as the character *CHAR* in *TABLE*. For example, (SETSYNTAX '{ '*%*[*TABLE*) gives the left brace character the same syntax class as the left bracket.

(**SYNTAXP** *CODE CLASS TABLE*) [Function]

CODE is a character code; *TABLE* is NIL, T, or a real read table or terminal table. Returns T if *CODE* has the syntax class *CLASS* in *TABLE*; NIL otherwise.

CLASS can also be a read macro type (MACRO, SPLICE, INFIX), or a read macro option (FIRST, IMMEDIATE, etc.), in which case SYNTAXP returns T if the syntax class is a read macro with the specified property.

SYNTAXP will *not* accept a character as an argument, only a character *code*.

For convenience in use with SYNTAXP, the atom BREAK may be used to refer to *all* break characters, i.e., it is the union of LEFTPAREN, RIGHTPAREN, LEFTBRACKET, RIGHTBRACKET, STRINGDELIM, and BREAKCHAR. For purely symmetrical reasons, the atom SEPR corresponds to all separator characters. However, since the only separator characters are those that also appear in SEPRCHAR, SEPR and SEPRCHAR are equivalent.

Note that GETSYNTAX never returns BREAK or SEPR as a value although SETSYNTAX and SYNTAXP accept them as arguments. Instead, GETSYNTAX returns one of the disjoint basic syntax classes that comprise BREAK. BREAK as an argument to SETSYNTAX is interpreted to mean BREAKCHAR if the character is not already of one of the BREAK classes. Thus, if *%*(is of class LEFTPAREN, then (SETSYNTAX '*%*(' 'BREAK) doesn't do anything, since *%*(is already a break character, but (SETSYNTAX '*%*(' 'BREAKCHAR) means make *%*(be *just* a break character, and therefore disables the LEFTPAREN function of *%*(. Similarly, if one of the format characters is disabled completely, e.g., by (SETSYNTAX '*%*(' 'OTHER), then (SETSYNTAX '*%*(' 'BREAK) would make *%*(be *only* a break character; it would *not* restore *%*(as LEFTPAREN.

The following functions provide a way of collectively accessing and setting the separator and break characters in a read table:

(**GETSEPR** *RD_TBL*) [Function]

Returns a list of separator character codes in *RD_TBL*. Equivalent to (GETSYNTAX 'SEPR *RD_TBL*).

(**GETBRK** *RD_TBL*) [Function]

Returns a list of break character codes in *RD_TBL*. Equivalent to (GETSYNTAX 'BREAK *RD_TBL*).

(**SETSEPR** *LST FLG RD_TBL*) [Function]

Sets or removes the separator characters for *RD_TBL*. *LST* is a list of characters or character codes. *FLG* determines the action of SETSEPR as follows: If *FLG*=NIL, makes *RD_TBL* have exactly the elements of *LST* as separators, discarding from *RD_TBL* any old separator characters not in *LST*. If *FLG*=0, removes from *RD_TBL* as separator characters all elements of *LST*. This provides an "UNSETSEPR". If *FLG*=1, makes each of the characters in *LST* be a separator in *RD_TBL*.

If *LST*=T, the separator characters are reset to be those in the system's read table for terminals, regardless of the value of *FLG*, i.e., (SETSEPR T) is equivalent to (SETSEPR (GETSEPR T)). If *RD_TBL* is T, then the characters are reset to those in the original system table.

Returns NIL.

(**SETBRK** *LST FLG RD_TBL*) [Function]

Sets the break characters for *RD_TBL*. Similar to SETSEPR.

As with SETSYNTAX to the BREAK class, if any of the list- or string-delimiting break characters are disabled by an appropriate SETBRK (or by making it be a separator character), its special action for READ will *not* be restored by simply making it be a break character again with SETBRK. However, making these characters be break characters when they already *are* will have no effect.

The action of the ESCAPE character (normally %) is not affected by SETSEPR or SETBRK. It can be disabled by setting its syntax to the class OTHER, and other characters can be used for escape on input by assigning them the class ESCAPE. As of this writing, however, there is no way to change the output escape character; it is "hardwired" as %. That is, on output, characters of special syntax that need to be preceded by the ESCAPE character will always be preceded by %, independent of the syntax of % or which, if any characters, have syntax ESCAPE.

The following function can be used for defeating the action of the ESCAPE character or characters:

(**ESCAPE** *FLG* *RDTBL*)

[Function]

If *FLG*=NIL, makes characters of class **ESCAPE** behave like characters of class **OTHER** on input. Normal setting is (**ESCAPE** **T**). **ESCAPE** returns the previous setting.

Read Macros

This is a description of the OLD-INTERLISP-T read macros. Read macros are user-defined syntax classes that can cause complex operations when certain characters are read. Read macro characters are defined by specifying as a syntax class an expression of the form:

(*TYPE* *OPTION* ... *OPTION* *FN*)

where *TYPE* is one of **MACRO**, **SPLICE**, or **INFIX**, and *FN* is the name of a function or a lambda expression. Whenever **READ** encounters a read macro character, it calls the associated function, giving it as arguments the input stream and read table being used for that call to **READ**. The interpretation of the value returned depends on the type of read macro:

MACRO This is the simplest type of read macro. The result returned from the macro is treated as the expression to be read, instead of the read macro character. Often the macro reads more input itself. For example, in order to cause ~*EXPR* to be read as (**NOT** *EXPR*), one could define ~ as the read macro:

```
(MACRO (LAMBDA (FL RDTBL)  
  (LIST 'NOT (READ FL RDTBL))
```

SPLICE The result (which should be a list or **NIL**) is spliced into the input using **NCONC**. For example, if \$ is defined by the read macro:

```
(SPLICE (LAMBDA NIL (APPEND FOO)))
```

and the value of **FOO** is (**A** **B** **C**), then when you input (**X** \$ **Y**), the result will be (**X** **A** **B** **C** **Y**).

INFIX The associated function is called with a third argument, which is a list, in **TCONC** format (Chapter 3), of what has been read at the current level of list nesting. The function's value is taken as a new **TCONC** list which replaces the old one. For example, the infix operator + could be defined by the read macro:

```
(INFIX (LAMBDA (FL RDTBL Z)  
  (RPLACA (CDR Z)  
    (LIST (QUOTE IPLUS)  
      (CADR Z)  
      (READ FL RDTBL)))  
  Z))
```

If an **INFIX** read macro character is encountered *not* in a list, the third argument to its associated function is **NIL**. If the function returns **NIL**, the read macro character is essentially ignored and reading continues. Otherwise, if the function returns a **TCONC** list of one element, that element is the value of

the READ. If it returns a TCONC list of more than one element, the list is the value of the READ.

The specification for a read macro character can be augmented to specify various options *OPTION* . . . *OPTION* , e.g., (MACRO FIRST IMMEDIATE FN). The following three disjoint options specify when the read macro character is to be effective:

- ALWAYS The default. The read macro character is always effective (except when preceded by the % character), and is a break character, i.e., a member of (GETSYNTAX 'BREAK RDTBL).
- FIRST The character is interpreted as a read macro character *only* when it is the first character seen after a break or separator character; in all other situations, the character is treated as having class OTHER. The read macro character is *not* a break character. For example, the quote character is a FIRST read macro character, so that DON'T is read as the single atom DON'T, rather than as DON followed by (QUOTE T).
- ALONE The read macro character is *not* a break character, and is interpreted as a read macro character only when the character would have been read as a separate atom if it were not a read macro character, i.e., when its immediate neighbors are both break or separator characters.

Making a FIRST or ALONE read macro character be a break character (with SETBRK) disables the read macro interpretation, i.e., converts it to syntax class BREAKCHAR. Making an ALWAYS read macro character be a break character is a no-op.

The following two disjoint options control whether the read macro character is to be protected by the ESCAPE character on output when a litatom containing the character is printed:

- ESCQUOTE or ESC The default. When printed with PRIN2, the read macro character will be preceded by the output escape character (%) as needed to permit the atom containing it to be read correctly. Note that for FIRST macros, this means that the character need be quoted only when it is the first character of the atom.
- NOESCQUOTE or NOESC The read macro character will always be printed without an escape. For example, the ? read macro in the T read table is a NOESCQUOTE character. Unless you are very careful what you are doing, read macro characters in FILERDTBL should never be NOESCQUOTE, since symbols that happen to contain the read macro character will not read back in correctly.

The following two disjoint options control when the macro's function is actually executed:

- IMMEDIATE or IMMED The read macro character is immediately activated, i.e., the current line is terminated, as if an EOL had been typed, a carriage-return line-feed is printed, and the entire line (including the macro character) is passed to the input function.

IMMEDIATE read macro characters enable you to specify a character that will take effect immediately, as soon as it is encountered in the input, rather than waiting for the line to be terminated. Note that this is not necessarily as soon as the character is *typed*. Characters that cause action as soon as they are typed are interrupt characters (see Chapter 30).

Note that since an IMMEDIATE macro causes any input before it to be sent to the reader, characters typed before an IMMEDIATE read macro character cannot be erased by Control-A or Control-Q once the IMMEDIATE character has been typed, since they have already passed through the line buffer. However, an INFIX read macro can still alter some of what has been typed earlier, via its third argument.

NONIMMEDIATE or NONIMMED The default. The read macro character is a normal character with respect to the line buffering, and so will not be activated until a carriage-return or matching right parenthesis or bracket is seen.

Making a read macro character be both ALONE and IMMEDIATE is a contradiction, since ALONE requires that the next character be input in order to see if it is a break or separator character. Thus, ALONE read macros are always NONIMMEDIATE, regardless of whether or not IMMEDIATE is specified.

Read macro characters can be "nested". For example, if = is defined by

```
(MACRO (LAMBDA (FL RDTBL)
  (EVAL (READ FL RDTBL))))
```

and ! is defined by

```
(SPLICE (LAMBDA (FL RDTBL)
  (READ FL RDTBL)))
```

then if the value of FOO is (A B C), and (X =FOO Y) is input, (X (A B C) Y) will be returned. If (X !=FOO Y) is input, (X A B C Y) will be returned.

Note: If a read macro's function calls READ, and the READ returns NIL, the function cannot distinguish the case where a RIGHTPAREN or RIGHTBRACKET followed the read macro character, (e.g. "(A B ')", from the case where the atom NIL (or "()") actually appeared. In Interlisp-D, a READ inside of a read macro when the next input character is a RIGHTPAREN or RIGHTBRACKET reads the character and returns NIL, just as if the READ had not occurred inside a read macro.

If a call to READ from within a read macro encounters an unmatched RIGHTBRACKET *within* a list, the bracket is simply put back into the buffer to be read (again) at the higher level. Thus, inputting an expression such as (A B ' (C D] works correctly.

(INREADMACROP)

[Function]

Returns NIL if currently *not* under a read macro function, otherwise the number of unmatched left parentheses or brackets.

(READMACROS *FLG* *RDTBL*)

[Function]

If *FLG*=NIL, turns off action of read macros in read table *RDTBL*. If *FLG*=T, turns them on. Returns previous setting.

The following read macros are standardly defined in Interlisp in the T and EDITRDTBL read tables:

' (single-quote) Returns the next expression, wrapped in a call to QUOTE; e.g., 'FOO reads as (QUOTE FOO). The macro is defined as a FIRST read macro, so that the quote character has no effect in the middle of a symbol. The macro is also ignored if the quote character is immediately followed by a separator character.

Control-Y Defined in T and EDITRDTBL. Returns the result of evaluating the next expression. For example, if the value of FOO is (A B), then (LIST 1 control-YFOO 2) is read as (LIST 1 (A B) 2). Note that no structure is copied; the third element of that input expression is still EQ to the value of FOO. Control-Y can thus be used to read structures that ordinarily have no read syntax. For example, the value returned from reading (KEY1 Control-Y (ARRAY 10)) has an array as its second element. Control-Y can be thought of as an "un-quote" character. The choice of character to perform this function is changeable with SETTERMCHARS (see Chapter 16).

` (backquote) Backquote makes it easier to write programs to construct complex data structures. Backquote is like quote, except that within the backquoted expression, forms can be evaluated. The general idea is that the backquoted expression is a "template" containing some constant parts (as with a quoted form) and some parts to be filled in by evaluating something. Unlike with control-Y, however, the evaluation occurs not at the time the form is read, but at the time the backquoted expression is evaluated. That is, the backquote macro returns an expression which, when evaluated, produces the desired structure.

Within the backquoted expression, the character "," (comma) introduces a form to be evaluated. The value of a form preceded by ",@" is to be spliced in, using APPEND. If it is permissible to destroy the list being spliced in (i.e., NCONC may be used in the translation), then ", ." can be used instead of ",@".

For example, if the value of FOO is (1 2 3 4), then the form

`(A , (CAR FOO) ,@(CDDR FOO) D E)

evaluates to (A 1 3 4 D E); it is logically equivalent to writing

(CONS 'A

```
(CONS (CAR FOO)
      (APPEND (CDDR FOO) ' (D E) ) )
```

Backquote is particularly useful for writing macros. For example, the body of a macro that refers to X as the macro's argument list might be

```
`(COND
  ((FIXP , (CAR X))
   , (CADR X))
  (T . , (CDDR X)))
```

which is equivalent to writing

```
(LIST 'COND
      (LIST (LIST 'FIXP (CAR X))
            (CADR X))
      (CONS 'T (CDDR X)))
```

Note that comma does *not* have any special meaning outside of a backquote context.

For users without a backquote character on their keyboards, backquote can also be written as | ' (vertical-bar, quote).

? Implements the ?= command for on-line help regarding the function currently being "called" in the typein (see Chapter 26).

| (vertical bar) When followed by an end of line, tab or space, | is ignored, i.e., treated as a separator character, enabling the editor's `CHANGECHAR` feature (see Chapter 26). Otherwise it is a "dispatching" read macro whose meaning depends on the character(s) following it. The following are currently defined:

' (quote) -- A synonym for backquote.

. (period) -- Returns the evaluation of the next expression, i.e., this is a synonym for Control-Y.

, (comma) -- Returns the evaluation of the next expression *at load time*, i.e., the following expression is quoted in such a manner that the compiler treats it as a literal whose value is not determined until the compiled expression is loaded.

O or o (the letter O) -- Treats the next number as octal, i.e., reads it in radix 8. For example, |o12 = 10 (decimal).

B or b -- Treats the next number as binary, i.e., reads it in radix 2. For example, |b101 = 5 (decimal).

X or x -- Treats the next number as hexadecimal, i.e., reads it in radix 16. The uppercase letters A through F are used as the digits after 9. For example, |x1A = 26 (decimal).

R or r -- Reads the next number in the radix specified by the (decimal) number that appears between the | and the R. When inputting a number in a

radix above ten, the upper-case letters A through Z can be used as the digits after 9 (but there is no digit above Z, so it is not possible to type all base-99 digits). For example, `|3r120` reads 120 in radix 3, returning 15.

`(, {, ^` -- Used internally by `HPRINT` and `HREAD` (see above) to print and read unusual expressions.

The dispatching characters that are letters can appear in either upper- or lowercase.

25. USER/ INPUT/OUTPUT PACKAGES

Interlisp-D can perform input/output operations on a large variety of physical devices.

This chapter presents a number of packages that have been developed for displaying and allowing the user to enter information. These packages are used to implement the user interface of many system facilities.

`INSPECT` (see the `INSPECT` section below) provides a window-based facility for displaying and changing the fields of a data object.

`PROMPTFORWARD` (see the `PROMPTFORWARD` section below) is a function used for entering a simple string of characters. Basic editing and prompting facilities are provided.

`ASKUSER` (see the `ASKUSER` section below) provides a more complicated prompting and answering facility, allowing a series of questions to be printed. Prompts and argument completion are supported.

`TTYIN` (see the `TTYIN Display Typein Editor` section below) is a display typein editor, that provides complex text editing facilities when entering an input line.

`PRETTYPRINT` (see the `Prettyprint` section below) is used for printing function definitions and other list structures, using multiple fonts and indenting lines to show the structure of the list.

Inspector

The Inspector provides a display-oriented facility for looking at and changing arbitrary Interlisp-D data structures. The inspector can be used to inspect all user datatypes and many system datatypes (although some objects such as numbers have no inspectable structure). The inspector displays the field names and values of an arbitrary object in a window that allows setting of the properties and further inspection of the values. This latter feature makes it possible to "walk" around all of the data structures in the system at the touch of a button. In addition, the inspector is integrated with the break package to allow inspection of any object on the stack and with the display and teletype structural editors to allow the editors to be used to "inspect" list structures and the inspector to "edit" datatypes.

The underlying mechanisms of the data inspector have been designed to allow their use as specialized editors in user applications. This functionality is described at the end of this section.

Note: Currently, the inspector does not have UNDOing. Also, variables whose values are changed will not be marked as such.

Calling the Inspector

INTERLISP-D REFERENCE MANUAL

There are several ways to open an inspect window onto an object. In addition to calling `INSPECT` directly (below), the inspector can also be called by buttoning an Inspect command inside an existing inspector window. Finally, if a non-list is edited with `EDITDEF` (see Chapter 17), the inspector is called. This also causes the inspector to be called by the `Dedit` command from the display editor or the `EV` command from the teletype editor if the selected piece of structure is a non-list.

(**INSPECT** *OBJECT* *ASTYPE* *WHERE*) [Function]

Creates an inspect window onto *OBJECT*. If *ASTYPE* is given, it will be taken as the record type of *OBJECT*. This allows records to be inspected with their property names. If *ASTYPE* is `NIL`, the data type of *OBJECT* will be used to determine its property names in the inspect window.

WHERE specifies the location of the inspect window. If *WHERE* is `NIL`, the user will be prompted for a location. If *WHERE* is a window, it will be used as the inspect window. If *WHERE* is a region, the inspect window will be created in that region of the screen. If *WHERE* is a position, the inspect window will have its lower left corner at that position on the screen.

`INSPECT` returns the inspect window onto *OBJECT*, or `NIL` if no inspection took place.

(**INSPECTCODE** *FN* *WHERE* — — —) [Function]

Opens a window and displays the compiled code of the function *FN* using `PRINTCODE`. The window is scrollable.

WHERE determines where the window should appear. It can be a position, a region, or a window. If `NIL`, the user is prompted to specify the position of the window.

Note: If the Tedit library package is loaded, `INSPECTCODE` uses it to create the code inspector window. Also, if `INSPECTCODE` is called to inspect the frame name in a break window (see Chapter 14), the location in the code that the frame's PC indicates it was executing at the time is highlighted.

Multiple Ways of Inspecting

For some datatypes there is more than one aspect that is of interest or more than one method of inspecting the object. In these cases, the inspector will bring up a menu of the possibilities and wait for the user to select one.

If the object is a litatom, the commands are the types for which the litatom has definitions as determined by `HASDEF`. Some typical commands are:

FNS Edit the definition of the selected litatom.



VARS Inspect the value.

PROPS Inspect the property list.

If the object is a list, there will be choice of how to inspect the list:

USER I/O PACKAGES

Inspect	Opens an inspect window in which the properties are numbers and the values are the elements of the list.
TtyEdit	Calls the teletype list structure editor on the list (see Chapter 16).
DisplayEdit	Calls the DEdit display editor on the list (see Chapter 16).
As a PLIST	Inspects the list as a property list, if the list is in property list form: ((PROP VAL) ... (PROP VAL)) .
As an ALIST	Inspects the list as an association-list, if the list is in ASSOC list form: (PROP VAL ... PROP VAL) .
As a record	Brings up a submenu with all of the RECORDs in the system and inspect the list with the one chosen.
As a "record type"	Inspects the list as the record of the type named in its CAR, if the CAR of the list is the name of a TYPE-RECORD (see Chapter 8).

If the object is a bitmap, the choice is between inspecting the bitmap's contents with the bitmap editor (EDITBM) or inspecting the bitmap's fields.

Other datatypes may include multiple methods for inspecting objects of that type.

Inspect Windows

An inspect window displays two columns of values. The lefthand column lists the property names of the structure being inspected. The righthand column contains the values of the properties named on the left. For variable length data such as lists and arrays, the "property names" are numbers from 1 to the length of the inspected item and the values are the corresponding elements. For arrays, the property names are the array element numbers and the values are the corresponding elements of the array.

For large lists or arrays, or datatypes with many fields, the initial window may be too small to contain all of them. In these cases, the unseen elements can be scrolled into view (from the bottom) or the window can be reshaped to increase its size.

In an inspect window, the LEFT button is used to select things, the MIDDLE button to invoke commands that apply to the selected item. Any property or value can be selected by pointing the cursor directly at the text representing it, and clicking the LEFT button. There is one selected item per window and it is marked by having its surrounding box inverted.

The options offered by the MIDDLE button depend on whether the selection is a property or a value. If the selected item is a value, the options provide different ways of inspecting the selected structure. The exact commands that are given depend on the type of the value. An example of the menu you may see is:

INTERLISP-D REFERENCE MANUAL



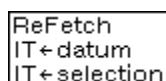
```
DisplayEdit
TtyEdit
Inspect
As a record
As a PLIST
```

If the selected item is a property name, the command `SET` will appear. If selected, the user will be asked to type in an expression, and the selected property will be set to the result of evaluating the read form. The evaluation of the read form and the replacement of the selected item property will appear as their own history events and are individually undoable. Properties of system datatypes cannot be set. (There are often consistency requirements which can be inadvertently violated in ways that crash the system. This may be true of some user datatypes as well, however the system doesn't know which ones. Users are advised to exercise caution.)

It is possible to copy-select property names or values out of an inspect window. Litatoms, numbers and strings are copied as they are displayed. Unprintable objects (such as bitmaps, etc.) come out as an appropriate system expression, such that if it is evaluated, the object is re-created.

Inspect Window Commands

By pressing the `MIDDLE` button in the title of the inspect window, a menu of commands that apply to the inspect window is brought up:



```
ReFetch
IT←datum
IT←selection
```

ReFetch [Inspect Window Command]

An inspect window is not automatically updated when the structure it is inspecting is changed. The `ReFetch` command will refetch and redisplay all of the fields of the object being inspected in the inspect window.

IT←datum [Inspect Window Command]

Sets the variable `IT` to object being inspected in the inspect window.

IT←selection [Inspect Window Command]

Sets the variable `IT` to the property name or value currently selected in the inspect window.

Interaction With Break Windows

The break window facility (see Chapter 14) knows about the inspector in the sense that the backtrace frame window is an inspect window onto the frame selected from the back trace menu during a break. Thus you can call the inspector on an object that is bound on the stack by selecting its frame in the back trace menu, selecting its value with the `LEFT` button in the back trace frame window, and

selecting the inspect command with the MIDDLE button in the back trace frame window. The values of variables in frames can be set by selecting the variable name with the LEFT button and then the "Set" command with the MIDDLE button.

Note: The inspector will only allow the setting of named variables. Even with this restriction it is still possible to crash the system by setting variables inside system frames. Exercise caution in setting variables in other than your own code.

Controlling the Amount Displayed During Inspection

The amount of information displayed during inspection can be controlled using the following variables:

MAXINSPECTCDRLEVEL [Variable]

The inspector prints only the first MAXINSPECTCDRLEVEL elements of a long list, and will make the tail containing the unprinted elements the last item. The last item can be inspected to see further elements. Initially 50.

MAXINSPECTARRAYLEVEL [Variable]

The inspector prints only the first MAXINSPECTARRAYLEVEL elements of an array. The remaining elements can be inspected by calling the function (INSPECT/ARRAY ARRAY BEGINOFFSET) which inspects the BEGINOFFSET through the BEGINOFFSET + MAXINSPECTARRAYLEVEL elements of ARRAY. Initially 300.

INSPECTPRINTLEVEL [Variable]

When printing the values, the inspector resets PRINTLEVEL (see Chapter 25) to the value of INSPECTPRINTLEVEL. Initially (2 . 5).

INSPECTALLFIELDSFLG [Variable]

If INSPECTALLFIELDSFLG is T, the inspector will show computed fields (ACCESSFNFS, Chapter 8) as well as regular fields for structures that have a record definition. Initially T.

Inspect Macros

The Inspector can be extended to inspect new structures and datatypes by adding entries to the list INSPECTMACROS. An entry should be of the form (OBJECTTYPE . INSPECTINFO). OBJECTTYPE is used to determine the types of objects that are inspected with this macro. If OBJECTTYPE is a litatom, the INSPECTINFO will be used to inspect items whose type name is OBJECTTYPE. If OBJECTTYPE is a list of the form (FUNCTION DATUM-PREDICATE), DATUM-PREDICATE will be APPLIED to the item and if it returns non-NIL, the INSPECTINFO will be used to inspect the item.

INSPECTINFO can be one of two forms. If INSPECTINFO is a litatom, it should be a function that will be applied to three arguments (the item being inspected, OBJECTTYPE, and the value of WHERE passed to INSPECT) that should do the inspection. If INSPECTINFO is not a litatom, it should be a list

INTERLISP-D REFERENCE MANUAL

of (PROPERTIES FETCHFN STOREFN PROPCOMMANDFN VALUECOMMANDFN TITLECOMMANDFN TITLE SELECTIONFN WHERE PROPPRINTFN) where the elements of this list are the arguments for INSPECTW.CREATE, described below. From this list, the WHERE argument will be evaluated; the others will not. If WHERE is NIL, the value of WHERE that was passed to INSPECT will be used.

Examples:

The entry ((FUNCTION MYATOMP) PROPNAME GETPROP PUTPROP) on INSPECTMACROS would cause all objects satisfying the predicate MYATOMP to have their properties inspected with GETPROP and PUTPROP. In this example, MYATOMP should make sure the object is a litatom.

The entry (MYDATATYPE . MYINSPECTFN) on INSPECTMACROS would cause all datatypes of type MYDATATYPE to be passed to the function MYINSPECTFN.

INSPECTWs

The inspector is built on the abstraction of an INSPECTW. An INSPECTW is a window with certain window properties that display an object and respond to selections of the object's parts. It is characterized by an object and its list of properties. An INSPECTW displays the object in two columns with the property names on the left and the values of those properties on the right. An INSPECTW supports the protocol that the LEFT mouse button can be used to select any property name or property value and the MIDDLE button calls a user provided function on the selected value or property. For the Inspector application, this function puts up a menu of the alternative ways of inspecting values or of the ways of setting properties. INSPECTWs are created with the following function:

```
(INSPECTW.CREATE DATUM PROPERTIES FETCHFN STOREFN PROPCOMMANDFN  
VALUECOMMANDFN TITLECOMMANDFN TITLE SELECTIONFN WHERE PROPPRINTFN) [Function]
```

Creates an INSPECTW that views the object *DATUM*. If *PROPERTIES* is a list, it is taken as the list of properties of *DATUM* to display. If *PROPERTIES* is a litatom, it is APPLIED to *DATUM* and the result is used as the list of properties to display.

FETCHFN is a function of two arguments (OBJECT PROPERTY) that should return the value of the PROPERTY property of OBJECT. The result of this function will be printed (with PRIN2) in the INSPECTW as the value.

STOREFN is a function of three arguments (OBJECT PROPERTY NEWVALUE) that changes the PROPERTY property of OBJECT to NEWVALUE. It is used by the default *PROPCOMMANDFN* and *VALUECOMMANDFN* to change the value of a property and also by the function INSPECTW.REPLACE (described below). This can be NIL if the user provides command functions which do not call INSPECTW.REPLACE. Each replace action will be a separate event on the history list. Users are encouraged to provide UNDOable STOREFNs.

PROPCOMMANDFN is a function of three arguments (PROPERTY OBJECT INSPECTW) which gets called when the user presses the MIDDLE button and the selected item in the

USER I/O PACKAGES

INSPECTW is a property name. *PROPERTY* will be the name of the selected property, *OBJECT* will be the datum being viewed, and *INSPECTW* will be the window. If *PROPCOMMANDFN* is a string, it will get printed in the *PROMPTWINDOW* when the *MIDDLE* button is pressed. This provides a convenient way to notify the user about disabled commands on the properties. *DEFAULT.INSPECTW.PROPCOMMANDFN*, the default *PROPCOMMANDFN*, will present a menu with the single command *Set* on it. If selected, the *Set* command will read a value from the user and set the selected property to the result of *EVALuating* this read value.

VALUECOMMANDFN is a function of four arguments (*VALUE* *PROPERTY* *OBJECT* *INSPECTW*) that gets called when the user presses the *MIDDLE* button and the selected item in the *INSPECTW* is a property value. *VALUE* will be the selected value (as returned by *FETCHFN*), *PROPERTY* will be the name of the property *VALUE* is the value of, *OBJECT* will be the datum being viewed, and *INSPECTW* will be the *INSPECTW* window.

DEFAULT.INSPECTW.VALUECOMMANDFN, the default *VALUECOMMANDFN*, will present a menu of possible ways of inspecting the value and create a new *Inspect* window if one of the menu items is selected.

TITLECOMMANDFN is a function of two arguments (*INSPECTW* *OBJECT*) which gets called when the user presses the *MIDDLE* button and the cursor is in the title or border of the inspect window *INSPECTW*. This command function is provided so that users can implement commands that apply to the entire object. The default *TITLECOMMANDFN* (*DEFAULT.INSPECTW.TITLECOMMANDFN*) presents a menu with the commands *ReFetch*, *IT←datum*, and *IT←election*.

TITLE specifies the title of the window. If *TITLE* is *NIL*, the title of the window will be the printed form of *DATUM* followed by the string " Inspector". If *TITLE* is the litatom *DON'T*, the inspect window will not have a title. If *TITLE* is any other litatom, it will be applied to the *DATUM* and the potential inspect window (if it is known). If this result is the litatom *DON'T*, the inspect window will not have a title; otherwise the result will be used as a title. If *TITLE* is not a litatom, it will be used as the title.

SELECTIONFN is a function of three arguments (*PROPERTY* *VALUEFLG* *INSPECTW*) which gets called when the user releases the left button and the cursor is on one of the items. The *SELECTIONFN* allows a program to take action on the user's selection of an item in the inspect window. At the time this function is called, the selected item has been "selected". The function *INSPECTW.SELECTITEM* (described below) can be used to turn off this selection. *PROPERTY* will be the name of the property of the selected item. *VALUEFLG* will be *NIL* if the selected item is the property name; *T* if the selected item is the property value.

WHERE indicates where the inspect window should go. Its interpretation is described in *INSPECT* (see above).

PROPPRINTFN is a function of two arguments (*PROPERTY* *DATUM*) which gets called to determine what to print in the property place for the property *PROPERTY*. If *PROPPRINTFN* returns *NIL*, no property name will be printed and the value will be printed to the left of the other values.

INTERLISP-D REFERENCE MANUAL

An inspect window uses the following window property names to hold information: *DATUM*, *FETCHFN*, *STOREFN*, *PROPCOMMANDFN*, *VALUECOMMANDFN*, *SELECTIONFN*, *PROPPRINTFN*, *INSPECTWTITLE*, *PROPERTIES*, *CURRENTITEM* and *SELECTABLEITEMS*.

(**INSPECTW.REDISPLAY** *INSPECTW PROPS* -) [Function]

Updates the display of the objects being inspected in *INSPECTW*. If *PROPS* is a property name or a list of property names, only those properties are updated. If *PROPS* is *NIL*, all properties are redisplayed. This function is provided because inspect windows do not automatically update their display when the object they are showing changes.

This function is called by the *ReFetch* command in the title command menu of an *INSPECTW* (see above).

(**INSPECTW.REPLACE** *INSPECTW PROPERTY NEWVALUE*) [Function]

Calls the *STOREFN* of the inspect window *INSPECTW* to change the property named *PROPERTY* to the value *NEWVALUE* and updates the display of *PROPERTY*'s value in the display. This provides a functional interface for user *PROPCOMMANDFN*s.

(**INSPECTW.SELECTITEM** *INSPECTW PROPERTY VALUEFLG*) [Function]

Sets the selected item in an inspect window. The item is inverted on the display and put on the window property *CURRENTITEM* of *INSPECTW*. If *INSPECTW* has a *CURRENTITEM*, it is deselected. *PROPERTY* is the name of the property of the selected item. *VALUEFLG* is *NIL* if the selected item is the property name; *T* if the selected item is the property value. If *PROPERTY* is *NIL*, no item will be selected. This provides a way of deselecting all items.

PROMPTFORWORD

PROMPTFORWORD is a function that reads in a sequence of characters, generally from the keyboard, without involving *READ*-like syntax. A user can supply a prompting string, as well as a "candidate" string, which is printed and used if the user types only a word terminator character (or doesn't type anything before a given time limit). As soon as any characters are typed the "candidate" string is erased and the new input takes its place.

PROMPTFORWORD accepts user type-in until one of the "word terminator" characters is typed. Normally, the word terminator characters are *EOL*, *ESCAPE*, *LF*, *SPACE*, or *TAB*. This list can be changed using the *TERMINCHAR.LST* argument to **PROMPTFORWORD**, for example if it is desirable to allow the user to input lines including spaces.

PROMPTFORWORD also recognizes the following special characters:

Control-A

BACKSPACE

DELETE Any of these characters deletes the last character typed and appropriately erases it from the echo stream if it is a display stream.

USER I/O PACKAGES

Control-Q Erases all the type-in so far.

Control-R Reprints the accumulated string.

Control-V "Quotes" the next character: after typing Control-V, the next character typed is added to the accumulated string, regardless of any special meaning it has. Allows the user to include editing characters and word terminator characters in the accumulated string.

Control-W Erases the last word.

? Calls up a "help" facility. The action taken is defined by the `GENERATE?LIST.FN` argument to `PROMPTFORWORD` (see below). Normally, this prints a list of possible candidates.

(PROMPTFORWORD *PROMPT.STR CANDIDATE.STR GENERATE?LIST.FN ECHO.CHANNEL DONTechotypeIN.FLG URGENCY.OPTION TERMINCHARS.LST KEYBD.CHANNEL*) [Function]

`PROMPTFORWORD` has a multiplicity of features, which are specified through a rather large number of input arguments, but the default settings for them (i.e., when they aren't given, or are given as `NIL`) is such to minimize the number needed in the average case, and an attempt has been made to order the more frequently non-defaulted arguments at the beginning of the argument list. The default input and echo are both to the terminal; the terminal table in effect during input allows most control characters to be `INDICATE'd`.

`PROMPTFORWORD` returns `NIL` if a null string is typed; this would occur when no candidate is given and only a terminator is typed, or when the candidate is erased and a terminator is typed with no other input still un-erased. In all other cases, `PROMPTFORWORD` returns a string.

`PROMPTFORWORD` is controlled through the following arguments:

<i>PROMPT.STR</i>	If non- <code>NIL</code> , this is coerced to a string and used for prompting; an additional space is output after this string.
<i>CANDIDATE.STR</i>	If non- <code>NIL</code> , this is coerced to a string and offered as initial contents of the input buffer.
<i>GENERATE?LIST.FN</i>	If non- <code>NIL</code> , this is either a string to be printed out for help, or a function to be applied to <i>PROMPT.STR</i> and <i>CANDIDATE.STR</i> (after both have been coerced to strings), and which should return a list of potential candidates. The help string or list of potential candidates will then be printed on a separate line, the prompt will be restarted, and any type-in will be re-echoed. Note: If <i>GENERATE?LIST.FN</i> is a function, its value list will be cached so that it will be run at most once per call to <code>PROMPTFORWORD</code> .
<i>ECHO.CHANNEL</i>	Coerced to an output stream; <code>NIL</code> defaults to <code>T</code> , the "terminal output stream", normally (<code>TTYDISPLAYSTREAM</code>). To achieve echoing to the "current output stream", use (<code>GETSTREAM NIL 'OUTPUT</code>). If echo is to a display stream, it will have a flashing caret showing where the next input is to be echoed.
<i>DONTechotypeIN.FLG</i>	If <code>T</code> , there is no echoing of the input characters. If the value of <i>DONTechotypeIN.FLG</i> is a single-character atom or string, that character is

INTERLISP-D REFERENCE MANUAL

echoed instead of the actual input. For example, LOGIN prompts for a password with *DONTECHOTYPEIN.FLG* being "*".

URGENCY.OPTION If NIL, PROMPTFORWARD quietly wait for input, as READ does; if a number, this is the number of seconds to wait for the user to respond (if timeout is reached, then *CANDIDATE.WORD* is returned, regardless of any other type-in activity); if T, this means to wait forever, but periodically flash the window to alert the user; if TTY, then PROMPTFORWARD grabs the TTY immediately. When *URGENCY.OPTION* = TTY, the cursor is temporarily changed to a different shape to indicate the urgent nature of the request.

TERMINCHARS.LST This is list of "word terminator" character codes; it defaults to (CHARCODE (EOL ESCAPE LF SPACE TAB)). This may also be a single character code.

KEYBD.CHANNEL If non-NIL, this is coerced to a stream, and the input bytes are taken from that stream. NIL defaults to the keyboard input stream. Note that this is not the same as the terminal input stream T, which is a buffered keyboard input stream, not suitable for use with PROMPTFORWARD.

Examples:

```
(PROMPTFORWARD
  "What is your FOO word?" 'Mumble
  (FUNCTION (LAMBDA () ' (Grumble Bletch)))
  PROMPTWINDOW NIL 30)
```

This first prompts the user for input by printing the first argument as a prompt into PROMPTWINDOW; then the proffered default answer, Mumble, is printed out and the caret starts flashing just after it to indicate that the upcoming input will be echoed there. If the user fails to complete a word within 30 seconds, then the result will be the string Mumble.

```
(FRESHLINE T)
(LIST
  (PROMPTFORWARD
    (CONCAT "{" HOST " } Login:")
    (USERNAME NIL NIL T))
  (PROMPTFORWARD
    " (password)" NIL NIL NIL '*))
```

This first prompts in whatever window is currently (TTYDISPLAYSTREAM), and then takes in a username; the second call prompts with (password) and takes in another word (the password) without proffering a candidate, echoing the typed-in characters as "*".

ASKUSER

DWIM, the compiler, the editor, and many other system packages all use ASKUSER, an extremely general user interaction package, for their interactions with the user at the terminal. ASKUSER takes as its principal argument KEYLST which is used to drive the interaction. KEYLST specifies what the user can type at any given point, how ASKUSER should respond to the various inputs, what value should be returned by ASKUSER, and is also used to present the user at any given point with a list of the

USER I/O PACKAGES

possible responses. `ASKUSER` also takes other arguments which permit specifying a wait time, a default value, a message to be printed on entry, a flag indicating whether or not typeahead is to be permitted, a flag indicating whether the transaction is to be stored on the history list (see Chapter 13), a default set of options, and an (optional) input file/string.

(ASKUSER WAIT DEFAULT MESS KEYLST TYPEAHEAD LISPXPRTFLG OPTIONSLST FILE) [Function]

WAIT is either `NIL` or a number (of seconds). *DEFAULT* is a single character or a sequence (list) of characters to be used as the default inputs for the case when *WAIT* is not `NIL` and more than *WAIT* seconds elapse without any input. In this case, the character(s) from *DEFAULT* are processed exactly as though they had been typed, except that `ASKUSER` first types "...".

MESS is the initial message to be printed by `ASKUSER`, if any, and can be a string, or a list. In the latter case, each element of the list is printed, separated by spaces, and terminated with a " ? ". *KEYLST* and *OPTIONSLST* are described. *TYPEAHEAD* is `T` if the user is permitted to typeahead a response to `ASKUSER`. `NIL` means any typeahead should be cleared and saved. *LISPXPRTFLG* determines whether or not the interaction is to be recorded on the history list. *FILE* can be either `NIL` (in which case it defaults to the terminal input stream, `T`) or a stream.

All input operations take place from *FILE* until an unacceptable input is encountered, i.e., one that does not conform to the protocol defined by *KEYLST*. At that point, *FILE* is set to `T`, *DEFAULT* is set to `NIL`, the input buffer is cleared, and a bell is rung. Unacceptable inputs are not echoed.

The value of `ASKUSER` is the result of packing all the keys that were matched, unless the *RETURN* option is specified (see the Options section below).

(MAKEKEYLST LST DEFAULTKEY LCASEFLG AUTOCOMLETEFLG) [Function]

LST is a list of atoms or strings. `MAKEKEYLST` returns an `ASKUSER` *KEYLST* which will permit the user to specify one of the elements on *LST* by either typing enough characters to make the choice unambiguous, or else typing a number between 1 and *N*, where *N* is the length of *LST*.

For example, if `ASKUSER` is called with *KEYLST* = `(MAKEKEYLST ' (CONNECT SUPPORT COMPILE) )`, then the user can type `C-O-N`, `S`, `C-O-M`, `1`, `2`, or `3` to indicate one of the three choices.

If *LCASEFLG* = `T`, then echoing of upper case elements will be in lower case (but the value returned will still be one of the elements of *LST*). If *DEFAULTKEY* is non-`NIL`, it will be the last key on the *KEYLST*. Otherwise, a key which permits the user to indicate "No - none of the above" choices, in which case the value returned by `ASKUSER` will be `NIL`.

AUTOCOMLETEFLG is used as the value of the *AUTOCOMLETEFLG* option of the resulting key list.

INTERLISP-D REFERENCE MANUAL

Format of KEYLST

KEYLST is a list of elements of the form (KEY PROMPTSTRING . OPTIONS), where KEY is an atom or a string (equivalent), PROMPTSTRING is an atom or a string, and OPTIONS a list of options in property list format. The options are explained below. If an option is specified in OPTIONS, the value of the option is the next element. Otherwise, if the option is specified in the OPTIONSLST argument to ASKUSER, its value is the next element on OPTIONSLST. Thus, OPTIONSLST can be used to provide default options for an entire KEYLST, rather than having to include the option at each level. If an option does not appear on either OPTIONS or OPTIONSLST, its value is NIL.

For convenience, an entry on KEYLST of the form (KEY . ATOM/STRING), can be used as an abbreviation for (KEY ATOM/STRING CONFIRMFLG T), and an entry of just the form KEY, i.e., a non-list, as an abbreviation for (KEY NIL CONFIRMFLG T).

As each character is read, it is matched against the currently active keys. A character matches a key if it is the same character as that in the corresponding position in the key, or, if the character is an alphabetic character, if the characters are the same without regard for upper/lower case differences, i.e. "A" matches "a" and vice versa (unless the NOCASEFLG option is T, see the Options section below). In other words, if two characters have already been input and matched, the third character is matched with each active key by comparing it with the third character of that key. If the character matches with one or more of the keys, the entries on KEYLST corresponding to the remaining keys are discarded. If the character does not match with any of the keys, the character is not echoed, and a bell is rung instead.

When a key is complete, PROMPTSTRING is printed (NIL is equivalent to "", the empty string, i.e., nothing will be printed). Then, if the value of the CONFIRMFLG option is T, ASKUSER waits for confirmation of the key by a carriage return or space. Otherwise, the key does not require confirmation.

Then, if the value of the KEYLST option is not NIL, its value becomes the new KEYLST, and the process recurses. Otherwise, the key is a "leaf," i.e., it terminates a particular path through the original, top-level KEYLST, and ASKUSER returns the result of packing all the keys that have been matched and completed along the way (unless the RETURN option is used to specify some other value, as described below).

For example, when ASKUSER is called with KEYLST = NIL, the following KEYLST is used as the default:

```
((Y "escr") (N "ocr"))
```

This KEYLST specifies that if (as soon as) the user types Y (or y), ASKUSER echoes with Y, prompts with escr, and returns Y as its value. Similarly, if the user types N, ASKUSER echoes the N, prompts with ocr, and returns N. If the user types ?, ASKUSER prints:

```
Yes  
No
```

USER I/O PACKAGES

to indicate his possible responses. All other inputs are unacceptable, and ASKUSER will ring the bell and not echo or print anything.

For a more complicated example, the following is the KEYLST used for the compiler questions:

```
((ST "ore and redefine " KEYLST (" (F . "orget
exprs"))
(S . "ame as last time")
(F . "File only")
(T . "o terminal")
1
2
(Y . "es")
(N . "o"))
```

When ASKUSER is called with this KEYLST, and the user types an S, two keys are matched: ST and S. The user can then type a T, which matches only the ST key, or confirm the S key by typing a **CR** or space. If the user confirms the S key, ASKUSER prompts with "ame as last time", and returns S as its value. (Note that the confirming character is not included in the value.) If the user types a T, ASKUSER prompts with "ore and redefine", and makes (" (F . "orget exprs")) be the new KEYLST, and waits for more input. The user can then type an F, or confirm the " (which essentially starts out with all of its characters matched). If he confirms the " , ASKUSER returns ST as its value the result of packing ST and ". If he types F, ASKUSER prompts with "orget exprs", and waits for confirmation again. If the user then confirms, ASKUSER returns STF, the result of packing ST and F.

At any point the user can type a ? and be prompted with the possible responses. For example, if the user types S and then ?, ASKUSER will type:

```
STore and redefine Forget exprs
STore and redefine
Same as last time
```

Options

KEYLST	When a key is complete, if the value of the KEYLST option is not NIL, this value becomes the new KEYLST and the process recurses. Otherwise, the key terminates a path through the original, top-level KEYLST, and ASKUSER returns the indicated value.
CONFIRMFLG	If T, the key must be confirmed with either a carriage return or a space. If the value of CONFIRMFLG is a list, the confirming character may be any member of the list.
PROMPTCONFIRMFLG	If T, whenever confirmation is required, the user is prompted with the string [confirm].
NOCASEFLG	If T, says do not perform case independent matching on alphabetic characters. If NIL, do perform case independent matching, i.e. "A" matches with "a" and vice versa.

INTERLISP-D REFERENCE MANUAL

RETURN	If non-NIL, EVAL of the value of the RETURN option is returned as the value of ASKUSER. Note that different RETURN options can be specified for different keys. The variable ANSWER is bound in ASKUSER to the list of keys that have been matched. In other words, RETURN (PACK ANSWER) would be equivalent to what ASKUSER normally does.
NOECHOFLG	If non-NIL, characters that are matched (or automatically supplied as a result of typing \$ (escape) or confirming) are not echoed, nor is the confirming character, if any. The value of NOECHOFLG is automatically NIL when ASKUSER is reading from a file or string. The decision about whether or not to echo a character that matches several keys is determined by the value of the NOECHOFLG option for the first key.
EXPLAINSTRING	If the value of the EXPLAINSTRING option is non-NIL, its value is printed when the user types a ?, rather than KEY + PROMPTSTRING. EXPLAINSTRING enables more elaborate explanations in response to a ? than what the user sees when he is prompted as a result of simply completing keys. For example: One of the entries on the KEYLST used by ADDTOFILES? is: <pre>(] "Nowherecr" NOECHOFLG T EXPLAINSTRING "] - nowhere, item is marked as a dummycr")</pre> When the user types], ASKUSER just prints Nowherecr, i.e., the] is not echoed. If the user types ?, the explanation corresponding to this entry will be: <pre>] - nowhere, item is marked as a dummy</pre>
KEYSTRING	If non-NIL, characters that are matched are echoed as though the value of KEYSTRING were used in place of the key. KEYSTRING is also used for computing the value returned. The main reason for this feature is to enable echoing in lowercase.
PROMPTON	If non-NIL, PROMPTSTRING is printed only when the key is confirmed with a member of the value of PROMPTON.
COMPLETEON	When a confirming character is typed, the N characters that are automatically supplied, as specified in case (4), are echoed only when the key is confirmed with a member of the value of PROMPTON.

The PROMPTON and COMPLETEON options enable the user to construct a KEYLST which will cause ASKUSER to emulate the action of the TENEX exec. The protocol followed by the TENEX exec is that the user can type as many characters as he likes in specifying a command. The command can be completed with a carriage return or space, in which case no further output is forthcoming, or with a \$ (escape), in which case the rest of the characters in the command are echoed, followed by some prompting information. The following KEYLST would handle the TENEX COPY and CONNECT comands:

```
((COPY " (FILE LIST) "
  PROMPTON ($)
  COMPLETEON ($)
  CONFIRMFLG ($))

(CONNECT " (TO DIRECTORY) "
```

USER I/O PACKAGES

	PROMPTON (\$)
	COMPLETEON (\$)
	CONFIRMFLG (\$))
AUTOCOMLETEFLG	If the value of the AUTOCOMLETEFLG option is not NIL, ASKUSER will automatically supply unambiguous characters whenever it can, i.e., ASKUSER acts as though \$ (escape) were typed after each character (except that it does not ring the bell if there are no unambiguous characters).
MACROCHARS	value is a list of dotted pairs of form (CHARACTER . FORM). When CHARACTER is typed, and it does not match any of the current keys, FORM is evaluated and nothing else happens, i.e. the matching process stays where it is. For example, ? could have been implemented using this option. Essentially MACROCHARS provides a read macro facility while inside of ASKUSER (since ASKUSER does READC's, read macros defined via the readtable are never invoked).
EXPLAINDELIMITER	value is what is printed to delimit explanation in response to ?. Initially a carriage return, but can be reset, e.g. to a comma, for more linear output.

Operation

All input operations are executed with the terminal table in the variable ASKUSERTTBL, in which the following is true:

- (CONTROL T) has been executed (see the Line-Buffering section of Chapter 30), so that ASKUSER can interact with the user after each character is typed
- (ECHOMODE NIL) has been executed (see the Terminal Control Functions section of Chapter 30), so that ASKUSER can decide after it reads a character whether or not the character should be echoed, and with what, e.g. unacceptable inputs are never echoed.

As each character is typed, it is matched against KEYLST, and appropriate echoing and/or prompting is performed. If the user types an unacceptable character, ASKUSER simply rings the bell and allows him to try again.

At any point, the user can type ? and receive a list of acceptable responses at that point (generated from KEYLST), or type a Control-A, Control-Q, Control-X, or delete, which causes ASKUSER to reinitialize, and start over.

Note that ?, Control-A, Control-Q, and Control-X will not work if they are acceptable inputs, i.e., they match one of the keys on KEYLST. Delete will not work if it is an interrupt character, in which case it is not seen by ASKUSER.

When an acceptable sequence is completed, ASKUSER returns the indicated value.

INTERLISP-D REFERENCE MANUAL

Completing a Key

The decision about when a key is complete is more complicated than simply whether or not all of its characters have been matched. In the compiler questions example above, all of the characters in the *S* key are matched as soon as the *S* has been typed, but until the next character is typed, *ASKUSER* does not know whether the *S* completes the *S* key, or is simply the first character in the *ST* key. Therefore, a key is considered to be complete when:

1. All of its characters have been matched and it is the only key left, i.e., there are no other keys for which this key is a substring.
2. All of its characters have been matched and a confirming character is typed.
3. All of its characters have been matched, and the value of the *CONFIRMFLG* option is *NIL*, and the value of the *KEYLST* option is not *NIL*, and the next character matches one of the keys on the value of the *KEYLST* option.
4. There is only one key left and a confirming character is typed. Note that if the value of *CONFIRMFLG* is *T*, the key still has to be confirmed, regardless of whether or not it is complete. For example, if the first entry in the above example were instead

```
(ST "ore and redefine " CONFIRMFLG T KEYLST (" (F . "orget  
exprs"))
```

and the user wanted to specify the *STF* path, he would have to type *ST*, then confirm before typing *F*, even though the *ST* completed the *ST* key by the rule in Case 1. However, he would be prompted with *ore and redefine* as soon as he typed the *T*, and completed the *ST* key.

Case 2 says that confirmation can be used to complete a key in the case where it is a substring of another key, even where the value of *CONFIRMFLG* is *NIL*. In this case, the confirming character doubles as both an indicator that the key is complete, and also to confirm it, if necessary. This situation corresponds to typing *Scr* in the above example.

Case 3 says that if there were another entry whose key was *STX* in the above example, so that after the user typed *ST*, two keys, *ST* and *STX*, were still active, then typing *F* would complete the *ST* key, because *F* matches the *(F . "orget exprs")* entry on the value of the *KEYLST* option of the *ST* entry. In this case, *ore and redefine* would be printed before the *F* was echoed.

Finally, Case 4 says that the user can use confirmation to specify completion when only one key is left, even when all of its characters have not been matched. For example, if the first key in the above example were *STORE*, the user could type *ST* and then confirm, and *ORE* would be echoed, followed by whatever prompting was specified. In this case, the confirming character also confirms the key if necessary, so that no further action is required, even when the value of *CONFIRMFLG* is *T*.

Case 4 permits the user not to have to type every character in a key when the key is the only one left. Even when there are several active keys, the user can type <escape> to specify the next *N*>0 common characters among the currently active keys. The effect is exactly the same as though these characters

USER I/O PACKAGES

had been typed. If there are no common characters in the active keys at that point, i.e. $N = 0$, the \$ is treated as an incorrect input, and the bell is rung. For example, if KEYLST is (CLISPFLG CLISPIFYPACKFLG CLISPIFTRANFLG), and the user types C followed by \$, ASKUSER will supply the L, I, S, and P. The user can then type F followed by a carriage return or space to complete and confirm CLISPFLG, as per Case 4, or type I, followed by \$, and ASKUSER will supply the F, etc. Note that the characters supplied do not have to correspond to a terminal segment of any of the keys. Note also that the \$ does not confirm the key, although it may complete it in the case that there is only one key active.

If the user types a confirming character when several keys are left, the next $N > 0$ common characters are still supplied, the same as with \$. However, ASKUSER assumes the intent was to complete a key, i.e., Case 4 is being invoked. Therefore, after supplying the next N characters, the bell is rung to indicate that the operation was not completed. In other words, typing a confirming character has the same effect as typing an \$ in that the next N common characters are supplied. Then, if there is only one key left, the key is complete (Case 4) and confirmation is not required. If the key is not the only key left, the bell is rung.

Special Keys

& This can be used as a key to match with any single character, provided the character does not match with some other key at that level. For the purposes of echoing and returning a value, the effect is the same as though the character that were matched actually appeared as the key.

<escape> This can be used as a key to match with the result of a single call to READ. For example, if the KEYLST were:

```
((COPY " (FILE LIST) "
  PROMPTON ($)
  COMPLETEON ($)
  CONFIRMFLG ($)
  KEYLST (($ NIL RETURN ANSWER))))
```

then if the user typed COP FOOcr, (COPY FOO) would be returned as the value of ASKUSER. One advantage of using \$, rather than having the calling program perform the READ, is that the call to READ from inside ASKUSER is ERRORSET protected, so that the user can back out of this path and reinitialize ASKUSER, e.g. to change from a COPY command to a CONNECT command, simply by typing Control-E.

Escape Escape This can be used as a key to match with the result of a single call to READLINE.

A list A list can be used as a key, in which case the list/form is evaluated and its value "matches" the key. This feature is provided primarily as an escape hatch for including arbitrary input operations as part of an ASKUSER sequence. For example, the effect of \$\$ (escape, escape) could be achieved simply by using (READLINE T) as a key.

INTERLISP-D REFERENCE MANUAL

" " The empty string can be used as a key. Since it has no characters, all of its characters are automatically matched. " " essentially functions as a place marker. For example, one of the entries on the KEYLST used by ADDTOFILES? is:

```
( " "File/list: "
  EXPLAINSTRING "a file name or name of a
  function list"
```

KEYLST (\$))

Thus, if the user types a character that does not match any of the other keys on the KEYLST, then the character completes the " " key, by virtue of case (4), since the character will match with the \$ in the inner KEYLST. ASKUSER then prints File/list: before echoing the character, then calls READ. The character will be read as part of the READ. The value returned by ASKUSER will be the value of the READ.

Note: For Escape, Escape Escape, or a list, if the last character read by the input operation is a separator, the character is treated as a confirming character for the key. However, if the last character is a break character, it will be matched against the next key.

Startup Protocol and Typeahead

Interlisp permits and encourages the user to typeahead; in actual practice, the user frequently does this. This presents a problem for ASKUSER. When ASKUSER is entered and there has been typeahead, was the input intended for ASKUSER, or was the interaction unanticipated, and the user simply typing ahead to some other program, e.g. the programmer's assistant? Even where there was no typeahead, i.e., the user starts typing after the call to ASKUSER, the question remains of whether the user had time to see the message from ASKUSER and react to it, or simply began typing ahead at an inauspicious moment. Thus, what is needed is an interlock mechanism which warns the user to stop typing, gives him a chance to respond to the warning, and then allows him to begin typing to ASKUSER.

Therefore, when ASKUSER is first entered, and the interaction is to take place with a terminal, and typeahead to ASKUSER is not permitted, the following protocol is observed:

USER I/O PACKAGES

1. If there is typeahead, ASKUSER clears and saves the input buffers and rings the bell to warn the user to stop typing. The buffers will be restored when ASKUSER completes operation and returns.
2. If MESS, the message to be printed on entry, is not NIL (the typical case), ASKUSER then prints MESS if it is a string, otherwise CAR of MESS, if MESS is a list.
3. After printing MESS or CAR of MESS, ASKUSER waits until the output has actually been printed on the terminal to make sure that the user has actually had a chance to see the output. This also give the user a chance to react. ASKUSER then checks to see if anything additional has been typed in the intervening period since it first warned the user in (1). If something has been typed, ASKUSER clears it out and again rings the bell. This latter material, i.e., that typed between the entry to ASKUSER and this point, is discarded and will not be restored since it is not certain whether the user simply reacted quickly to the first warning (bell) and this input is intended for ASKUSER, or whether the user was in the process of typing ahead when the call to ASKUSER occurred, and did not stop typing at the first warning, and therefore this input is a continuation of input intended for another program.

Anything typed after (3) is considered to be intended for ASKUSER, i.e., once the user sees MESS or CAR of MESS, he is free to respond. For example, UNDO (see Chapter 13) calls ASKUSER when the number of undosaves are exceeded for an event with MESS = (LIST NUMBER-UNDOSAVES "undosaves, continue saving"). Thus, the user can type a response as soon as NUMBER-UNDOSAVES is typed.

4. ASKUSER then types the rest of MESS, if any.
5. Then ASKUSER goes into a wait loop until something is typed. If WAIT, the wait time, is not NIL, and nothing is typed in WAIT seconds, ASKUSER will type ". . ." and treat the elements of DEFAULT, the default value, as a list of characters, and begin processing them exactly as though they had been typed. If the user does type anything within WAIT seconds, he can then wait as long as he likes, i.e., once something has been typed, ASKUSER will not use the default value specified in DEFAULT.

If the user wants to consider his response for more than WAIT seconds, and does not want ASKUSER to default, he can type a carriage return or a space, which are ignored if they are not specified as acceptable inputs by KEYLIST (see below) and they are the first thing typed.

If the calling program knows that the user is expecting an interaction with ASKUSER, e.g., another interaction preceded this one, it can specify in the call to ASKUSER that typeahead is permitted. In this case, ASKUSER simply notes whether there is any typeahead, then prints MESS and goes into a wait loop as described above.

If there is typeahead that contains unacceptable input, ASKUSER will assume that the typeahead was not intended for ASKUSER, and will restore the typeahead when it completes operation and returns.

6. Finally, if the interaction is not with the terminal, i.e., the optional input file/string is specified, ASKUSER simply prints MESS and begins reading from the file/string.

TTYIN Display Typein Editor

TTYIN is an Interlisp function for reading input from the terminal. It features altmode completion, spelling correction, help facility, and fancy editing, and can also serve as a glorified free text input function. This document is divided into two major sections: how to use TTYIN from the user's point of view, and from the programmer's.

TTYIN exists in implementations for Interlisp-10 and Interlisp-D. The two are substantially compatible, but the capabilities of the two systems differ (Interlisp-D has a more powerful display and allows greater access to the system primitives needed to control it effectively; it also has a mouse, greatly reducing the need for keyboard-oriented editing commands). Descriptions of both are included in this document for completeness, but Interlisp-D users may find large sections irrelevant.

Entering Input With TTYIN

There are two major ways of using TTYIN: set LISPXREADFN to TTYIN, so the LISPX executive uses it to obtain input; and call TTYIN from within a program to gather text input. Mostly the same rules apply to both; places where it makes a difference are mentioned below.

The following characters may be used to edit your input, independent of what kind of terminal you are on. The more TTYIN knows about your terminal, of course, the nicer some of these will behave. Some functions are performed by one of several characters; any character that you happen to have assigned as an interrupt character will, of course, not be read by TTYIN. There is a (somewhat inelegant) way of changing which characters perform which functions, described under TTYINREADMACROS later on.

Control-A BACKSPACE

DELETE Deletes a character. At the start of the second or subsequent lines of your input, deletes the last character of the previous line.

Control-W Deletes a "word". Generally this means back to the last space or parenthesis.

Control-Q Deletes the current line, or if the current line is blank, deletes the previous line.

Control-R Refreshes the current line. Two in a row refreshes the whole buffer (when doing multi-line input).

ESCAPE Tries to complete the current word from the spelling list provided to TTYIN, if any. In the case of ambiguity, completes as far as is uniquely determined, or rings the bell. For LISPX input, the spelling list may be USERWORDS (see discussion of TTYINCOMPLETEFLG).

Interlisp-10 only: If no spelling list was provided, but the word begins with a "<", tries directory name completion (or filename completion if there is already a matching ">" in the current word).

USER I/O PACKAGES

- ? If typed in the middle of a word will supply alternative completions from the `SPLST` argument to `TTYIN` (if any). `?ACTIVATEFLG` (see the Assorted Flags section below) must be true to enable this feature.
- Control-Y** Escapes to a Lisp user exec, from which you may return by the command `OK`. However, when in `READ` mode and the buffer is non-empty, `Control-Y` is treated as Lisp's `unquote` macro instead, so you have to use meta-`Control-Y` (below) to invoke the user exec.
- LF in Interlisp-10** Retrieves characters from the previous non-empty buffer when it is able to; e.g., when typed at the beginning of the line this command restores the previous line you typed at `TTYIN`; when typed in the middle of a line fills in the remaining text from the old line; when typed following `↑Q` or `↑W` restores what those commands erased.
- ;** If typed as the first character of the line means the line is a comment; it is ignored, and `TTYIN` loops back for more input.

Note: The exact behaviour of this character is determined by the value of `TTYINCOMMENTCHAR` (see the Assorted Flags section below).
- Control-X** Goes to the end of your input (or end of expression if there is an excess right parenthesis) and returns if parentheses are balanced, beeps if not.

During most kinds of input, `TTYIN` is in "autofill" mode: if a space is typed near the right margin, a carriage return is simulated to start a new line. In fact, on cursor-addressable displays, lines are always broken, if possible, so that no word straddles the end of the line. The "pseudo-carriage return" ending the line is still read as a space, however; i.e., the program keeps track of whether a line ends in a carriage return or is merely broken at some convenient point. You won't get carriage returns in your strings unless you explicitly type them.

Mouse Commands

The mouse buttons are interpreted as follows during `TTYIN` input:

- LEFT** Moves the caret to where the cursor is pointing. As you hold down **LEFT**, the caret moves around with the cursor; after you let up, any typein will be inserted at the new position.
- MIDDLE** Like **LEFT**, but moves only to word boundaries.
- RIGHT** Deletes text from the caret to the cursor, either forward or backward. While you hold down **RIGHT**, the text to be deleted is complemented; when you let up, the text actually goes away. If you let up outside the scope of the text, nothing is killed (this is how to "cancel" the command). This is roughly the same as `CTRL-RIGHT` with no initial selection (below).

If you hold down `CTRL` and/or `SHIFT` while pressing the mouse buttons, you instead get secondary selection, move selection or delete selection. You make a selection by bugging **LEFT** (to select a character) or **MIDDLE** (to select a word), and optionally extend the selection either left or right using **RIGHT**. While you are doing this, the caret does not move, but your selected text is highlighted in a manner indicating what is about to happen. When you have made your selection (all mouse buttons up now), lift up on `CTRL` and/or `SHIFT` and the action you have selected will occur, which is:

INTERLISP-D REFERENCE MANUAL

SHIFT	The selected text as typein at the caret. The text is highlighted with a broken underline during selection.
CTRL	Delete the selected text. The text is complemented during selection.
CTRL-SHIFT	Combines the above: delete the selected text and insert it at the caret. This is how you move text about.

You can cancel a selection in progress by pressing LEFT or MIDDLE as if to select, and moving outside the range of the text.

The most recent text deleted by mouse command can be inserted at the caret by typing Middle-blank key (on the Xerox 1132) or the Open key (on the Xerox 1108). This is the same key that retrieves the previous buffer when issued at the end of a line.

Display Editing Commands

On terminals with a meta key: In Interlisp-10, TTYIN reads from the terminal in binary mode, allowing many more editing commands via the meta key, in the style of TVEDIT commands. Note that due to Tenex's unfortunate way of handling typeahead, it is not possible to type ahead edit commands before TTYIN has started (i.e., before its prompt appears), because the meta bit will be thrown away. Also, since Escape has numerous other meanings in Lisp and even in TTYIN (for completion), this is not used as a substitute for the meta key.

In Interlisp-D: Users will probably have little use for most of these commands, as cursor positioning can often be done more conveniently, and certainly more obviously, with the mouse. Nevertheless, some commands, such as the case changing commands, can be useful. The <bottom-blank> key can be used as a meta key if you perform (METASHIFT T) (see Chapter 30). Alternatively, you can use the variable EDITPREFIXCHAR as described in the next paragraph.

On display terminals without a meta key: If you want to type any of these commands, you need to prefix them with the "edit prefix" character. Set the variable EDITPREFIXCHAR to the character code of the desired prefix char. Type the edit prefix twice to give an "meta-escape" command. Some users of the TENEX TVEDIT program like to make escape (33Q) be the edit prefix, but this makes it somewhat awkward to ever use escape completion. EDITPREFIXCHAR is initially NIL.

On hardcopy terminals without a meta key: You probably want to ignore this section, since you won't be able to see what's going on when you issue edit commands; there is no attempt made to echo anything reasonable.

In the descriptions below, "current word" means the word the cursor is under, or if under a space, the previous word. Currently parentheses are treated as spaces, which is usually what you want, but can occasionally cause confusion in the word deletion commands. Most commands can be preceded by numbers or escape (means infinity), only the first of which requires the meta key (or the edit prefix). Some commands also accept negative arguments, but some only look at the magnitude of the arg.

USER I/O PACKAGES

Most of these commands are taken from the display editors TVEDIT and/or E, and are confined to work within one line of text unless otherwise noted.

Cursor Movement Commands:

■ **Meta-DELETE**

Meta-BS

Meta-< Back up one (or n) characters.

Meta-SPACE

Meta-> Moves forward one (or n) characters.

Meta-^ Moves up one (or n) lines.

Meta-lf Moves down one (or n) lines.

Meta-(Moves back one (or n) words.

Meta-) Moves ahead one (or n) words.

Meta-TAB Moves to end of line; with an argument moves to nth end of line; **Meta-ESC-TAB** goes to end of buffer.

Control-Meta-L Moves to start of line (or nth previous, or start of buffer).

Meta-{
Meta-} Go to start and end of buffer, respectively.

Meta-[Moves to beginning of the current list, where cursor is currently under an element of that list or its closing paren. (See also the auto-parenthesis-matching feature below under "Flags".)

Meta-] Moves to end of current list.

Meta-Sx Skips ahead to next (or nth) occurrence of character x, or rings the bell.

Meta-Bx Backward search.

Buffer Modification Commands:

Meta-Zx Zaps characters from cursor to next (or nth) occurrence of x. There is no unzip command yet.

Meta-A

Meta-R Repeat the last S, B or Z command, regardless of any intervening input (note this differs from TEdit's A command).

Meta-K Kills the character under the cursor, or n chars starting at the cursor.

Meta-CR When the buffer is empty is the same as **LF**, i.e. restores buffer's previous contents. Otherwise is just like a **CR** (except that it also terminates an insert). Thus, **Meta-CR CR** will repeat the previous input (as will **LF CR** without the meta key).

INTERLISP-D REFERENCE MANUAL

- Meta-O** Does "Open line", inserting a crlf after the cursor, i.e., it breaks the line but leaves the cursor where it is.
- Meta-T** Transposes the characters before and after the cursor. When typed at the end of a line, transposes the previous two characters. Refuses to handle funny cases, such as tabs.
- Meta-G** Grabs the contents of the previous line from the cursor position onward. **Meta-nG** grabs the nth previous line.
- Meta-L** Lowercases current word, or n words on line. **Meta-ESC-L** lowercases the rest of the line, or if given at the end of line lowercases the entire line.
- Meta-U** Uppercases analogously.
- Meta-C** Capitalize. If you give it an argument, only the first word is capitalized; the rest are just lowercased.
- Control-Meta-Q** Deletes the current line. **Control-Meta-ESC-Q** deletes from the current cursor position to the end of the buffer. No other arguments are handled.
- Control-Meta-W** Deletes the current word, or the previous word if sitting on a space.
- Meta-J** "Justify" this line. This will break it if it is too long, or move words up from the next line if too short. Will not join to an empty line, or one starting with a tab (both of which are interpreted as paragraph breaks). Any new line breaks it introduces are considered spaces, not carriage returns. **Meta-nJ** justifies n lines.
- The linelength is defined as TTYJUSTLENGTH, ignoring any prompt characters at the margin. If TTYJUSTLENGTH is negative, it is interpreted as relative to the right margin. TTYJUSTLENGTH is initially -8 in Interlisp-D, 72 in Interlisp-10.
- Meta-ESC-F** "Finishes" the input, regardless of where the cursor is. Specifically, it goes to the end of the input and enters a CR, **control-Z** or "", depending on whether normal, REPEAT or READ input is happening. Note that a "" won't necessarily end a READ, but it seems likely to in most cases where you would be inclined to use this command, and makes for more predictable behavior.

Miscellaneous Commands:

- Meta-P** Interlisp-D: Prettyprint buffer. Clears the buffer and reprints it using prettyprint. If there are not enough right parentheses, it will supply more; if there are too many, any excess remains unprettyprinted at the end of the buffer. May refuse to do anything if there is an unclosed string or other error trying to read the buffer.
- Meta-N** Refresh line. Same as **Control-R**. **Meta-ESC-N** refreshes the whole buffer; **Meta-nN** refreshes n lines. Cursor movement in TTYIN depends on TTYIN being the only source of output to the screen; if you do a **Control-T**, or a system message appears, or line noise occurs, you may need to refresh the line for best results. In Interlisp-10, if for some reason your terminal falls out of binary mode (e.g. can happen when returning to a Lisp running in a lower fork), **Meta-<anything>** is unreadable, so you'd have to type **Control-R** instead.
- Control-Meta-Y** Gets user exec. Thus, this is like regular **Control-Y**, except when doing a READ (when control-Y is a read macro and hence does not invoke this function).

USER I/O PACKAGES

Control-Meta-ESC-Y Gets a user exec, but first unreads the contents of the buffer from the cursor onward. Thus if you typed at TTYIN something destined for the Lisp executive, you can do **Control-Meta-L-ESC-Control-Y** and give it to Lisp.

Meta-← Adds the current word to the spelling list USERWORDS. With zero arg, removes word. See TTYINCOMPLETEFLG (see the Assorted Flags section below).

Note to Datamedia, Heath users: In addition to simple cursor movement commands and insert/delete, TTYIN uses the display's cursor-addressing capability to optimize cursor movements longer than a few characters, e.g. **Meta-TAB** to go to the end of the line. In order to be able to address the cursor, TTYIN has to know where it is to begin with. Lisp keeps track of the current print position within the line, but does not keep track of the line on the screen (in fact, it knows precious little about displays, much like Tenex). Thus, TTYIN establishes where it is by forcing the cursor to appear on the last line of the screen. Ordinarily this is the case anyway (except possibly on startup), but if the cursor happens to be only halfway down the screen at the time, there is a possibly unsettling leap of the cursor when TTYIN starts.

Using TTYIN for Lisp Input

When TTYIN is loaded, or a sysout containing TTYIN is started up, the function SETREADFN is called. If the terminal is a display, it sets LISPXREADFN (see Chapter 13) to be TTYINREAD. If the terminal is not a display terminal, SETREADFN will set the variable to READ. (SETREADFN 'READ) will also set it to READ.

There are two principal differences between TTYINREAD and READ: (1) parenthesis balancing. The input does not activate on an exactly balancing right paren/bracket unless the input started with a paren/bracket, e.g., USE (FOO) FOR (FIE) will all be on one line, terminated by **CR**; and (2) read macros.

In Interlisp-10, TTYIN does not use a read table (TTYIN behaves as though using the default initial Lisp terminal input readtable), so read macros and redefinition of syntax characters are not supported; however, " ' " (QUOTE) and "Control-Y" (EVAL) are built in, and a simple implementation of ? and ?= is supplied. Also, the TTYINREADMACROS facility described below can supply some of the functionality of immediate read macros in the editor.

In Interlisp-D, read macros are (mostly) supported. Immediate read macros take effect only if typed at the end of the input (it's not clear what their semantics should be elsewhere).

Useful Macros

There are two useful edit macros that allow you to use TTYIN as a character editor: (1) ED loads the current expression into the ttyin buffer to be edited (this is good for editing comments and strings). Input is terminated in the usual way (by typing a balancing right programmer's assistant command FIX will load the buffer with the event's input, rather than calling the editor. If you really wanted the

INTERLISP-D REFERENCE MANUAL

Interlisp editor for your fix, you can say `FIX EVENT - TTY`: once you got TTYIN's version to force you into the editor.

Programming With TTYIN

(**TTYIN** PROMPT SPLST HELP OPTIONS ECHOTOFILE TABS UNREADBUF RDTBL) [Function]

TTYIN prints PROMPT, then waits for input. The value returned in the normal case is a list of all atoms on the line, with comma and parens returned as individual atoms; OPTIONS may be used to get a different kind of value back.

PROMPT is an atom or string (anything else is converted to a string). If NIL, the value of DEFAULTPROMPT, initially "** ", will be used. If PROMPT is T, no prompt will be given. PROMPT may also be a dotted pair (PROMPT1 . PROMPT2), giving the prompt for the first and subsequent (or overflow) lines, each prompt being a string/atom or NIL to denote absence of prompt. The default prompt for overflow lines is "...". Note that rebinding DEFAULTPROMPT gives a convenient way to affect all the "ordinary" prompts in some program module.

SPLST is a spelling list, i.e., a list of atoms or dotted pairs (SYNONYM . ROOT). If supplied, it is used to check and correct user responses, and to provide completion if the user types escape. If SPLST is one of the Lisp system spelling lists (e.g., USERWORDS or SPELLINGS3), words that are escape-completed get moved to the front, just as if a FIXSPELL had found them. Autocompletion is also performed when user types a break character (cr, space, paren, etc), unless one of the "nofixspell" options below is selected; i.e., if the word just typed would uniquely complete by escape, TTYIN behaves as though escape had been typed.

HELP, if non-NIL, determines what happens when the user types ? or HELP. If HELP = T, program prints back SPLST in suitable form. If HELP is any other litatom, or a string containing no spaces, it performs (DISPLAYHELP HELP). Anything else is printed as is. If HELP is NIL, ? and HELP are treated as any other atoms the user types. [DISPLAYHELP is a user-supplied function, initially a noop; systems with a suitable HASH package, for example, have defined it to display a piece of text from a hashfile associated with the key HELP.]

OPTIONS is an atom or list of atoms chosen from among the following:

NOFIXSPELL	Uses SPLST for HELP and Escape completion, but does not attempt any FIXSPELLing. Mainly useful if SPLST is incomplete and the caller wants to handle corrections in a more flexible way than a straight FIXSPELL.
MUSTAPPROVE	Does spelling correction, but requires confirmation.
CRCOMPLETE	Requires confirmation on spelling correction, but also does autocompletion on <cr> (i.e. if what user has typed so far uniquely identifies a member of SPLST, completes it). This allows you to have the benefits of autocompletion and still allow new words to be typed.

USER I/O PACKAGES

DIRECTORY	(only if SPLST = NIL) Interprets Escape to mean directory name completion [Interlisp-10 only].
USER	Like DIRECTORY, but does username completion. This is identical to DIRECTORY under Tenex [Interlisp-10 only].
FILE	(only if SPLST = NIL) Interprets Escape to mean filename completion [Sumex and Tops20 only].
FIX	If response is not on, or does not correct to, SPLST, interacts with user until an acceptable response is entered. A blank line (returning NIL) is always accepted. Note that if you are willing to accept responses that are not on SPLST, you probably should specify one of the options NOXFISPELL, MUSTAPPROVE or CRCOMPLETE, lest the user's new response get FIXSPelled away without their approval.
STRING	Line is read as a string, rather than list of atoms. Good for free text.
NORAISE	Does not convert lower case letters to upper case.
NOVALUE	For use principally with the ECHOTOFILE arg (below). Does not compute a value, but returns T if user typed anything, NIL if just a blank line.
REPEAT	For multi-line input. Repeatedly prompts until user types Control-Z (as in Tenex sndmsg). Returns one long list; with STRING option returns a single string of everything typed, with carriage returns (EOL) included in the string.
TEXT	Implies REPEAT, NORAISE, and NOVALUE. Additionally, input may be terminated with Control-V, in which case the global flag CTRLVFLG will be set true (it is set to NIL on any other termination). This flag may be utilized in any way the caller desires.
COMMAND	Only the first word on the line is treated as belonging to SPLST, the remainder of the line being arbitrary text; i.e., "command format". If other options are supplied, COMMAND still applies to the first word typed. Basically, it always returns (CMD . REST-OF-INPUT), where REST-OF-INPUT is whatever the other options dictate for the remainder. E.g. COMMAND NOVALUE returns (CMD) or (CMD . T), depending on whether there was further input; COMMAND STRING returns (CMD . "REST-OF-INPUT"). When used with REPEAT, COMMAND is only in effect for the first line typed; furthermore, if the first line consists solely of a command, the REPEAT is ignored, i.e., the entire input is taken to be just the command.
READ	Parens, brackets, and quotes are treated a la READ, rather than being returned as individual atoms. Control characters may be input via the Control-Vx notation. Input is terminated roughly along the lines of READ conventions: a balancing or over-balancing right paren/bracket will activate the input, or <cr> when no parenthesis remains unbalanced. READ overrides all other options (except NORAISE).

INTERLISP-D REFERENCE MANUAL

- LISPXREAD** Like **READ**, but implies that **TTYIN** should behave even more like **READ**, i.e., do **NORAISE**, not be errorset-protected, etc.
- NOPROMPT** Interlisp-D only: The prompt argument is treated as usual, except that **TTYIN** assumes that the prompt for the first line has already been printed by the caller; the prompt for the first line is thus used only when redisplaying the line.

ECHOTOFILE if specified, user's input is copied to this file, i.e., **TTYIN** can be used as a simple text-to-file routine if **NOVALUE** is used. If **ECHOTOFILE** is a list, copies to all files in the list. **PROMPT** is not included on the file.

TABS is a special addition for tabular input. It is a list of tabstops (numbers). When user types a tab, **TTYIN** automatically spaces over to the next tabstop (thus the first tabstop is actually the second "column" of input). Also treats specially the characters ***** and **"**; they echo normally, and then automatically tab over.

UNREADBUF allows the caller to "preload" the **TTYIN** buffer with a line of input. **UNREADBUF** is a list, the elements of which are unread into the buffer (i.e., "the outer parentheses are stripped off") to be edited further as desired; a simple carriage return (or Control-Z for **REPEAT** input) will thus cause the buffer's contents to be returned unchanged. If doing **READ** input, the "PRIN2 names" of the input list are used, i.e., quotes and %'s will appear as needed; otherwise the buffer will look as though **UNREADBUF** had been **PRIN1**'ed. **UNREADBUF** is treated somewhat like **READBUF**, so that if it contains a pseudo-carriage return (the value of **HISTSTRO**), the input line terminates there.

Input can also be unread from a file, using the **HISTSTR1** format: **UNREADBUF** = (<value of **HISTSTR1**> (**FILE** **START** . **END**)), where **START** and **END** are file byte pointers. This makes **TTYIN** a miniature text file editor.

RDTBL [Interlisp-D only] is the read table to use for **READING** the input when one of the **READ** options is given. A lot of character interpretations are hardwired into **TTYIN**, so currently the only effect this has is in the actual **READ**, and in deciding whether a character typed at the end of the input is an immediate read macro, for purposes of termination.

If the global variable **TYPEAHEADFLG** is **T**, or option **LISPXREAD** is given, **TTYIN** permits type-ahead; otherwise it clears the buffer before prompting the user.

Using **TTYIN** as a General Editor

The following may be useful as a way of outsiders to call **TTYIN** as an editor. These functions are currently only in Interlisp-D.

USER I/O PACKAGES

(**TTYINEDIT** *EXPRS WINDOW PRINTFN PROMPT*) [Function]

This is the body of the edit macro EE. Switches the tty to *WINDOW*, clears it, prettyprints *EXPRS*, a list of expressions, into it, and leaves you in TTYIN to edit it as Lisp input. Returns a new list of expressions.

If *PRINTFN* is non-NIL, it is a function of two arguments, *EXPRS* and *FILE*, which is called instead of PRETTYPRINT to print the expressions to the window (actually to a scratch file). Note that *EXPRS* is a list, so normally the outer parentheses should not be printed. *PRINTFN* = T is shorthand for "unpretty"; use PRIN2 instead of PRETTYPRINT.

PROMPT determines what prompt is printed, if any. If T, no prompt is printed. If NIL, it defaults to the value of TTYINEDITPROMPT.

TTYINAUTOCLOSEFLG [Variable]

If TTYINAUTOCLOSEFLG is true, TTYINEDIT closes the window on exit.

TTYINEDITWINDOW [Variable]

If the WINDOW arg to TTYINEDIT is NIL, it uses the value of TTYINEDITWINDOW, creating it if it does not yet exist.

TTYINPRINTFN [Variable]

The default value for PRINTFN in EE's call to TTYINEDIT.

(**SET.TTYINEDIT.WINDOW** *WINDOW*) [Function]

Called under a RESETLST. Switches the tty to *WINDOW* (defaulted as in TTYINEDIT) and clears it. The window's position is left so that TTYIN will be happy with it if you now call TTYIN yourself. Specifically, this means positioning an integral number of lines from the bottom of the window, the way the top-level tty window normally is.

(**TTYIN.SCRATCHFILE**) [Function]

Returns, possibly creating, the scratchfile that TTYIN uses for prettyprinting its input. The file pointer is set to zero. Since TTYIN does use this file, beware of multiple simultaneous use of the file.

?= Handler

In Interlisp, the ?= read macro displays the arguments to the function currently "in progress" in the typein. Since TTYIN wants you to be able to continue editing the buffer after a ?=, it processes this macro specially on its own, printing the arguments below your typein and then putting the cursor back where it was when ?= was typed. For users who want special treatment of ?=, the following hook exists:

INTERLISP-D REFERENCE MANUAL

TTYIN?=FN [Variable]

The value of this variable, if non-NIL, is a user function of one argument that is called when ?= is typed. The argument is the function that ?= thinks it is inside of. The user function should return one of the following:

NIL Normal ?= processing is performed.

T Nothing is done. Presumably the user function has done something privately, perhaps diddled some other window, or called TTYIN.PRINTARGS (below).

a list (ARGS . STUFF) Treats STUFF as the argument list of the function in question, and performs the normal ?= processing using it.

anything else The value is printed in lieu of what ?= normally prints.

At the time that ?= is typed, nothing has been "read" yet, so you don't have the normal context you might expect inside a conventional readmacro. If the user function wants to examine the typed-in arguments being passed to the fn, however, it can call the function TTYIN.READ?=ARGS:

(TTYIN.READ?=ARGS) [Function]

When called inside TTYIN?=FN user function, returns everything between the function and the typing of ?= as a list (like an arglist). Returns NIL if ?= was typed immediately after the function name.

(TTYIN.PRINTARGS FN ARGS ACTUALS ARGTYPE) [Function]

Does the function / argument printing for ?=. *ARGS* is an argument list, *ACTUALS* is a list of actual parameters (from the typein) to match up with args. *ARGTYPE* is a value of the function *ARGTYPE*; it defaults to (*ARGTYPE FN*).

Read Macros

When doing READ input in Interlisp-10, no Lisp-style read macros are available (but the ' and control-Y macros are built in). Principally because of the usefulness of the editor read macros (set by SETTERMCHARS), and the desire for a way of changing the meanings of the display editing commands, the following exists as a hack:

TTYINREADMACROS [Variable]

Value is a set of shorthand inputs useable during READ input. It is an alist of entries (CHARCODE . SYNONYM). If the user types the indicated character (the meta bit is denoted by the 200Q bit in the char code), TTYIN behaves as though the synonym character had been typed.

Special cases: 0 - the character is ignored; 200Q - pure meta bit; means to read another char and turn on its meta bit; 400Q - macro quote: read another char and use its original meaning. For example, if you have macros ((33Q . 200Q) (30Q . 33Q)), then Escape (33Q) will behave as an edit prefix, and control-X (30Q) will behave like Escape.

USER I/O PACKAGES

Note: currently, synonyms for meta commands are not well-supported, working only when the command is typed with no argument.

Slightly more powerful macros also can be supplied; they are recognized when a character is typed on an empty line, i.e., as the first thing after the prompt. In this case, the TTYINREADMACROS entry is of the form (CHARCODE T . RESPONSE) or (CHARCODE CONDITION . RESPONSE), where CONDITION is a list that evaluates true. If RESPONSE is a list, it is EVALed; otherwise it is left unevaluated. The result of this evaluation (or RESPONSE itself) is treated as follows:

- NIL The macro is ignored and the character reads normally, i.e., as though TTYINREADMACROS had never existed.
- An integer A character code, treated as above. Special case: -1 is treated like 0, but says that the display may have been altered in the evaluation of the macro, so TTYIN should reset itself appropriately.
- Anything else This TTYIN input is terminated (with a crlf) and returns the value of "response" (turned into a list if necessary). This is the principal use of this facility. The macro character thus stands for the (possibly computed) response, terminated if necessary with a crlf. The original character is not echoed.

Interrupt characters, of course, cannot be read macros, as TTYIN never sees them, but any other characters, even non-control chars, are allowed. The ability to return NIL allows you to have conditional macros that only apply in specified situations (e.g., the macro might check the prompt (LISPXID) or other contextual variables). To use this specifically to do immediate editor read macros, do the following for each edit command and character you want to invoke it with:

```
(ADDTTOVAR TTYINREADMACROS (CHARCODE 'CHARMACRO?
EDITCOM) )
```

For example, (ADDTTOVAR TTYINREADMACROS (12Q CHARMACRO? !NX)) will make linefeed do the !NX command. Note that this will only activate linefeed at the beginning of a line, not anywhere in the line. There will probably be a user function to do this in the next release.

Note that putting (12Q T . !NX) on TTYINREADMACROS would also have the effect of returning !NX from the READ call so that the editor would do an !NX. However, TTYIN would also return !NX outside the editor (probably resulting in a u.b.a. error, or convincing DWIM to enter the editor), and also the clearing of the output buffer (performed by CHARMACRO?) would not happen.

Assorted Flags

These flags control aspects of TTYIN's behavior. Some have already been mentioned. In Interlisp-D, the flags are all initially set to T.

INTERLISP-D REFERENCE MANUAL

TYPEAHEADFLG [Variable]

If true, TTYIN always permits typeahead; otherwise it clears the buffer for any but LISPXREAD input.

?ACTIVATEFLG [Variable]

If true, enables the feature whereby ? lists alternative completions from the current spelling list.

SHOWPARENFLG [Variable]

If true, then whenever you are typing Lisp input and type a right parenthesis/bracket, TTYIN will briefly move the cursor to the matching parenthesis/bracket, assuming it is still on the screen. The cursor stays there for about 1 second, or until you type another character (i.e., if you type fast you'll never notice it). This feature was inspired by a similar EMACS feature, and turned out to be pretty easy to implement.

TTYINBSFLG [Variable]

Causes TTYIN to always physically backspace, even if you're running on a non-display (not a DM or Heath), rather than print \deletedtext\ (this assumes your hardcopy terminal or glass tty is capable of backspacing). If TTYINBSFLG is LF, then in addition to backspacing, TTYIN x's out the deleted characters as it backs up, and when you stop deleting, it outputs a linefeed to drop to a new, clean line before resuming. To save paper, this linefeed operation is not done when only a single character is deleted, on the grounds that you can probably figure out what you typed anyway.

TTYINRESPONSES [Variable]

An association list of special responses that will be handled by routines designated by the programmer. See "Special Responses", below.

TTYINERRORSETFLG [Variable]

[Interlisp-D only] If true, non-LISPXREAD inputs are errorset-protected (Control-E traps back to the prompt), otherwise errors propagate upwards. Initially NIL.

TTYINCOMMENTCHAR [Variable]

This variable affects the treatment of lines beginning with the comment character (usually ";"). If TTYINCOMMENTCHAR is a character code, and the first character on a line of typein is equal to TTYINCOMMENTCHAR, then the line is erased from the screen and no input function will see it. If TTYINCOMMENTCHAR is NIL, this feature is disabled. TTYINCOMMENTCHAR is initially NIL.

TTYINCOMPLETEFLG [Variable]

If true, enables Escape completion from USERWORDS during READ inputs. Details below.

USER I/O PACKAGES

USERWORDS (see Chapter 20) contains words you mentioned recently: functions you have defined or edited, variables you have set or evaluated at the executive level, etc. This happens to be a very convenient list for context-free escape completion; if you have recently edited a function, chances are good you may want to edit it again (typing "EF xx\$") or type a call to it. If there is no completion for the current word from USERWORDS, the escape echoes as "\$", i.e. nothing special happens; if there is more than one possible completion, you get beeped. If typed when not inside a word, Escape completes to the value of LASTWORD, i.e., the last thing you typed that the p.a. "noticed" (setting TTYINCOMPLETEFLG to 0 disables this latter feature), except that Escape at the beginning of the line is left alone (it is a p.a. command).

If you really wanted to enter an escape, you can, of course, just quote it with a control-V, like you can other control chars.

You may explicitly add words to USERWORDS yourself that wouldn't get there otherwise. To make this convenient online the edit command [←] means "add the current atom to USERWORDS" (you might think of the command as "pointing out this atom"). For example, you might be entering a function definition and want to "point to" one or more of its arguments or prog variables. Giving an argument of zero to this command will instead remove the indicated atom from USERWORDS.

Note that this feature loses some of its value if the spelling list is too long, for then the completion takes too long computationally and, more important, there are too many alternative completions for you to get by with typing a few characters followed by escape. Lisp's maintenance of the spelling list USERWORDS keeps the "temporary" section (which is where everything goes initially unless you say otherwise) limited to #USERWORDS atoms, initially 100. Words fall off the end if they haven't been used (they are "used" if FIXSPELL corrects to one, or you use <escape> to complete one).

Special Responses

There is a facility for handling "special responses" during any non-READ TTYIN input. This action is independent of the particular call to TTYIN, and exists to allow you to effectively "advise" TTYIN to intercept certain commands. After the command is processed, control returns to the original TTYIN call. The facility is implemented via the list TTYINRESPONSES.

TTYINRESPONSES

[Variable]

TTYINRESPONSES is a list of elements, each of the form:

(COMMANDS RESPONSE-FORM OPTION)

COMMANDS is a single atom or list of commands to be recognized; RESPONSE-FORM is EVALed (if a list), or APPLIED (if an atom) to the command and the rest of the line. Within this form one can reference the free variables COMMAND (the command the user typed) and LINE (the rest of the line). If OPTION is the atom LINE, this means to pass the rest of line as a list; if it is STRING, this means to pass it as a string; otherwise, the command is only valid if there is nothing else on the line. If RESPONSE-FORM returns the atom IGNORE, it is not treated as a special response (i.e. the input is returned normally as the result of TTYIN).

INTERLISP-D REFERENCE MANUAL

Suggested use: global commands or options can be added to the toplevel value of `TTYINRESPONSES`. For more specialized commands, rebind `TTYINRESPONSES` to `(APPEND NEWENTRIES TTYINRESPONSES)` inside any module where you want to do this sort of special processing.

Special responses are not checked for during `READ`-style input.

Display Types

[This is not relevant in Interlisp-D]

`TTYIN` determines the type of display by calling `DISPLAYTERMP`, which is initially defined to test the value of the `GTTYP` jsys. It returns either `NIL` (for printing terminals) or a small number giving `TTYIN`'s internal code for the terminal type. The types `TTYIN` currently knows about:

- 0 = glass tty (capable of deleting chars by backspacing, but little else)
- 1 = Datamedia
- 2 = Heath

Only the Datamedia has full editing power. `DISPLAYTERMP` has built into it the correct terminal types for Sumex and Stanford campus 20's: Datamedia = 11 on tenex, 5 on tops20; Heath = 18 on Tenex, 25 on tops20. You can override those values by setting the variable `DISPLAYTYPES` to be an association list associating the `GTTYP` value with one of these internal codes. For example, Sumex displays correspond to `DISPLAYTYPES = ((11 . 1) (18 . 2))` [although this is actually compiled into `DISPLAYTERMP` for speed]. Any display terminal other than Datamedia and Heath can probably safely be assigned to "0" for glass tty.

To add new terminal types, you have to choose a number for it, add new code to `TTYIN` for it and recompile. The `TTYIN` code specifies what the capabilities of the terminal are, and how to do the primitive operations: up, down, left, right, address cursor, erase screen, erase to end of line, insert character, etc.

For terminals lacking a meta key (currently only Datamedias have it), set the variable `EDITPREFIXCHAR` to the ascii code of an edit "prefix" (i.e., anything typed preceded by the prefix is considered to have the meta bit on). If your `EDITPREFIXCHAR` is 33Q (Escape), you can type a real Escape by typing 3 of them (2 won't do, since that means "Meta-Escape", a legitimate argument to another command). You could also define an Escape synonym with `TTYINREADMACROS` if you wanted (but currently it doesn't work in filename completion). Setting `EDITPREFIXCHAR` for a terminal that is not equipped to handle the full range of editing functions (only the Heath and Datamedia are currently so equipped) is not guaranteed to work, i.e. the display will not always be up to date; but if you can keep track of what you're doing, together with an occasional control-R to help out, go right ahead.

Prettyprint

The standard way of printing out function definitions (on the terminal or into files) is to use PRETTYPRINT.

(PRETTYPRINT *FNS* PRETTYDEFLG $\rightarrow$) [Function]

FNS is a list of functions. If *FNS* is atomic, its value is used). The definitions of the functions are printed in a pretty format on the primary output file using the primary readable. For example, if FACTORIAL were defined by typing

```
(DEFINEQ (FACTORIAL [LAMBDA (N) (COND ((ZEROP N) 1)
(T (ITIMES N (FACTORIAL (SUB1 N))

(PRETTYPRINT '(FACTORIAL)) would print out
(FACTORIAL
  [LAMBDA (N)
    (COND
      ((ZEROP N)
        1)
      (T (ITIMES N (FACTORIAL (SUB1 N))
```

PRETTYDEFLG is T when called from PRETTYDEF (and hence MAKEFILE). Among other actions taken when this argument is true, PRETTYPRINT indicates its progress in writing the current output file: whenever it starts a new function, it prints on the terminal the name of that function if more than 30 seconds (real time) have elapsed since the last time it printed the name of a function.

Prettyprint operates correctly on functions that are BROKEN, BROKEN-IN, ADVISED, or have been compiled with their definitions saved on their property lists: it prints the original, pristine definition, but does not change the current state of the function. If a function is not defined but is known to be on one of the files noticed by the file package, PRETTYPRINT loads in the definition (using LOADFNS) and prints it (except when called from PRETTYDEF). If PRETTYPRINT is given an atom which is not the name of a function, but has a value, it prettyprints the value. Otherwise, PRETTYPRINT attempts spelling correction. If all fails, PRETTYPRINT returns (FN NOT PRINTABLE). Note that PRETTYPRINT will return (FN NOT PRINTABLE) if FN does not have an accessible expr definition, or if it doesn't have any definition at all.

(PP *FN* ... *FN*) [NLambda NoSpread Function]

For prettyprinting functions to the terminal. PP calls PRETTYPRINT with the primary output file set to T and the primary read table set to T. The primary output file and primary readable are restored after printing.

(PP FOO) is equivalent to (PRETTYPRINT '(FOO)); (PP FOO FIE) is equivalent to (PRETTYPRINT '(FOO FIE)).

As described above, when PRETTYPRINT, and hence PP, is called with the name of a function that is not defined, but whose definition is on a file known to the file package, the definition is automatically

INTERLISP-D REFERENCE MANUAL

read in and then prettyprinted. However, if the user does not intend on editing or running the definition, but simply wants to see the definition, the function PF described below can be used to simply copy the corresponding characters from the file to the terminal. This results in a savings in both space and time, since it is not necessary to allocate storage to actually read in the definition, and it is not necessary to re-prettyprint it (since the function is already in prettyprint format on the file).

(**PF** *FN FROMFILES TOFILE*) [NLambda NoSpread Function]

Copies the definition of *FN* found on each of the files in *FROMFILES* to *TOFILE*. If *TOFILE* = NIL, defaults to T. If *FROMFILES* = NIL, defaults to (WHEREIS *FN* NIL T) (see Chapter 17). The typical usage of PF is simply to type "PF *FN*".

PF prints a message if it can't find a file on *FROMFILES*, or it can't find the function *FN* on a file.

When printing to the terminal, PF performs several transformations on the characters in the file that comprise the definition for *FN*:

1. Font information is stripped out (except in Interlisp-D, whose display supports multiple fonts)
2. Occurrences of the CHANGECHAR (see the Special Prettyprint Controls section below) are not printed
3. Since functions typically tend to be printed to a file with a larger linelength than when printing to a terminal, the number of leading spaces on each line is cut in half (unless PFDEFAULT is T; initially NIL)
4. Comments are elided, if **COMMENT**FLG is non-NIL (see the Comment Feature section below).

(**SEE** *FROMFILE TOFILE*) [NLambda NoSpread Function]

Copies all of the text from *FROMFILE* to *TOFILE* (defaults to T), processing all text as PF does. Used to display the contents of files on the terminal.

(**PP*** *X*) [NLambda NoSpread Function]

(**PF*** *FN FROMFILES TOFILE*) [NLambda NoSpread Function]

(**SEE*** *FROMFILE TOFILE*) [NLambda NoSpread Function]

These functions operate exactly like PP, PF, and SEE, except that they bind **COMMENT**FLG to NIL, so comments are printed in full.

While the function PRETTYPRINT prints entire function definitions, the function PRINTDEF can be used to print parts of functions, or arbitrary Interlisp structures:

(**PRINTDEF** *EXPR LEFT DEF TAILFLG FNSLST FILE*) [Function]

Prints the expression *EXPR* in a pretty format on *FILE* using the primary readtable. *LEFT* is the left hand margin (LINELENGTH determines the right hand margin). PRINTDEF

USER I/O PACKAGES

initially performs (TAB LEFT T), which means to space to position *LEFT*, unless already beyond this position, in which case it does nothing.

DEF = T means *EXPR* is a function definition, or a piece of one. If *DEF* = NIL, no special action is taken for LAMBDA's, PROG's, COND's, comments, CLISP, etc. DEF is NIL when PRETTYDEF calls PRETTYPRINT to print variables and property lists, and when PRINTDEF is called from the editor via the command PPV.

TAILFLG = T means *EXPR* is interpreted as a tail of a list, to be printed without parentheses.

FNSLST is for use for printing with multiple fonts (see Chapter 27). PRINTDEF prints occurrences of any function in the list FNSLST in a different font, for emphasis. MAKEFILE passes as FNSLST the list of all functions on the file being made.

Comment Feature

A facility for annotating Interlisp functions is provided in PRETTYPRINT. Any expression beginning with the atom * is interpreted as a comment and printed in the right margin. Example:

```
(FACTORIAL
  [LAMBDA (N)
    (COND
      ((ZEROP N)
        1)
      (T
        (* RECURSIVE DEFINITION:
          N! = N*N-1!)
        (ITIMES N (FACTORIAL (SUB1 N))
```

These comments actually form a part of the function definition. Accordingly, * is defined as an nlambda nospread function that returns its argument, similar to QUOTE. When running an interpreted function, * is entered the same as any other Interlisp function. Therefore, comments should only be placed where they will not harm the computation, i.e., where a quoted expression could be placed. For example, writing

```
(ITIMES N (FACTORIAL (SUB1 N)) (* RECURSIVE
DEFINITION))
```

in the above function would cause an error when ITIMES attempted to multiply N, N-1!, and RECURSIVE.

For compilation purposes, * is defined as a macro which compiles into no instructions (unless the comment has been placed where it has been used for value, in which case the compiler prints an appropriate error message and compiles * as QUOTE). Thus, the compiled form of a function with comments does not use the extra atom and list structure storage required by the comments in the source (interpreted) code. This is the way the comment feature is intended to be used.

INTERLISP-D REFERENCE MANUAL

A comment of the form `(* E X)` causes `X` to be evaluated at prettyprint time, as well as printed as a comment in the usual way. For example, `(* E (RADIX 8))` as a comment in a function containing octal numbers can be used to change the radix to produce more readable printout.

The comment character `*` is stored in the variable `COMMENTFLG`. The user can set it to some other value, e.g. `";`, and use this to indicate comments.

COMMENTFLG [Variable]

If `CAR` of an expression is `EQ` to `COMMENTFLG`, the expression is treated as a comment by `PRETTYPRINT`. `COMMENTFLG` is initialized to `*`. Note that whatever atom is chosen for `COMMENTFLG` should also have an appropriate function definition and compiler macro, for example, by copying those of `*`.

Comments are designed mainly for documenting listings. Therefore, when prettyprinting to the terminal, comments are suppressed and printed as the string `**COMMENT**`. The value of `**COMMENT**FLG` determines the action.

****COMMENT**FLG** [Variable]

If `**COMMENT**FLG` is `NIL`, comments are printed. Otherwise, the value of `**COMMENT**FLG` is printed. Initially `"**COMMENT**"`.

(COMMENT1 L —) [Function]

Prints the comment `L`. `COMMENT1` is a separate function to permit the user to write prettyprint macros that use the regular comment printer. For example, to cause comments to be printed at a larger than normal linelength, one could put an entry for `*` on `PRETTYPRINTMACROS`:

```
(* LAMBDA (X) (RESETFORM (LINELENGTH 100) (COMMENT1 X)))
```

This macro resets the line length, prints the comment, and then restores the line length.

`COMMENT1` expects to be called from within the environment established by `PRINTDEF`, so ordinarily the user should call it only from within prettyprint macros.

Comment Pointers

For a well-commented collection of programs, the list structure, atom, and print name storage required to represent the comments in core can be significant. If the comments already appear on a file and are not needed for editing, a significant savings in storage can be achieved by simply leaving the text of the comment on the file when the file is loaded, and instead retaining in core only a pointer to the comment. When this feature is enabled, `*` is defined as a read macro (see Chapter 25) in `FILERDTBL` which, instead of reading in the entire text of the comment, constructs an expression containing

USER I/O PACKAGES

- The name of the file in which the text of the comment is contained
- The address of the first character of the comment
- The number of characters in the comment
- A flag indicating whether the comment appeared at the right hand margin or centered on the page

For output purposes, `*` is defined on `PRETTYPRINTMACROS` (see the Prettyprint Control Functions section below) so that it prints the comments represented by such pointers by simply copying the corresponding characters from one file to another, or to the terminal. Normal comments are processed the same as before, and can be intermixed freely with comment pointers.

The comment pointer feature is controlled by the function `NORMALCOMMENTS`.

(`NORMALCOMMENTS` *FLG*) [Function]

If *FLG* is `NIL`, the comment pointer feature is enabled. If *FLG* is `T`, the comment pointer feature is disabled (the default).

`NORMALCOMMENTS` can be changed as often as desired. Thus, some files can be loaded normally, and others with their comments converted to comment pointers.

For convenience of editing selected comments, an edit macro, `GET*`, is included, which loads in the text of the corresponding comment. The editor's `PP*` command, in contrast, prints the comment without reading it by simply copying the corresponding characters to the terminal. `GET*` is defined in terms of `GETCOMMENT`:

(`GETCOMMENT` *X* *DESTFL* *-*) [Function]

If *X* is a comment pointer, replaces *X* with the actual text of the comment, which it reads from its file. Returns *X* in all cases. If *DESTFL* is non-`NIL`, it is the name of an open file, to which `GETCOMMENT` copies the comment; in this case, *X* remains a comment pointer, but it has been changed to point to the new file (unless `NORMALCOMMENTS` has been set to `DONTUPDATE`).

(`PRINTCOMMENT` *X*) [Function]

Defined as the prettyprint macro for `*`: copies the comment to the primary output file by using `GETCOMMENT`.

(`READCOMMENT` *FL* *RDTBL* *LST*) [Function]

Defined as the read macro for `*` in `FILERDTBL`: if `NORMALCOMMENTSFLG` is `NIL`, it constructs a comment pointer, unless it believes the expression beginning with `*` is not actually a comment, e.g., if the next atom is `"`, `.` or `E`.

INTERLISP-D REFERENCE MANUAL

Note that a certain amount of care is required in using the comment pointer feature. Since the text of the comment resides on the file pointed to by the comment pointer, that file must remain in existence as long as the comment is needed. `GETCOMMENT` helps out by changing the comment pointer to always point at the most recent file that the comment lives on. However, if the user has been performing repeated `MAKEFILE`'s (see Chapter 17) in which differing functions have changed at each invocation of `MAKEFILE`, it is possible for the comment pointers in memory to be pointing at several versions of the same file, since a comment pointer is only updated when the function it lives in is prettyprinted, not when the function has been copied verbatim to the new file. This can be a problem for file systems that have a built-in limit on the number of versions of a given file that will be made before old versions are expunged. In such a case, the user should set the version retention count of any directories involved to be infinite. `GETCOMMENT` prints an error message if the file that the comment pointer points at has disappeared.

Similarly, one should be cognizant of comment pointers in sysouts, and be sure to retain any files thus pointed to.

When using comment pointers, the user should also not set `PRETTYFLG` to `NIL` or call `MAKEFILE` with option `FAST`, since this will prevent functions from being prettyprinted, and hence not get the text of the comment copied into the new file.

If the user changes the value of `COMMENTFLG` but still wishes to use the comment pointer feature, the new `COMMENTFLG` should be given the same read-macro definition in `FILERDTBL` as `*` has, and the same entry be put on `PRETTYPRINTMACROS`. For example, if `COMMENTFLG` is reset to be `" ; "`, then `(SETSYNTAX ' ; ' * FILERDTBL)` should be performed, and `( ; . PRINTCOMMENT)` added to `PRETTYPRINTMACROS`.

Converting Comments to Lower Case

This section is for users using terminals without lower case, who nevertheless would like their comments to be converted to lower case for more readable listings. If the second atom in a comment is `%%`, the text of the comment is converted to lower case so that it looks like English instead of Lisp. Note that comments are converted only when they are actually written to a file by `PRETTYPRINT`.

The algorithm for conversion to lower case is the following: If the first character in an atom is `^`, do not change the atom (but remove the `^`). If the first character is `%`, convert the atom to lower case. Note that the user must type `%%` as `%` is the escape character. If the atom (minus any trailing punctuation marks) is an Interlisp word (i.e., is a bound or free variable for the function containing the comment, or has a top level value, or is a defined function, or has a non-`NIL` property list), do not change it. Otherwise, convert the atom to lower case. Conversion only affects the upper case alphabet, i.e., atoms already converted to lower case are not changed if the comment is converted again. When converting, the first character in the comment and the first character following each period are left capitalized. After conversion, the comment is physically modified to be the lower case text minus the `%%` flag, so that conversion is thus only performed once (unless the user edits the comment inserting additional upper case text and another `%%` flag).

USER I/O PACKAGES

LCASELST [Variable]

Words on LCASELST will always be converted to lower case. LCASELST is initialized to contain words which are Interlisp functions but also appear frequently in comments as English words (AND, EVERY, GET, GO, LAST, LENGTH, LIST, etc.). Therefore, if one wished to type a comment including the lisp function GO, it would be necessary to type ↑GO in order that it might be left in upper case.

UCASELST [Variable]

Words on UCASELST (that do not appear on LCASELST) will be left in upper case. UCASELST is initialized to NIL.

ABBREVLST [Variable]

ABBREVLST is used to distinguish between abbreviations and words that end in periods. Normally, words that end in periods and occur more than halfway to the right margin cause carriage-returns. Furthermore, during conversion to lowercase, words ending in periods, except for those on ABBREVLST, cause the first character in the next word to be capitalized. ABBREVLST is initialized to the upper and lower case forms of ETC., I.E., and E.G..

Special Prettyprint Controls

PRETTYTABFLG [Variable]

In order to save space on files, tabs are used instead of spaces for the initial spaces on each line, assuming that each tab corresponds to 8 spaces. This results in a reduction of file size by about 30%. Tabs are not used if PRETTYTABFLG is set to NIL (initially T).

#RPARS [Variable]

Controls the number of right parentheses necessary for square bracketing to occur. If #RPARS = NIL, no brackets are used. #RPARS is initialized to 4.

FIRSTCOL [Variable]

The starting column for comments. Comments run between FIRSTCOL and the line length set by LINELENGTH (see Chapter 25). If a word in a comment ends with a "." and is not on the list ABBREVLST, and the position is greater than halfway between FIRSTCOL and LINELENGTH, the next word in the comment begins on a new line. Also, if a list is encountered in a comment, and the position is greater than halfway, the list begins on a new line.

PRETTYLCOM [Variable]

If a comment has more than PRETTYLCOM elements (using COUNT), it is printed starting at column 10, instead of FIRSTCOL. Comments are also printed starting at column 10 if their second element is also a *, i.e., comments of the form (* * --).

INTERLISP-D REFERENCE MANUAL

#CAREFULCOLUMNS [Variable]

In the interests of efficiency, PRETTYPRINT approximates the number of characters in each atom, rather than calling NCHARS, when computing how much will fit on a line. This procedure works satisfactorily in most cases. However, users with unusually long atoms in their programs, e.g., such as produced by CLISPIFY, may occasionally encounter some glitches in the output produced by PRETTYPRINT. The value of #CAREFULCOLUMNS tells PRETTYPRINT how many columns (counting from the right hand margin) in which to actually compute NCHARS instead of approximating. Setting #CAREFULCOLUMNS to 20 or 30 will eliminate the glitches, although it will slow down PRETTYPRINT slightly. #CAREFULCOLUMNS is initially 0.

(WIDEPAPER *FLG*) [Function]

(WIDEPAPER *T*) sets FILELINELENGTH (see Chapter 25), FIRSTCOL, and PRETTYLCOM to large values appropriate for pretty printing files to be listed on wide paper. (WIDEPAPER) restores these parameters to their initial values. WIDEPAPER returns the previous setting of *FLG*.

PRETTYFLG [Variable]

If PRETTYFLG is NIL, PRINTDEF uses PRIN2 instead of prettyprinting. This is useful for producing a fast symbolic dump (see the FAST option of MAKEFILE in Chapter 17). Note that the file loads the same as if it were prettyprinted. PRETTYFLG is initially set to T. PRETTYFLG should not be set to NIL if comment pointers are being used.

CLISPIFYPRETTYFLG [Variable]

Used to inform PRETTYPRINT to call CLISPIFY on selected function definitions before printing them (see Chapter 21).

PRETTYPRINTMACROS [Variable]

An association-list that enables the user to control the formatting of selected expressions. CAR of each expression being PRETTYPRINTed is looked up on PRETTYPRINTMACROS, and if found, CDR of the corresponding entry is applied to the expression. If the result of this application is NIL, PRETTYPRINT ignores the expression; i.e., it prints nothing, assuming that the prettyprintmacro has done any desired printing. If the result of applying the prettyprint macro is non-NIL, the result is prettyprinted in the normal fashion. This gives the user the option of computing some other expression to be prettyprinted in its place.

Note: "prettyprinted in the normal fashion" includes processing prettyprint macros, unless the prettyprint macro returns a structure EQ to the one it was handed, in which case the potential recursion is broken.

PRETTYPRINTTYPENAME [Variable]

A list of elements of the form (TYPENAME . FN). For types other than lists and atoms, the type name of each datum to be prettyprinted is looked up on

USER I/O PACKAGES

PRETTYPRINTYPEMACROS, and if found, the corresponding function is applied to the datum about to be printed, instead of simply printing it with PRIN2.

PRETTYEQUIVLST

[Variable]

An association-list that tells PRETTYPRINT to treat a CAR-of-form the same as some other CAR-of-form. For example, if (QLAMBDA . LAMBDA) appears on PRETTYEQUIVLST, then expressions beginning with QLAMBDA are prettyprinted the same as LAMBDA. Currently, PRETTYEQUIVLST only allows (i.e., supports in an interesting way) equivalences to forms that PRETTYPRINT internally handles. Equivalence to forms for which the user has specified a prettyprint macro should be made by adding further entries to PRETTYPRINTMACROS

CHANGECHAR

[Variable]

If non-NIL, and PRETTYPRINT is printing to a file or display terminal, PRETTYPRINT prints CHANGECHAR in the right hand margin while printing those expressions marked by the editor as having been changed (see Chapter 16). CHANGECHAR is initially |.

26. GRAPHICS OUTPUT OPERATIONS

Streams are used as the basis for all I/O operations. Files are implemented as streams that can support character printing and reading operations, and file pointer manipulation. An image stream is a type of stream that also provides an interface for graphical operations. All of the operations that can be applied to streams can be applied to image streams. For example, an image stream can be passed as the argument to `PRINT`, to print something on an image stream. In addition, special functions are provided to draw lines and curves and perform other graphical operations. Calling these functions on a stream that is not an image stream will generate an error.

Primitive Graphics Concepts

The Interlisp-D graphics system is based on manipulating bitmaps (rectangular arrays of pixels), positions, regions, and textures. These objects are used by all of the graphics functions.

Positions

A position denotes a point in an X, Y coordinate system. A `POSITION` is an instance of a record with fields `XCOORD` and `YCOORD` and is manipulated with the standard record package facilities. For example, `(create POSITION XCOORD ← 10 YCOORD ← 20)` creates a position representing the point (10,20).

(POSITIONP *X*) [Function]

Returns *X* if *X* is a position; `NIL` otherwise.

Regions

A Region denotes a rectangular area in a coordinate system. Regions are characterized by the coordinates of their bottom left corner and their width and height. A `REGION` is a record with fields `LEFT`, `BOTTOM`, `WIDTH`, and `HEIGHT`. It can be manipulated with the standard record package facilities. There are access functions for the `REGION` record that return the `TOP` and `RIGHT` of the region.

The following functions are provided for manipulating regions:

(CREATEREGION *LEFT BOTTOM WIDTH HEIGHT*) [Function]

Returns an instance of the `REGION` record which has *LEFT*, *BOTTOM*, *WIDTH* and *HEIGHT* as respectively its *LEFT*, *BOTTOM*, *WIDTH*, and *HEIGHT* fields.

INTERLISP-D REFERENCE MANUAL

Example: (CREATEREGION 10 -20 100 200) will create a region that denotes a rectangle whose width is 100, whose height is 200, and whose lower left corner is at the position (10,-20).

(**REGIONP** *X*) [Function]

Returns *X* if *X* is a region, NIL otherwise.

(**INTERSECTREGIONS** *REGION REGION ... REGION*) [NoSpread Function]

Returns a region which is the intersection of a number of regions. Returns NIL if the intersection is empty.

(**UNIONREGIONS** *REGION REGION ... REGION*) [NoSpread Function]

Returns a region which is the union of a number of regions, i.e. the smallest region that contains all of them. Returns NIL if there are no regions given.

(**REGIONSINTERSECTP** *REGION REGION*) [Function]

Returns T if *REGION* intersects *REGION*. Returns NIL if they do not intersect.

(**SUBREGIONP** *LARGEREGION SMALLREGION*) [Function]

Returns T if *SMALLREGION* is a subregion (is equal to or entirely contained in) *LARGEREGION*; otherwise returns NIL.

(**EXTENDREGION** *REGION INCLUDEREGION*) [Function]

Changes (destructively modifies) the region *REGION* so that it includes the region *INCLUDEREGION*. It returns *REGION*.

(**MAKEWITHINREGION** *REGION LIMITREGION*) [Function]

Changes (destructively modifies) the left and bottom of the region *REGION* so that it is within the region *LIMITREGION*, if possible. If the dimension of *REGION* are larger than *LIMITREGION*, *REGION* is moved to the lower left of *LIMITREGION*. If *LIMITREGION* is NIL, the value of the variable *WHOLEDISPLAY* (the screen region) is used. **MAKEWITHINREGION** returns the modified *REGION*.

(**INSIDEP** *REGION POSORX Y*) [Function]

If *POSORX* and *Y* are numbers, it returns T if the point (*POSORX*, *Y*) is inside of *REGION*. If *POSORX* is a *POSITION*, it returns T if *POSORX* is inside of *REGION*. If *REGION* is a *WINDOW*, the window's interior region in window coordinates is used. Otherwise, it returns NIL.

Bitmaps

The display primitives manipulate graphical images in the form of bitmaps. A bitmap is a rectangular array of "pixels," each of which is an integer representing the color of one point in the bitmap image. A bitmap is created with a specific number of bits allocated for each pixel. Most bitmaps used for the display screen use one bit per pixel, so that at most two colors can be represented. If a pixel is 0, the corresponding location on the image is white. If a pixel is 1, its location is black. This interpretation can be changed for the display screen with the function `VIDEOCOLOR`. Bitmaps with more than one bit per pixel are used to represent color or grey scale images. Bitmaps use a positive integer coordinate system with the lower left corner pixel at coordinate (0,0). Bitmaps are represented as instances of the datatype `BITMAP`. Bitmaps can be saved on files with the `VAR`s file package command.

(`BITMAPCREATE` *WIDTH HEIGHT BITSPERPIXEL*) [Function]

Creates and returns a new bitmap which is *WIDTH* pixels wide by *HEIGHT* pixels high, with *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is `NIL`, it defaults to 1.

(`BITMAPP` *X*) [Function]

Returns *X* if *X* is a bitmap, `NIL` otherwise.

(`BITMAPWIDTH` *BITMAP*) [Function]

Returns the width of *BITMAP* in pixels.

(`BITMAPHEIGHT` *BITMAP*) [Function]

Returns the height of *BITMAP* in pixels.

(`BITSPERPIXEL` *BITMAP*) [Function]

Returns the number of bits per pixel of *BITMAP*.

(`BITMAPBIT` *BITMAP X Y NEWVALUE*) [Function]

If *NEWVALUE* is between 0 and the maximum value for a pixel in *BITMAP*, the pixel (*X*, *Y*) is changed to *NEWVALUE* and the old value is returned. If *NEWVALUE* is `NIL`, *BITMAP* is not changed but the value of the pixel is returned. If *NEWVALUE* is anything else, an error is generated. If (*X*, *Y*) is outside the limits of *BITMAP*, 0 is returned and no pixels are changed. *BITMAP* can also be a window or display stream. Note: non-window image streams are "write-only"; the *NEWVALUE* argument must be non-`NIL`.

(`BITMAPCOPY` *BITMAP*) [Function]

Returns a new bitmap which is a copy of *BITMAP* (same dimensions, bits per pixel, and contents).

(`EXPANDBITMAP` *BITMAP WIDTHFACTOR HEIGHTFACTOR*) [Function]

Returns a new bitmap that is *WIDTHFACTOR* times as wide as *BITMAP* a

INTERLISP-D REFERENCE MANUAL

nd *HEIGHTFACTOR* times as high. Each pixel of *BITMAP* is copied into a *WIDTHFACTOR* times *HEIGHTFACTOR* block of pixels. If NIL, *WIDTHFACTOR* defaults to 4, *HEIGHTFACTOR* to 1.

(**ROTATEBITMAP** *BITMAP*) [Function]

Given an m-high by n-wide bitmap, this function returns an n-high by m-wide bitmap. The returned bitmap is the image of the original bitmap, rotated 90 degrees clockwise.

(**SHRINKBITMAP** *BITMAP* *WIDTHFACTOR* *HEIGHTFACTOR* *DESTINATIONBITMAP*) [Function]

Returns a copy of *BITMAP* that has been shrunk by *WIDTHFACTOR* and *HEIGHTFACTOR* in the width and height, respectively. If NIL, *WIDTHFACTOR* defaults to 4, *HEIGHTFACTOR* to 1. If *DESTINATIONBITMAP* is not provided, a bitmap that is $1/WIDTHFACTOR$ by $1/HEIGHTFACTOR$ the size of *BITMAP* is created and returned. *WIDTHFACTOR* and *HEIGHTFACTOR* must be positive integers.

(**PRINTBITMAP** *BITMAP* *FILE*) [Function]

Prints the bitmap *BITMAP* on the file *FILE* in a format that can be read back in by READBITMAP.

(**READBITMAP** *FILE*) [Function]

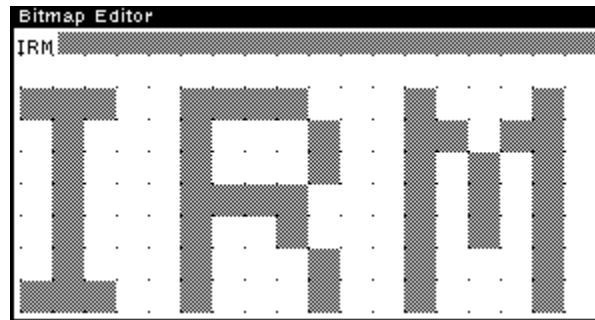
Creates a bitmap by reading an expression (written by PRINTBITMAP) from the file *FILE*.

(**EDITBM** *BMSPEC*) [Function]

EDITBM provides an easy-to-use interactive editing facility for various types of bitmaps. If *BMSPEC* is a bitmap, it is edited. If *BMSPEC* is an atom whose value is a bitmap, its value is edited. If *BMSPEC* is NIL, EDITBM asks for dimensions and creates a bitmap. If *BMSPEC* is a region, that portion of the screen bitmap is used. If *BMSPEC* is a window, it is brought to the top and its contents edited.

EDITBM sets up the bitmap being edited in an editing window. The editing window has two major areas: a gridded edit area in the lower part of the window and a display area in the upper left part. In the edit area, the left button will add points, the middle button will erase points. The right button provides access to the normal window commands to reposition and reshape the window. The actual size bitmap is shown in the display area. For example, the following is a picture of the bitmap editing window editing a eight-high by eighteen-wide bitmap:

GRAPHICS OUTPUT OPERATIONS

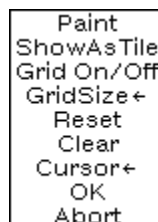


If the bitmap is too large to fit in the edit area, only a portion will be editable. This portion can be changed by scrolling both up and down in the left margin and left and right in the bottom margin. Pressing the middle button while in the display area will bring up a menu that allows global placement of the portion of the bitmap being edited. To allow more of the bitmap to be editing at once, the window can be reshaped to make it larger or the `GridSize←` command described below can be used to reduce the size of a bit in the edit area.

The bitmap editing window can be reshaped to provide more or less room for editing. When this happens, the space allocated to the editing area will be changed to fit in the new region.

Whenever the left or middle button is down and the cursor is not in the edit area, the section of the display of the bitmap that is currently in the edit area is complemented. Pressing the left button while not in the edit region will put the lower left 16 x 16 section of the bitmap into the cursor for as long as the left button is held down.

Pressing the middle button while not in either the edit area or the display area (i.e., while in the grey area in the upper right or in the title) will bring up a command menu.



There are commands to stop editing, to restore the bitmap to its initial state and to clear the bitmap. Holding the middle button down over a command will result in an explanatory message being printed in the prompt window. The commands are described below:

Paint Puts the current bitmap into a window and call the window `PAINT` command on it. The `PAINT` command implements drawing with various brush sizes and shapes but only on an actual sized bitmap. The `PAINT` mode is left by pressing the `RIGHT` button and selecting the `QUIT` command from

INTERLISP-D REFERENCE MANUAL

the menu. At this point, you will be given a choice of whether or not the changes you made while in PAINT mode should be made to the current bitmap.

- ShowAsTile** Tessellates the current bitmap in the upper part of the window. This is useful for determining how a bitmap will look if it were made the display background (using the function `CHANGEBACKGROUND`). Note: The tiled display will not automatically change as the bitmap changes; to update it, use the `ShowAsTile` command again.
- Grid, On/Off** Turns the editing grid display on or off.
- GridSize←** Allows specification of the size of the editing grid. Another menu will appear giving a choice of several sizes. If one is selected, the editing portion of the bitmap editor will be redrawn using the selected grid size, allowing more or less of the bitmap to be edited without scrolling. The original size is chosen heuristically and is typically about 8. It is particularly useful when editing large bitmaps to set the edit grid size smaller than the original.
- Reset** Sets all or part of the bitmap to the contents it had when `EDITBM` was called. Another menu will appear giving a choice between resetting the entire bitmap or just the portion that is in the edit area. The second menu also acts as a confirmation, since not selecting one of the choices on this menu results in no action being taken.
- Clear** Sets all or part of the bitmap to 0. As with the `Reset` command, another menu gives a choice between clearing the entire bitmap or just the portion that is in the edit area.
- Cursor←** Sets the cursor to the lower left part of the bitmap. This prompts the user to specify the cursor "hot spot" by clicking in the lower left corner of the grid.
- OK** Copies the changed image into the original bitmap, stops the bitmap editor and closes the edit windows. The changes the bitmap editor makes during the interaction occur on a copy of the original bitmap. Unless the bitmap editor is exited via `OK`, no changes are made in the original.
- Stop** Stops the bitmap editor without making any changes to the original bitmap.

Textures

A Texture denotes a pattern of gray which can be used to (conceptually) tessellate the plane to form an infinite sheet of gray. It is currently either a 4 by 4 pattern or a 16 by N ($N \leq 16$) pattern. Textures are created from bitmaps using the following function:

(**CREATETEXTUREFROMBITMAP** *BITMAP*) [Function]

Returns a texture object that will produce the texture of *BITMAP*. If *BITMAP* is too large, its lower left portion is used. If *BITMAP* is too small, it is repeated to fill out the texture.

GRAPHICS OUTPUT OPERATIONS

(**TEXTUREP** *OBJECT*)

[Function]

Returns *OBJECT* if it is a texture; NIL otherwise.

The functions which accept textures (**TEXTUREP**, **BITBLT**, **DSPTEXTURE**, etc.) also accept bitmaps up to 16 bits wide by 16 bits high as textures. When a region is being filled with a bitmap texture, the texture is treated as if it were 16 bits wide (if less, the rest is filled with white space).

The common textures white and black are available as system constants **WHITESHAE** and **BLACKSHAE**. The global variable **GRAYSHAE** is used by many system facilities as a background gray shade and can be set by the user.

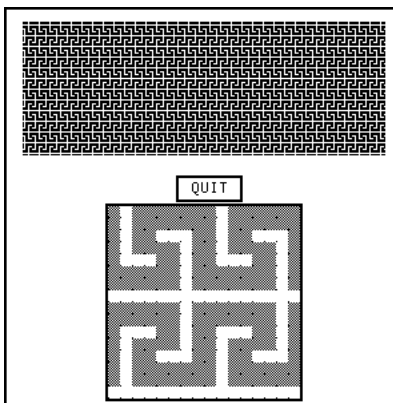
(**EDITSHAE** *SHAE*)

[Function]

Opens a window that allows the user to edit textures. Textures can be either small (4 by 4) patterns or large (16 by 16). In the edit area, the left button adds bits to the shade and the middle button erases bits from the shade. The top part of the window is painted with the current texture whenever all mouse keys are released. Thus it is possible to directly compare two textures that differ by more than one pixel by holding a mouse key down until all changes are made. When the "quit" button is selected, the texture being edited is returned.

If *SHAE* is a texture object, **EDITSHAE** starts with it. If *SHAE* is T, it starts with a large (16 by 16) white texture. Otherwise, it starts with **WHITESHAE**.

The following is a picture of the texture editor, editing a large (16 by 16) pattern:



Opening Image Streams

An image stream is an output stream which "knows" how to process graphic commands to a graphics output device. Besides accepting the normal character-output functions (**PRINT**, etc.), an image

INTERLISP-D REFERENCE MANUAL

stream can also be passed as an argument to functions to draw curves, to print characters in multiple fonts, and other graphics operations.

Each image stream has an "image stream type," a litatom that specifies the type of graphic output device that the image stream is processing graphics commands for. Currently, the built-in image stream types are `DISPLAY` (for the display screen), `INTERPRESS` (for Interpress format printers), and `PRESS` (for Press format printers). There are also library packages available that define image stream types for the IRIS display, 4045 printer, FX-80 printer, C150 printer, etc.

Image streams to the display (display streams) interpret graphics commands by immediately executing the appropriate operations to cause the desired image to appear on the display screen. Image streams for hardcopy devices such as Interpress printers interpret the graphic commands by saving information in a file, which can later be sent to the printer.

Note: Not all graphics operations can be properly executed for all image stream types. For example, `BITBLT` may not be supported to all printers. This functionality is still being developed, but even in the long run some operations may be beyond the physical or logical capabilities of some devices or image file formats. In these cases, the stream will approximate the specified image as best it can.

(**OPENIMAGESTREAM** *FILE* *IMAGETYPE* *OPTIONS*) [Function]

Opens and returns an image stream of type *IMAGETYPE* on a destination specified by *FILE*. If *FILE* is a file name on a normal file storage device, the image stream will store graphics commands on the specified file, which can be transmitted to a printer by explicit calls to `LISTFILES` and `SEND.FILE.TO.PRINTER`. If *IMAGETYPE* is `DISPLAY`, then the user is prompted for a window to open. *FILE* in this case will be used as the title of the window.

If *FILE* is a file name on the LPT device, this indicates that the graphics commands should be stored in a temporary file, and automatically sent to the printer when the image stream is closed by `CLOSEF`. *FILE* = `NIL` is equivalent to *FILE* = `{LPT}`. File names on the LPT device are of the form `{LPT}PRINTERNAME.TYPE`, where `PRINTERNAME`, `TYPE`, or both may be omitted. `PRINTERNAME` is the name of the particular printer to which the file will be transmitted on closing; it defaults to the first printer on `DEFAULTPRINTINGHOST` that can print *IMAGETYPE* files. The `TYPE` extension supplies the value of *IMAGETYPE* when it is defaulted (see below). `OPENIMAGESTREAM` will generate an error if the specified printer does not accept the kind of file specified by *IMAGETYPE*.

If *IMAGETYPE* is `NIL`, the image type is inferred from the extension field of *FILE* and the `EXTENSIONS` properties in the list `PRINTFILETYPES`. Thus, the extensions `IP`, `IPR`, and `INTERPRESS` indicate Interpress format, and the extension `PRESS` indicates Press format. If *FILE* is a printer file with no extension (of the form `{LPT}PRINTERNAME`), then *IMAGETYPE* will be the type that the indicated printer can print. If *FILE* has no extension but is not on the printer device `{LPT}`, then *IMAGETYPE* will default to the type accepted by the first printer on `DEFAULTPRINTINGHOST`.

GRAPHICS OUTPUT OPERATIONS

OPTIONS is a list in property list format, (PROP1 VAL1 PROP2 VAL2 —), used to specify certain attributes of the image stream; not all attributes are meaningful or interpreted by all types of image streams. Acceptable properties are:

REGION Value is the region on the page (in stream scale units, 0,0 being the lower-left corner of the page) that text will fill up. It establishes the initial values for DSPLEFTMARGIN, DSPRIGHTMARGIN, DSPBOTTOMMARGIN (the point at which carriage returns cause page advancement) and DSPTOPMARGIN (where the stream is positioned at the beginning of a new page).

If this property is not given, the value of the variable DEFAULTPAGEREGION, is used.

FONTS Value is a list of fonts that are expected to be used in the image stream. Some image streams (e.g. Interpress) are more efficient if the expected fonts are specified in advance, but this is not necessary. The first font in this list will be the initial font of the stream, otherwise the default font for that image stream type will be used.

HEADING Value is the heading to be placed automatically on each page. NIL means no heading.

Examples: Suppose that Tremor: is an Interpress printer, Quake is a Press printer, and DEFAULTPRINTINGHOST is (Tremor: Quake):

(OPENIMAGESTREAM) returns an Interpress image stream on printer Tremor:.

(OPENIMAGESTREAM NIL 'PRESS) returns a Press stream on Quake.

(OPENIMAGESTREAM ' {LPT} .INTERPRESS) returns an Interpress stream on Tremor:.

(OPENIMAGESTREAM ' {CORE} FOO .PRESS) returns a Press stream on the file {CORE} FOO .PRESS.

(**IMAGESTREAMP** *X* *IMAGETYPE*) [NoSpread Function]

Returns *X* (possibly coerced to a stream) if it is an output image stream of type *IMAGETYPE* (or of any type if *IMAGETYPE* = NIL), otherwise NIL.

(**IMAGESTREAMTYPE** *STREAM*) [Function]

Returns the image stream type of *STREAM*.

(**IMAGESTREAMTYPEP** *STREAM* *TYPE*) [Function]

Returns T if *STREAM* is an image stream of type *TYPE*.

Accessing Image Stream Fields

The following functions manipulate the fields of an image stream. These functions return the old value (the one being replaced). A value of NIL for the new value will return the current setting without changing it. These functions do not change any of the bits drawn on the image stream; they just affect future operations done on the image stream.

(**DSPCLIPPINGREGION** *REGION STREAM*) [Function]

The clipping region is a region that limits the extent of characters printed and lines drawn (in the image stream's coordinate system). Initially set so that no clipping occurs.

Warning: For display streams, the window system maintains the clipping region during window operations. Users should be very careful about changing this field.

(**DSPFONT** *FONT STREAM*) [Function]

The font field specifies the font used when printing characters to the image stream.

Note: DSPFONT determines its new font descriptor from *FONT* by the same coercion rules that FONTPROP and FONTCREATE use, with one additional possibility: If *FONT* is a list of the form (PROP VAL PROP VAL ...) where PROP is acceptable as a font-property to FONTCOPY, then the new font is obtained by (FONTCOPY (DSPFONT NIL STREAM) PROP VAL PROP VAL ...). For example, (DSPFONT ' (SIZE 12) STREAM) would change the font to the 12 point version of the current font, leaving all other font properties the same.

(**DSPTOPMARGIN** *YPOSITION STREAM*) [Function]

The top margin is an integer that is the Y position after a new page (in the image stream's coordinate system). This function has no effect on windows.

(**DSPBOTTOMMARGIN** *YPOSITION STREAM*) [Function]

The bottom margin is an integer that is the minimum Y position that characters will be printed by PRIN1 (in the image stream's coordinate system). This function has no effect on windows.

(**DSPLEFTMARGIN** *XPOSITION STREAM*) [Function]

The left margin is an integer that is the X position after an end-of-line (in the image stream's coordinate system). Initially the left edge of the clipping region.

(**DSPRIGHTMARGIN** *XPOSITION STREAM*) [Function]

The right margin is an integer that is the maximum X position that characters will be printed by PRIN1 (in the image stream's coordinate system). This is initially the position of the right edge of the window or page.

GRAPHICS OUTPUT OPERATIONS

The line length of a window or image stream (as returned by `LINELENGTH`) is computed by dividing the distance between the left and right margins by the width of an uppercase "A" in the current font. The line length is changed whenever the font, left margin, or right margin are changed or whenever the window is reshaped.

(**DSPOPERATION** *OPERATION STREAM*) [Function]

The operation is the default `BITBLT` operation used when printing or drawing on the image stream. One of `REPLACE`, `PAINT`, `INVERT`, or `ERASE`. Initially `REPLACE`. This is a meaningless operation for most printers which support the model that once dots are deposited on a page they cannot be removed.

(**DSPLINEFEED** *DELTAY STREAM*) [Function]

The linefeed is an integer that specifies the *Y* increment for each linefeed, normally negative. Initially minus the height of the initial font.

(**DSPCLEOL** *DSPSTREAM XPOS YPOS HEIGHT*) [Function]

"Clear to end of line". Clears a region from (*XPOS*, *YPOS*) to the right margin of the display, with a height of *HEIGHT*. If *XPOS* and *YPOS* are `NIL`, clears the remainder of the current display line, using the height of the current font.

(**DSPRUBOUTCHAR** *DSPSTREAM CHAR X Y TTBL*) [Function]

Backs up over character code *CHAR* in the *DSPSTREAM*, erasing it. If *X*, *Y* are supplied, the rubbing out starts from the position specified. **DSPRUBOUTCHAR** assumes *CHAR* was printed with the terminal table *TTBL*, so it knows to handle control characters, etc. *TTBL* defaults to the primary terminal table.

(**DSPSCALE** *SCALE STREAM*) [Function]

Returns the scale of the image stream *STREAM*, a number indicating how many units in the streams coordinate system correspond to one printer's point (1/72 of an inch). For example, `DSPSCALE` returns 1 for display streams, and 35.27778 for Interpress and Press streams (the number of microns per printer's point). In order to be device-independent, user graphics programs must either not specify position values absolutely, or must multiply absolute point quantities by the `DSPSCALE` of the destination stream. For example, to set the left margin of the Interpress stream *XX* to one inch, do

```
(DSPLEFTMARGIN (TIMES 72 (DSPSCALE NIL XX)) XX)
```

The `SCALE` argument to `DSPSCALE` is currently ignored. In a future release it will enable the scale of the stream to be changed under user control, so that the necessary multiplication will be done internal to the image stream interface. In this case, it would be possible to set the left margin of the Interpress stream *XX* to one inch by doing

```
(DSPSCALE 1 XX)
(DSPLEFTMARGIN 72 XX)
```

INTERLISP-D REFERENCE MANUAL

(**DSPSPACEFACTOR** *FACTOR STREAM*) [Function]

The space factor is the amount by which to multiply the natural width of all following space characters on *STREAM*; this can be used for the justification of text. The default value is 1. For example, if the natural width of a space in *STREAM*'s current font is 12 units, and the space factor is set to two, spaces appear 24 units wide. The values returned by **STRINGWIDTH** and **CHARWIDTH** are also affected.

The following two functions only have meaning for image streams that can display color:

(**DSPCOLOR** *COLOR STREAM*) [Function]

Sets the default foreground color of *STREAM*. Returns the previous foreground color. If *COLOR* is **NIL**, it returns the current foreground color without changing anything. The default color is white

(**DSPBACKCOLOR** *COLOR STREAM*) [Function]

Sets the background color of *STREAM*. Returns the previous background color. If *COLOR* is **NIL**, it returns the current background color without changing anything. The default background color is black.

Current Position of an Image Stream

Each image stream has a "current position," which is a position (in the image stream's coordinate system) where the next printing operation will start from. The functions which print characters or draw on an image stream update these values appropriately. The following functions are used to explicitly access the current position of an image stream:

(**DSPXPOSITION** *XPOSITION STREAM*) [Function]

Returns the X coordinate of the current position of *STREAM*. If *XPOSITION* is non-**NIL**, the X coordinate is set to it (without changing the Y coordinate).

(**DSPYPOSITION** *YPOSITION STREAM*) [Function]

Returns the Y coordinate of the current position of *STREAM*. If *YPOSITION* is non-**NIL**, the Y coordinate is set to it (without changing the X coordinate).

(**MOVETO** *X Y STREAM*) [Function]

Changes the current position of *STREAM* to the point (*X*, *Y*).

(**RELMOVETO** *DX DY STREAM*) [Function]

Changes the current position to the point (*DX*, *DY*) coordinates away from current position of *STREAM*.

GRAPHICS OUTPUT OPERATIONS

(**MOVETOUPPERLEFT** *STREAM REGION*)

[Function]

Moves the current position to the beginning position of the top line of text. If *REGION* is non-NIL, it must be a *REGION* and the X position is changed to the left edge of *REGION* and the Y position changed to the top of *REGION* less the font ascent of *STREAM*. If *REGION* is NIL, the X coordinate is changed to the left margin of *STREAM* and the Y coordinate is changed to the top of the clipping region of *STREAM* less the font ascent of *STREAM*.

Moving Bits Between Bitmaps With BITBLT

BITBLT is the primitive function for moving bits from one bitmap to another, or from a bitmap to an image stream.

(**BITBLT** *SOURCE SOURCELEFT SOURCEBOTTOM DESTINATION DESTINATIONLEFT*
DESTINATIONBOTTOM WIDTH HEIGHT SOURCTYPE OPERATION TEXTURE
CLIPPINGREGION) [Function]

Transfers a rectangular array of bits from *SOURCE* to *DESTINATION*. *SOURCE* can be a bitmap, or a display stream or window, in which case its associated bitmap is used. *DESTINATION* can be a bitmap or an arbitrary image stream.

WIDTH and *HEIGHT* define a pair of rectangles, one in each of the *SOURCE* and *DESTINATION* whose left, bottom corners are at, respectively, (*SOURCELEFT*, *SOURCEBOTTOM*) and (*DESTINATIONLEFT*, *DESTINATIONBOTTOM*). If these rectangles overlap the boundaries of either source or destination they are both reduced in size (without translation) so that they fit within their respective boundaries. If *CLIPPINGREGION* is non-NIL it should be a *REGION* and is interpreted as a clipping region within *DESTINATION*; clipping to this region may further reduce the defining rectangles. These (possibly reduced) rectangles define the source and destination rectangles for BITBLT.

The mode of *transferring* bits is defined by *SOURCTYPE* and *OPERATION*. *SOURCTYPE* and *OPERATION* specify whether the source bits should come from *SOURCE* or *TEXTURE*, and how these bits are combined with those of *DESTINATION*. *SOURCTYPE* and *OPERATION* are described further below.

TEXTURE is a texture. BITBLT aligns the texture so that the upper-left pixel of the texture coincides with the upper-left pixel of the destination bitmap.

SOURCELEFT, *SOURCEBOTTOM*, *DESTINATIONLEFT*, and *DESTINATIONBOTTOM* default to 0. *WIDTH* and *HEIGHT* default to the width and height of the *SOURCE*. *TEXTURE* defaults to white. *SOURCTYPE* defaults to INPUT. *OPERATION* defaults to REPLACE. If *CLIPPINGREGION* is not provided, no additional clipping is done. BITBLT returns T if any bits were moved; NIL otherwise.

Note: If *SOURCE* or *DESTINATION* is a window or image stream, the remaining arguments are interpreted as values in the coordinate system of the window or image

INTERLISP-D REFERENCE MANUAL

stream and the operation of BITBLT is translated and clipped accordingly. Also, if a window or image stream is used as the destination to BITBLT, its clipping region further limits the region involved.

SOURCETYPE specifies whether the source bits should come from the bitmap *SOURCE*, or from the texture *TEXTURE*. *SOURCETYPE* is interpreted as follows:

- INPUT The source bits come from *SOURCE*. *TEXTURE* is ignored.
- INVERT The source bits are the inverse of the bits from *SOURCE*. *TEXTURE* is ignored.
- TEXTURE The source bits come from *TEXTURE*. *SOURCE*, *SOURCELEFT*, and *SOURCEBOTTOM* are ignored.

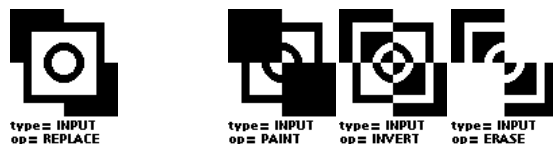
OPERATION specifies how the source bits (as specified by *SOURCETYPE*) are combined with the bits in *DESTINATION* and stored back into *DESTINATION*. *DESTINATION* is one of the following:

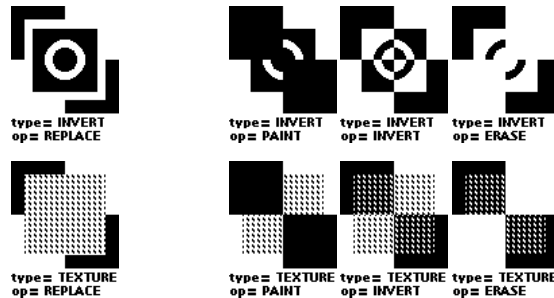
- REPLACE All source bits (on or off) replace destination bits.
- PAINT Any source bits that are on replace the corresponding destination bits. Source bits that are off have no effect. Does a logical OR between the source bits and the destination bits.
- INVERT Any source bits that are on invert the corresponding destination bits. Does a logical XOR between the source bits and the destination bits.
- ERASE Any source bits that are on erase the corresponding destination bits. Does a logical AND operation between the inverse of the source bits and the destination bits.

Different combinations of *SOURCETYPE* and *OPERATION* can be specified to achieve many different effects. Given the following bitmaps as the values of *SOURCE*, *TEXTURE*, and *DESTINATION*:



BITBLT would produce the results given below for the difference combinations of *SOURCETYPE* and *OPERATION* (assuming *CLIPPINGREGION*, *SOURCELEFT*, etc. are set correctly, of course):





(**BLTSHADE** TEXTURE DESTINATION DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT OPERATION CLIPPINGREGION) [Function]

BLTSHADE is the *SOURCETYPE* = TEXTURE case of BITBLT. It fills the specified region of the destination bitmap *DESTINATION* with the texture *TEXTURE*. *DESTINATION* can be a bitmap or image stream.

(**BITMAPIMAGESIZE** BITMAP DIMENSION STREAM) [Function]

Returns the size that *BITMAP* will be when BITBLT'ed to *STREAM*, in *STREAM*'s units. *DIMENSION* can be one of *WIDTH*, *HEIGHT*, or *NIL*, in which case the dotted pair (*WIDTH* . *HEIGHT*) will be returned.

Drawing Lines

Interlisp-D provides several functions for drawing lines and curves on image streams. The line drawing functions are intended for interactive applications where efficiency is important. They do not allow the use of "brush" patterns, like the curve drawing functions, but (for display streams) they support drawing a line in INVERT mode, so redrawing the line will erase it. DRAWCURVE can be used to draw lines using a brush.

(**DRAWLINE** X Y X Y WIDTH OPERATION STREAM COLOR DASHING) [Function]

Draws a straight line from the point (*X* , *Y*) to the point (*X* , *Y*) on the image stream *STREAM*. The position of *STREAM* is set to (*X* , *Y*). If *X* equals *X* and *Y* equals *Y* , a point is drawn at (*X* , *Y*).

WIDTH is the width of the line, in the units of the device. If *WIDTH* is *NIL*, the default is 1.

OPERATION is the BITBLT operation used to draw the line. If *OPERATION* is *NIL*, the value of *DSPOPERATION* for the image stream is used.

COLOR is a color specification that determines the color used to draw the line for image streams that support color. If *COLOR* is *NIL*, the *DSPCOLOR* of *STREAM* is used.

DASHING is a list of positive integers that determines the dashing characteristics of the line. The line is drawn for the number of points indicated by the first element of the

INTERLISP-D REFERENCE MANUAL

dashing list, is not drawn for the number of points indicated by the second element. The third element indicates how long it will be on again, and so forth. The dashing sequence is repeated from the beginning when the list is exhausted. A brush LINEWITHBRUSH-by-LINEWITHBRUSH is used.

If *DASHING* is NIL, the line is not dashed.

(**DRAWBETWEEN** *POSITION POSITION WIDTH OPERATION STREAM COLOR DASHING*) [Function]

Draws a line from the point *POSITION* to the point *POSITION* onto the destination bitmap of *STREAM*. The position of *STREAM* is set to *POSITION*.

In the Medley release, when using the color argument, Interpress **DRAWLINE** treats 16x16 bitmaps or negative numbers as shades/textures. Positive numbers continue to refer to color maps, and so cannot be used as textures. To convert an integer shade into a negative number use NEGSHADE (e.g. (NEGSHADE 42495) is -23041).

(**DRAWTO** *X Y WIDTH OPERATION STREAM COLOR DASHING*) [Function]

Draws a line from the current position to the point (*X*, *Y*) onto the destination bitmap of *STREAM*. The position of *STREAM* is set to (*X*, *Y*).

(**RELDRAWTO** *DX DY WIDTH OPERATION STREAM COLOR DASHING*) [Function]

Draws a line from the current position to the point (*DX*, *DY*) coordinates away onto the destination bitmap of *STREAM*. The position of *STREAM* is set to the end of the line. If *DX* and *DY* are both 0, nothing is drawn.

Drawing Curves

A curve is drawn by placing a brush pattern centered at each point along the curve's trajectory. A brush pattern is defined by its shape, size, and color. The predefined brush shapes are ROUND, SQUARE, HORIZONTAL, VERTICAL, and DIAGONAL; new brush shapes can be created using the INSTALLBRUSH function, described below. A brush size is an integer specifying the width of the brush in the units of the device. The color is a color specification, which is only used if the curve is drawn to an image stream that supports colors.

A brush is specified to the various drawing functions as a list of the form (SHAPE WIDTH COLOR), for example (SQUARE 2) or (VERTICAL 4 RED). A brush can also be specified as a positive integer, which is interpreted as a ROUND brush of that width. If a brush is a litatom, it is assumed to be a function which is called at each point of the curve's trajectory (with three arguments: the X-coordinate of the point, the Y-coordinate, and the image stream), and should do whatever image stream operations are necessary to draw each point. Finally, if a brush is specified as NIL, a (ROUND 1) brush is used as default.

GRAPHICS OUTPUT OPERATIONS

The appearance of a curve is also determined by its dashing characteristics. Dashing is specified by a list of positive integers. If a curve is dashed, the brush is placed along the trajectory for the number of units indicated by the first element of the dashing list. The brush is off, not placed in the bitmap, for a number of units indicated by the second element. The third element indicates how long it will be on again, and so forth. The dashing sequence is repeated from the beginning when the list is exhausted. The units used to measure dashing are the units of the brush. For example, specifying the dashing as (1 1) with a brush of (ROUND 16) would put the brush on the trajectory, skip 16 points, and put down another brush. A curve is not dashed if the dashing argument to the drawing function is NIL.

The curve functions use the image stream's clipping region and operation. Most types of image streams only support the PAINT operation when drawing curves. When drawing to a display stream, the curve-drawing functions accept the operation INVERT if the brush argument is 1. For brushes larger than 1, these functions will use the ERASE operation instead of INVERT. For display streams, the curve-drawing functions treat the REPLACE operation the same as PAINT.

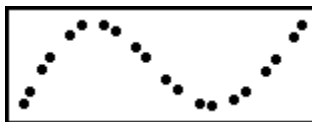
(**DRAWCURVE** *KNOTS CLOSED BRUSH DASHING STREAM*) [Function]

Draws a "parametric cubic spline curve" on the image stream *STREAM*. *KNOTS* is a list of positions to which the curve will be fitted. If *CLOSED* is non-NIL, the curve will be closed; otherwise it ends at the first and last positions in *KNOTS*. *BRUSH* and *DASHING* are interpreted as described above.

For example,

```
(DRAWCURVE ' ((10 . 10) (50 . 50) (100 . 10) (150 . 50))
  NIL ' (ROUND 5) ' (1 1 1 2) XX)
```

would draw a curve like the following on the display stream XX:



(**DRAWCIRCLE** *CENTERX CENTERY RADIUS BRUSH DASHING STREAM*) [Function]

Draws a circle of radius *RADIUS* about the point (*CENTERX*, *CENTERY*) onto the image stream *STREAM*. *STREAM*'s position is left at (*CENTERX*, *CENTERY*). The other arguments are interpreted as described above.

(**DRAWARC** *CENTERX CENTERY RADIUS STARTANGLE NDEGREES BRUSH DASHINGSTREAM*) [Function]

Draws an arc of the circle whose center point is (*CENTERX* *CENTERY*) and whose radius is *RADIUS* from the position at *STARTANGLE* degrees for *NDEGREES* number of degrees. If *STARTANGLE* is 0, the starting point will be (*CENTERX* (*CENTERY* + *RADIUS*)). If *NDEGREES* is positive, the arc will be counterclockwise. If *NDEGREES* is negative, the arc will be clockwise. The other arguments are interpreted as described in **DRAWCIRCLE**.

INTERLISP-D REFERENCE MANUAL

(**DRAWELLIPSE** *CENTERX* *CENTERY* *SEMIMINORRADIUS* *SEMIMAJORRADIUS*
ORIENTATION *BRUSH* *DASHING* *STREAM*) [Function]

Draws an ellipse with a minor radius of *SEMIMINORRADIUS* and a major radius of *SEMIMAJORRADIUS* about the point (*CENTERX*, *CENTERY*) onto the image stream *STREAM*. *ORIENTATION* is the angle of the major axis in degrees, positive in the counterclockwise direction. *STREAM*'s position is left at (*CENTERX*, *CENTERY*). The other arguments are interpreted as described above.

New brush shapes can be defined using the following function:

(**INSTALLBRUSH** *BRUSHNAME* *BRUSHFN* *BRUSHARRAY*) [Function]

Installs a new brush called *BRUSHNAME* with creation-function *BRUSHFN* and optional array *BRUSHARRAY*. *BRUSHFN* should be a function of one argument (a width), which returns a bitmap of the brush for that width. *BRUSHFN* will be called to create new instances of *BRUSHNAME*-type brushes; the sixteen smallest instances will be pre-computed and cached. "Hand-crafted" brushes can be supplied as the *BRUSHARRAY* argument. Changing an existing brush can be done by calling **INSTALLBRUSH** with new *BRUSHFN* and/or *BRUSHARRAY*.

(**DRAWPOINT** *X* *Y* *BRUSH* *STREAM* *OPERATION*) [Function]

Draws *BRUSH* centered around point (*X*, *Y*) on *STREAM*, using the operation *OPERATION*. *BRUSH* may be a bitmap or a brush.

Miscellaneous Drawing and Printing Operations

(**DSPFILL** *REGION* *TEXTURE* *OPERATION* *STREAM*) [Function]

Fills *REGION* of the image stream *STREAM* (within the clipping region) with the texture *TEXTURE*. If *REGION* is NIL, the whole clipping region of *STREAM* is used. If *TEXTURE* or *OPERATION* is NIL, the values for *STREAM* are used.

(**DRAWPOLYGON** *POINTS* *CLOSED* *BRUSH* *DASHING* *STREAM*) [Function]

Draws a polygon on the image stream *STREAM*. *POINTS* is a list of positions to which the figure will be fitted (the vertices of the polygon). If *CLOSED* is non-NIL, then the starting position is specified only once in *POINTS*. If *CLOSED* is NIL, then the starting vertex must be specified twice in *POINTS*. *BRUSH* and *DASHING* are interpreted as described in Chapter 27 of the Interlisp-D Reference Manual.

For example,

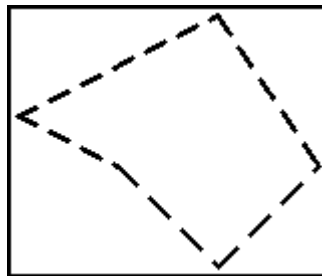
```
(DRAWPOLYGON ' ((100 . 100) (50 . 125)
                 (150 . 175) (200 . 100) (150 .
```

GRAPHICS OUTPUT OPERATIONS

50))

T ' (ROUND 3) ' (4 2) XX)

will draw a polygon like the following on the display stream XX.



(**FILLPOLYGON** *POINTS TEXTURE OPERATION WINDNUMBER STREAM*) [Function]

OPERATION is the **BITBLT** operation (see page 27.15 in the Interlisp-D Reference Manual) used to fill the polygon. If the *OPERATION* is NIL, the *OPERATION* defaults to the *STREAM* default *OPERATION*.

WINDNUMBER is the number for the winding rule convention. This number is either 0 or 1; 0 indicates the "zero" winding rule, 1 indicates the "odd" winding rule.

When filling a polygon, there is more than one way of dealing with the situation where two polygon sides intersect, or one polygon is fully inside the other. Currently, **FILLPOLYGON** to a display stream uses the "odd" winding rule, which means that intersecting polygon sides define areas that are filled or not filled somewhat like a checkerboard. For example,

```
(FILLPOLYGON
  ' ( ((110 . 110) (150 . 200) (190 . 110))
      ((135 . 125) (160 . 125) (160 . 150) (135 .
150)) )
  GRAYSHADE WINDOW)
```

will produce a display something like this:



This fill convention also takes into account all polygons in *POINTS*, if it specifies multiple polygons.

(**FILLCIRCLE** *CENTERX CENTERY RADIUS TEXTURE STREAM*) [Function]

Fills in a circular area of radius *RADIUS* about the point (*CENTERX*, *CENTERY*) in *STREAM* with *TEXTURE*. *STREAM*'s position is left at (*CENTERX*, *CENTERY*).

INTERLISP-D REFERENCE MANUAL

(**DSPRESET** *STREAM*) [Function]

Sets the X coordinate of *STREAM* to its left margin, sets its Y coordinate to the top of the clipping region minus the font ascent. For a display stream, this also fills its destination bitmap with its background texture.

(**DSPNEWPAGE** *STREAM*) [Function]

Starts a new page. The X coordinate is set to the left margin, and the Y coordinate is set to the top margin plus the linefeed.

(**CENTERPRINTINREGION** *EXP REGION STREAM*) [Function]

Prints *EXP* so that it is centered within *REGION* of the *STREAM*. If *REGION* is NIL, *EXP* will be centered in the clipping region of *STREAM*.

Drawing and Shading Grids

A grid is a partitioning of an arbitrary coordinate system (hereafter referred to as the "source system") into rectangles. This section describes functions that operate on grids. It includes functions to draw the outline of a grid, to translate between positions in a source system and grid coordinates (the coordinates of the rectangle which contains a given position), and to shade grid rectangles. A grid is defined by its "unit grid," a region (called a grid specification) which is the origin rectangle of the grid in terms of the source system. Its LEFT field is interpreted as the X-coordinate of the left edge of the origin rectangle, its BOTTOM field is the Y-coordinate of the bottom edge of the origin rectangle, its WIDTH is the width of the grid rectangles, and its HEIGHT is the height of the grid rectangles.

(**GRID** *GRIDSPEC WIDTH HEIGHT BORDER STREAM GRIDSHADE*) [Function]

Outlines the grid defined by *GRIDSPEC* which is *WIDTH* rectangles wide and *HEIGHT* rectangles high on *STREAM*. Each box in the grid has a border within it that is *BORDER* points on each side; so the resulting lines in the grid are 2**BORDER* thick. If *BORDER* is the atom *POINT*, instead of a border the lower left point of each grid rectangle will be turned on. If *GRIDSHADE* is non-NIL, it should be a texture and the border lines will be drawn using that texture.

(**SHADEGRIDBOX** *X Y SHADE OPERATION GRIDSPEC GRIDBORDER STREAM*) [Function]

Shades the grid rectangle (*X*, *Y*) of *GRIDSPEC* with texture *SHADE* using *OPERATION* on *STREAM*. *GRIDBORDER* is interpreted the same as for GRID.

GRAPHICS OUTPUT OPERATIONS

The following two functions map from the *X, Y* coordinates of the source system into the grid *X, Y* coordinates:

(**GRIDXCOORD** *XCOORD* *GRIDSPEC*) [Function]

Returns the grid *X*-coordinate (in the grid specified by *GRIDSPEC*) that contains the source system *X*-coordinate *XCOORD*.

(**GRIDYCOORD** *YCOORD* *GRIDSPEC*) [Function]

Returns the grid *Y*-coordinate (in the grid specified by *GRIDSPEC*) that contains the source system *Y*-coordinate *YCOORD*.

The following two functions map from the grid *X, Y* coordinates into the *X, Y* coordinates of the source system:

(**LEFTOFGRIDCOORD** *GRIDX* *GRIDSPEC*) [Function]

Returns the source system *X*-coordinate of the left edge of a grid rectangle at grid *X*-coordinate *GRIDX* (in the grid specified by *GRIDSPEC*).

(**BOTTOMOFGRIDCOORD** *GRIDY* *GRIDSPEC*) [Function]

Returns the source system *Y*-coordinate of the bottom edge of a grid rectangle at grid *Y*-coordinate *GRIDY* (in the grid specified by *GRIDSPEC*).

Display Streams

Display streams (image streams of type *DISPLAY*) are used to control graphic output operations to a bitmap, known as the "destination" bitmap of the display stream. For each window on the screen, there is an associated display stream which controls graphics operations to a specific part of the screen bitmap. Any of the functions that take a display stream will also take a window, and use the associated display stream. Display streams can also have a destination bitmap that is not connected to any window or display device.

(**DSPCREATE** *DESTINATION*) [Function]

Creates and returns a display stream. If *DESTINATION* is specified, it is used as the destination bitmap, otherwise the screen bitmap is used.

(**DSPDESTINATION** *DESTINATION* *DISPLAYSTREAM*) [Function]

Returns the current destination bitmap for *DISPLAYSTREAM*, setting it to *DESTINATION* if non-NIL. *DESTINATION* can be either the screen bitmap, or an auxilliary bitmap in order to construct figures, possibly save them, and then display them in a single operation.

Warning: The window system maintains the destination of a window's display stream. Users should be very careful about changing this field.

INTERLISP-D REFERENCE MANUAL

(**DSPXOFFSET** *XOFFSET DISPLAYSTREAM*) [Function]

(**DSPYOFFSET** *YOFFSET DISPLAYSTREAM*) [Function]

Each display stream has its own coordinate system, separate from the coordinate system of its destination bitmap. Having the coordinate system local to the display stream allows objects to be displayed at different places by translating the display stream's coordinate system relative to its destination bitmap. This local coordinate system is defined by the X offset and Y offset.

DSPXOFFSET returns the current X offset for *DISPLAYSTREAM*, the X origin of the display stream's coordinate system in the destination bitmap's coordinate system. It is set to *XOFFSET* if non-NIL.

DSPYOFFSET returns the current Y offset for *DISPLAYSTREAM*, the Y origin of the display stream's coordinate system in the destination bitmap's coordinate system. It is set to *YOFFSET* if non-NIL.

The X offset and Y offset for a display stream are both initially 0 (no X or Y-coordinate translation).

Warning: The window system maintains the X and Y offset of a window's display stream. Users should be very careful about changing these fields.

(**DSPTEXTURE** *TEXTURE DISPLAYSTREAM*) [Function]

Returns the current texture used as the background pattern for *DISPLAYSTREAM*. It is set to *TEXTURE* if non-NIL. Initially the value of WHITESHADE.

(**DSPSOURCETYPE** *SOURCETYPE DISPLAYSTREAM*) [Function]

Returns the current BITBLT sourcetype used when printing characters to the display stream. It is set to *SOURCETYPE*, if non-NIL. Must be either INPUT or INVERT. Initially INPUT.

(**DSPSCROLL** *SWITCHSETTING DISPLAYSTREAM*) [Function]

Returns the current value of the "scroll flag," a flag that determines the scrolling behavior of the display stream; either ON or OFF. If ON, the bits in the display streams's destination bitmap are moved after any linefeed that moves the current position out of the destination bitmap. Any bits moved out of the current clipping region are lost. Does not adjust the X offset, Y offset, or clipping region of the display stream. Initially OFF.

Sets the scroll flag to *SWITCHSETTING*, if non-NIL.

Note: The word "scrolling" also describes the use of "scroll bars" on the left and bottom of a window to move an object displayed in a window.

GRAPHICS OUTPUT OPERATIONS

Each window has an associated display stream. To get the window of a particular display stream, use WFROMDS:

(WFROMDS *DISPLAYSTREAM* DONTCREATE) [Function]

Returns the window associated with *DISPLAYSTREAM*, creating a window if one does not exist (and *DONTCREATE* is NIL). Returns NIL if the destination of *DISPLAYSTREAM* is not a screen bitmap that supports a window system.

If *DONTCREATE* is non-NIL, WFROMDS will never create a window, and returns NIL if *DISPLAYSTREAM* does not have an associated window.

TTYDISPLAYSTREAM calls WFROMDS with DONTCREATE = T, so it will not create a window unnecessarily. Also, if WFROMDS does create a window, it calls CREATEW with NOOPENFLG = T.

(DSPBACKUP *WIDTH* *DISPLAYSTREAM*) [Function]

Backs up *DISPLAYSTREAM* over a character which is *WIDTH* screen points wide. DSPBACKUP fills the backed over area with the display stream's background texture and decreases the X position by *WIDTH*. If this would put the X position less than *DISPLAYSTREAM*'s left margin, its operation is stopped at the left margin. It returns T if any bits were written, NIL otherwise.

Fonts

A font is the collection of images that are printed or displayed when characters are output to a graphic output device. Some simple displays and printers can only print characters using one font. Bitmap displays and graphic printers can print characters using a large number of fonts.

Fonts are identified by a distinctive style or family (such as Modern or Classic), a size (such as 10 points), and a face (such as bold or italic). Fonts also have a rotation that indicates the orientation of characters on the screen or page. A normal horizontal font (also called a portrait font) has a rotation of 0; the rotation of a vertical (landscape) font is 90 degrees. While any combination can be specified, in practice the user will find that only certain combinations of families, sizes, faces, and rotations are available for any graphic output device.

To specify a font to the functions described below, a FAMILY is represented by a literal atom, a SIZE by a positive integer, and a FACE by a three-element list of the form (WEIGHT SLOPE EXPANSION). WEIGHT, which indicates the thickness of the characters, can be BOLD, MEDIUM, or LIGHT; SLOPE can be ITALIC or REGULAR; and EXPANSION can be REGULAR, COMPRESSED, or EXPANDED, indicating how spread out the characters are. For convenience, faces may also be specified by three-character atoms, where each character is the first letter of the corresponding field. Thus, MRR is a synonym for (MEDIUM REGULAR REGULAR). In addition, certain common face combinations may be indicated by special literal atoms:

INTERLISP-D REFERENCE MANUAL

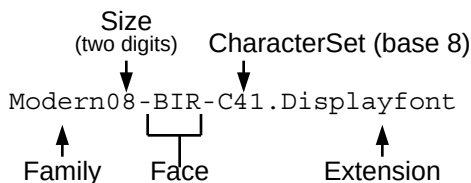
```
STANDARD = (MEDIUM REGULAR REGULAR) = MRR
ITALIC = (MEDIUM ITALIC REGULAR) = MIR
BOLD = (BOLD REGULAR REGULAR) = BRR
BOLDITALIC = (BOLD ITALIC REGULAR) = BIR
```

Interlisp represents all the information related to a font in an object called a font descriptor. Font descriptors contain the family, size, etc. properties used to represent the font. In addition, for each character in the font, the font descriptor contains width information for the character and (for display fonts) a bitmap containing the picture of the character.

The font functions can take fonts specified in a variety of different ways. DSPFONT, FONTCREATE, FONTCOPY, etc. can be applied to font descriptors, "font lists" such as '(MODERN 10), image streams (coerced to its current font), or windows (coerced to the current font of its display stream). The printout command ".FONT" will also accept fonts specified in any of these forms.

In general font files use the following format:

The family name (e.g., Modern); a two digit size (e.g., 08); a three letter Face (e.g., BIR, for Bold Italic Regular); the letter C followed by the font's character set in base 8 (e.g., C41); and finally an extension (e.g., Displayfont).



(**FONTCREATE** FAMILY SIZE FACE ROTATION DEVICE NOERRORFLG CHARSET) [Function
]

Returns a font descriptor for the specified font. *FAMILY* is a litatom specifying the font family. *SIZE* is an integer indicating the size of the font in points. *FACE* specifies the face characteristics in one of the formats listed above; if *FACE* is NIL, STANDARD is used. *ROTATION*, which specifies the orientation of the font, is 0 (or NIL) for a portrait font and 90 for a landscape font. *DEVICE* indicates the output device for the font, and can be any image stream type, such as DISPLAY, INTERPRESS, etc. *DEVICE* may also be an image stream, in which case the type of the stream determines the font device. *DEVICE* defaults to DISPLAY.

GRAPHICS OUTPUT OPERATIONS

The *FAMILY* argument to **FONTCREATE** may also be a list, in which case it is interpreted as a font-specification quintuple, a list of the form (*FAMILY SIZE FACE ROTATION DEVICE*). Thus, (**FONTCREATE** ' (GACHA 10 BOLD)) is equivalent to (**FONTCREATE** ' GACHA 10 ' BOLD). *FAMILY* may also be a font descriptor, in which case that descriptor is simply returned.

If a font descriptor has already been created for the specified font, **FONTCREATE** simply returns it. If it has not been created, **FONTCREATE** has to read the font information from a font file that contains the information for that font. The name of an appropriate font file, and the algorithm for searching depends on the device that the font is for, and is described in more detail below. If an appropriate font file is found, it is read into a font descriptor. If no file is found, for **DISPLAY** fonts **FONTCREATE** looks for fonts with less face information and fakes the remaining faces (such as by doubling the bit pattern of each character or slanting it). For hardcopy printer fonts, there is no acceptable faking algorithm.

If no acceptable font is found, the action of **FONTCREATE** is determined by **NOERRORFLG**. If **NOERRORFLG** is **NIL**, it generates a **FONT NOT FOUND** error with the offending font specification; otherwise, **FONTCREATE** returns **NIL**.

CHARSET is the character set which will be read to create the font. Defaults to 0. For more information on character sets, see **NS Characters**.

(**FONTP** *X*) [Function]

Returns *X* if *X* is a font descriptor; **NIL** otherwise.

(**FONTPROP** *FONT PROP*) [Function]

Returns the value of the *PROP* property of font *FONT*. The following font properties are recognized:

FAMILY The style of the font, represented as a literal atom, such as **CLASSIC** or **MODERN**.

SIZE A positive integer giving the size of the font, in printer's points (1/72 of an inch).

WEIGHT The thickness of the characters; one of **BOLD**, **MEDIUM**, or **LIGHT**.

SLOPE The "slope" of the characters in the font; one of **ITALIC** or **REGULAR**.

EXPANSION The extent to which the characters in the font are spread out; one of **REGULAR**, **COMPRESSED**, or **EXPANDED**. Most available fonts have **EXPANSION = REGULAR**.

FACE A three-element list of the form (**WEIGHT SLOPE EXPANSION**), giving all of the typeface parameters.

INTERLISP-D REFERENCE MANUAL

ROTATION	An integer that gives the orientation of the font characters on the screen or page, in degrees. A normal horizontal font (also called a portrait font) has a rotation of 0; the rotation of a vertical (landscape) font is 90.
DEVICE	The device that the font can be printed on; one of DISPLAY, INTERPRESS, etc.
ASCENT	An integer giving the maximum height of any character in the font from its base line (the printing position). The top line will be at BASELINE+ASCENT-1.
DESCENT	An integer giving the maximum extent of any character below the base line, such as the lower part of a "p". The bottom line of a character will be at BASELINE-DESCENT.
HEIGHT	Equal to ASCENT + DESCENT.
SPEC	The (FAMILY SIZE FACE ROTATION DEVICE) quintuple by which the font is known to Lisp.
DEVICESPEC	The (FAMILY SIZE FACE ROTATION DEVICE) quintuple that identifies what will be used to represent the font on the display or printer. It will differ from the SPEC property only if an implicit coercion is done to approximate the specified font with one that actually exists on the device.
SCALE	The units per printer's point (1/72 of an inch) in which the font is measured. For example, this is 35.27778 (the number of microns per printer's point) for Interpress fonts, which are measured in terms of microns.

(**FontCOPY** *OLDFont PROP VAL PROP VAL ...*) [NoSpread Function]

Returns a font descriptor that is a copy of the font *OLDFont*, but which differs from *OLDFont* in that *OLDFont*'s properties are replaced by the specified properties and values. Thus, (FontCOPY FONT 'WEIGHT 'BOLD 'DEVICE 'INTERPRESS) will return a bold Interpress font with all other properties the same as those of FONT. FontCOPY accepts the properties FAMILY, SIZE, WEIGHT, SLOPE, EXPANSION, FACE, ROTATION, and DEVICE. If the first property is a list, it is taken to be the *PROP VAL PROP VAL ...* sequence. Thus, (FontCOPY FONT ' (WEIGHT BOLD DEVICE INTERPRESS)) is equivalent to the example above.

If the property NOERROR is specified with value non-NIL, FontCOPY will return NIL rather than causing an error if the specified font cannot be created.

(**FontSAVAILABLE** *FAMILY SIZE FACE ROTATION DEVICE CHECKFILESTOO?*) [Function]

Returns a list of available fonts that match the given specification. *FAMILY*, *SIZE*, *FACE*, *ROTATION*, and *DEVICE* are the same as for FontCREATE. Additionally, any of them can be the atom *, in which case all values of that field are matched.

GRAPHICS OUTPUT OPERATIONS

If *CHECKFILESTOO?* is NIL, only fonts already loaded into virtual memory will be considered. If *CHECKFILESTOO?* is non-NIL, the font directories for the specified device will be searched. When checking font files, the *ROTATION* is ignored.

Note: The search is conditional on the status of the server which holds the font. Thus a file server crash may prevent FONTCREATE from finding a file that an earlier FONTSAVAILABLE returned.

Each element of the list returned will be of the form (*FAMILY SIZE FACE ROTATION DEVICE*).

Examples:

```
(FONTSAVAILABLE 'MODERN 10 'MRR 0 'DISPLAY)
```

will return ((MODERN 10 (MEDIUM REGULAR REGULAR) 0 DISPLAY)) if the regular Modern 10 font for the display is in virtual memory; NIL otherwise.

```
(FONTSAVAILABLE '* 14 '* '* 'INTERPRESS T)
```

will return a list of all the size 14 Interpress fonts, whether they are in virtual memory or in font files.

(**SETFONTDESCRIPTOR** *FAMILY SIZE FACE ROTATION DEVICE FONT*) [Function]

Indicates to the system that *FONT* is the font that should be associated with the *FAMILY SIZE FACE ROTATION DEVICE* characteristics. If *FONT* is NIL, the font associated with these characteristics is cleared and will be recreated the next time it is needed. As with FONTPROP and FONTCOPY, *FONT* is coerced to a font descriptor if it is not one already.

This functions is useful when it is desirable to simulate an unavailable font or to use a font with characteristics different from the interpretations provided by the system.

(**DEFAULTFONT** *DEVICE FONT* -) [Function]

Returns the font that would be used as the default (if NIL were specified as a font argument) for image stream type *DEVICE*. If *FONT* is a font descriptor, it is set to be the default font for *DEVICE*.

(**CHARWIDTH** *CHARCODE FONT*) [Function]

CHARCODE is an integer that represents a valid character (as returned by CHCON1). Returns the amount by which an image stream's **X**-position will be incremented when the character is printed.

(**CHARWIDTHY** *CHARCODE FONT*) [Function]

Like CHARWIDTH, but returns the **Y** component of the character's width, the amount by which an image stream's **Y**-position will be incremented when the character is printed. This will be zero for most characters in normal portrait fonts, but may be non-zero for landscape fonts or for vector-drawing fonts.

INTERLISP-D REFERENCE MANUAL

(STRINGWIDTH STR FONT FLG RDTBL) [Function]

Returns the amount by which a stream's **X**-position will be incremented if the printname for the Interlisp-D object *STR* is printed in font *FONT*. If *FONT* is NIL, DEFAULTFONT is used as *FONT*. If *FONT* is an image stream, its font is used. If *FLG* is non-NIL, the PRIN2-pname of *STR* with respect to the readtable *RDTBL* is used.

(STRINGREGION STR STREAM PRIN2FLG RDTBL) [Function]

Returns the region occupied by *STR* if it were printed at the current location in the image stream *STREAM*. This is useful, for example, for determining where text is in a window to allow the user to select it. The arguments *PRIN2FLG* and *RDTBL* are passed to STRINGWIDTH.

Note: STRINGREGION does not take into account any carriage returns in the string, or carriage returns that may be automatically printed if *STR* is printed to *STREAM*. Therefore, the value returned is meaningless for multi-line strings.

The following functions allow the user to access and change the bitmaps for individual characters in a display font. Note: Character code 256 can be used to access the "dummy" character, used for characters in the font with no bitmap defined.

(GETCHARBITMAP CHARCODE FONT) [Function]

Returns a bitmap containing a copy of the image of the character *CHARCODE* in the font *FONT*.

(PUTCHARBITMAP CHARCODE FONT NEWCHARBITMAP NEWCHARDESCENT) [Function]

Changes the bitmap image of the character *CHARCODE* in the font *FONT* to the bitmap *NEWCHARBITMAP*. If *NEWCHARDESCENT* is non-NIL, the descent of the character is changed to the value of *NEWCHARDESCENT*.

(EDITCHAR CHARCODE FONT) [Function]

Calls the bitmap editor (EDITBM) on the bitmap image of the character *CHARCODE* in the font *FONT*. *CHARCODE* can be a character code (as returned by CHCON1) or an atom or string, in which case the first character of *CHARCODE* is used.

(WRITESTRIKEFONTFILE FONT CHARSET FILENAME) [Function]

Takes a display font font descriptor and a character set number, and writes that character set into a file suitable for reading in again. Note that the font descriptor's current state is used (which was perhaps modified by INSPECTing the datum), so this provides a mechanism for creating/modifying new fonts.

For example:


```
(WRITESTRIKEFONTFILE (FONTCREATE 'GACHA 10) 0 '{DSK}Magic10-
MRR-C0.DISPLAYFONT)
```

If your `DISPLAYFONTDIRECTORIES` includes `{DSK}`, then a subsequent `(FONTCREATE 'MAGIC 10)` will create a new font descriptor whose appearance is the same as the old Gacha font descriptor.

However, the new font is identical to the old one in appearance only. The individual datatype fields and bitmap may not be the same as those in the old font descriptor, due to peculiarities of different font file formats.

Font Files and Font Directories

If `FONTCREATE` is called to create a font that has not been loaded into Interlisp, `FONTCREATE` has to read the font information from a font file that contains the information for that font. For printer devices, the font files have to contain width information for each character in the font. For display fonts, the font files have to contain, in addition, bitmap images for each character in the fonts. The font file names, formats, and searching algorithms are different for each device. There are a set of variables for each device, that determine the directories that are searched for font files. All of these variables must be set before Interlisp can auto-load font files. These variables should be initialized in the site-specific `INIT` file.

DISPLAYFONTDIRECTORIES [Variable]

Value is a list of directories searched to find font bitmap files for display fonts.

DISPLAYFONTEXTENSIONS [Variable]

Value is a list of file extensions used when searching `DISPLAYFONTDIRECTORIES` for display fonts. Initially set to `(DISPLAYFONT)`, but when using older font files it may be necessary to add `STRIKE` and `AC` to this list.

INTERPRESSFONTDIRECTORIES [Variable]

Value is a list of directories searched to find font widths files for Interpress fonts.

PRESSFONTWIDTHSFILES [Variable]

Value is a list of files (not directories) searched to find font widths files for Press fonts. Press font widths are packed into large files (usually named `FONT.SWIDTHS`).

Font Profiles

`PRETTYPRINT` contains a facility for printing different elements (user functions, system functions, clisp words, comments, etc.) in different fonts to emphasize (or deemphasize) their importance, and in

INTERLISP-D REFERENCE MANUAL

general to provide for a more pleasing appearance. Of course, in order to be useful, this facility requires that the user is printing on a device (such as a bitmapped display or a laser printer) which supports multiple fonts.

PRETTYPRINT signals font changes by inserting into the file a user-defined escape sequence (the value of the variable FONTESCAPECHAR) followed by the character code which specifies, by number, which font to use, i.e. $\uparrow A$ for font number 1, etc. Thus, if FONTESCAPECHAR were the character $\uparrow F$, $\uparrow F\uparrow C$ would be output to change to font 3, $\uparrow F\uparrow A$ to change to font 1, etc. If FONTESCAPECHAR consists of characters which are separator characters in FILERDTBL, then a file with font changes in it can also be loaded back in.

Currently, PRETTYPRINT uses the following font classes. The user can specify separate fonts for each of these classes, or use the same font for several different classes.

LAMBDAFONT	The font for printing the name of the function being prettyprinted, before the actual definition (usually a large font).
CLISPFONT	If CLISPFLG is on, the font for printing any clisp words, i.e. atoms with property CLISPWORD.
COMMENTFONT	The font used for comments.
USERFONT	The font for the name of any function in the file, or any member of the list FONTFNS.
SYSTEMFONT	The font for any other (defined) function.
CHANGEFONT	The font for an expression marked by the editor as having been changed.
PRETTYCOMFONT	The font for the operand of a file package command.
DEFAULTFONT	The font for everything else.

Note that not all combinations of fonts will be aesthetically pleasing (or even readable!) and the user may have to experiment to find a compatible set.

Although in some implementations LAMBDAFONT et al. may be defined as variables, one should not set them directly, but should indicate what font is to be used for each class by calling the function FONTPROFILE:

(**FONTPROFILE** *PROFILE*) [Function]

Sets up the font classes as determined by *PROFILE*, a list of elements which defines the correspondence between font classes and specific fonts. Each element of *PROFILE* is a list of the form:

(FONTCLASS FONT# DISPLAYFONT PRESSFONT
INTERPRESSFONT)

GRAPHICS OUTPUT OPERATIONS

FONTCLASS is the font class name and FONT# is the font number for that class. For each font class name, the escape sequence will consist of FONTESCAPECHAR followed by the character code for the font number, e.g. $\uparrow$ A for font number 1, etc.

If FONT# is NIL for any font class, the font class named DEFAULTFONT (which must always be specified) is used. Alternatively, if FONT# is the name of a previously defined font class, this font class will be equivalenced to the previously defined one.

DISPLAYFONT, PRESSFONT, and INTERPRESSFONT are font specifications (of the form accepted by FONTCREATE) for the fonts to use when printing to the display and to Press and Interpress printers respectively.

FONTPROFILE

[Variable]

This is the variable used to store the current font profile, in the form accepted by the function FONTPROFILE. Note that simply editing this value will not change the fonts used for the various font classes; it is necessary to execute (FONTPROFILE FONTPROFILE) to install the value of this variable.

The process of printing with multiple fonts is affected by a large number of variables: FONTPROFILE, FILELINELENGTH, PRETTYLCOM, etc. To facilitate switching back and forth between various sets of values for the font variables, Interlisp supports the idea of named "font configurations" encapsulating the values of all relevant variables.

To create a new font configuration, set all "relevant" variables to the values you want, and then call FONTNAME to save them (on the variable FONTDEFS) under a given name. To install a particular font configuration, call FONTSET giving it your name. To change the values in a saved font configuration, edit the value of the variable FONTDEFS.

Note: The list of variables saved by FONTNAME is stored in the variable FONTDEFSVARS. This can be changed by the user.

(FONTSET NAME)

[Function]

Installs font configuration for NAME. Also evaluates (FONTPROFILE FONTPROFILE) to install the font classes as specified in the new value of the variable FONTPROFILE. Generates an error if NAME not previously defined.

FONTDEFSVARS

[Variable]

The list of variables to be packaged by a FONTNAME. Initially FONTCHANGEFLG, FILELINELENGTH, COMMENTLINELENGTH, FIRSTCOL, PRETTYLCOM, LISTFILESTR, and FONTPROFILE.

FONTDEFS

[Variable]

An association list of font configurations. FONTDEFS is a list of elements of form (NAME . PARAMETER-PAIRS). To save a configuration on a file after performing a

INTERLISP-D REFERENCE MANUAL

FONTNAME to define it, the user could either save the entire value of FONTDEFS, or use the ALISTS file package command to dump out just the one configuration.

FONTESCAPECHAR [Variable]

The character or string used to signal the start of a font escape sequence.

FONTCHANGEFLG [Variable]

If T, enables fonts when prettyprinting. If NIL, disables fonts. ALL indicates that all calls to CHANGEFONT are executed.

LISTFILESTR [Variable]

In Interlisp-10, passed to the operating system by LISTFILES. Can be used to specify subcommands to the LIST command, e.g. to establish correspondance between font number and font name.

COMMENTLINELENGTH [Variable]

Since comments are usually printed in a smaller font, COMMENTLINELENGTH is provided to offset the fact that Interlisp does not know about font widths. When FONTCHANGEFLG = T, CAR of COMMENTLINELENGTH is the linelength used to print short comments, i.e. those printed in the right margin, and CDR is the linelength used when printing full width comments.

(CHANGEFONT FONT STREAM) [Function]

Executes the operations on *STREAM* to change to the font *FONT*. For use in PRETTYPRINTMACROS.

Image Objects

An Image Object is an object that includes information about an image, such as how to display it, how to print it, and how to manipulate it when it is included in a collection of images (such as a document). More generally, it enables you to include one kind of image, with its own semantics, layout rules, and editing paradigms, inside another kind of image. Image Objects provide a general-purpose interface between image users who want to manipulate arbitrary images, and image producers, who create images for use, say, in documents.

Images are encapsulated inside a uniform barrier—the IMAGEOBJ data type. From the outside, you communicate to the image by calling a standard set of functions. For example, calling one function tells you how big the image is; calling another causes the image object to be displayed where you tell it, and so on. Anyone who wants to create images for general use can implement his own brand of IMAGEOBJ. IMAGEOBJs have been implemented (in library packages) for bitmaps, menus, annotations, graphs, and sketches.

GRAPHICS OUTPUT OPERATIONS

Image Objects were originally implemented to support inserting images into TEdit text files, but the facility is available for use by any tools that manipulate images. The Image Object interface allows objects to exist in TEdit documents and be edited with their own editor. It also provides a facility in which objects can be shift-selected (or "copy-selected") between TEdit and non-TEdit windows. For example, the Image Objects interface allows you to copy-select graphs from a Grapher window into a TEdit window. The source window (where the object comes from) does not have to know what sort of window the destination window (where the object is inserted) is, and the destination does not have to know where the insertion comes from.

A new data type, IMAGEOBJ, contains the data and the procedures necessary to manipulate an object that is to be manipulated in this way. IMAGEOBJs are created with the function IMAGEOBJCREATE (below).

Another new data type, IMAGEFNS, is a vector of the procedures necessary to define the behavior of a type of IMAGEOBJ. Grouping the operations in a separate data type allows multiple instances of the same type of image object to share procedure vectors. The data and procedure fields of an IMAGEOBJ have a uniform interface through the function IMAGEOBJPROP. IMAGEFNS are created with the function IMAGEFNSCREATE:

```
(IMAGEFNSCREATE DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN BUTTONEVENTINFN  
COPYBUTTONEVENTINFN WHENMOVEDFN WHENINSERTEDFN WHENDELETEDFN  
WHENCOPIEDFN WHENOPERATEDONFN PREPRINTFN —) [Function]
```

Returns an IMAGEFNS object that contains the functions necessary to define the behavior of an IMAGEOBJ.

The arguments *DISPLAYFN* through *PREPRINTFN* should all be function names to be stored as the "methods" of the IMAGEFNS. The purpose of each IMAGEFNS method is described below.

Note: Image objects must be "registered" before they can be read by TEdit or HREAD. IMAGEFNSCREATE implicitly registers its GETFN argument.

```
(IMAGEOBJCREATE OBJECTDATUM IMAGEFNS) [Function]
```

Returns an IMAGEOBJ that contains the object datum *OBJECTDATUM* and the operations vector *IMAGEFNS*. *OBJECTDATUM* can be arbitrary data.

```
(IMAGEOBJPROP IMAGEOBJECT PROPERTY NEWVALUE) [NoSpread Function]
```

Accesses and sets the properties of an IMAGEOBJ. Returns the current value of the *PROPERTY* property of the image object *IMAGEOBJECT*. If *NEWVALUE* is given, the property is set to it.

IMAGEOBJPROP can be used on the system properties *OBJECTDATUM*, *DISPLAYFN*, *IMAGEBOXFN*, *PUTFN*, *GETFN*, *COPYFN*, *BUTTONEVENTINFN*, *COPYBUTTONEVENTINFN*, *WHENOPERATEDONFN*, and *PREPRINTFN*. Additionally, it can be used to save arbitrary properties on an IMAGEOBJ.

INTERLISP-D REFERENCE MANUAL

(**IMAGEFNSP** *X*) [Function]

Returns *X* if *X* is an IMAGEFNS object, NIL otherwise.

(**IMAGEOBJP** *X*) [Function]

Returns *X* if *X* is an IMAGEOBJ object, NIL otherwise.

IMAGEFNS Methods

Note: Many of the IMAGEFNS methods below are passed "host stream" arguments. The TEdit text editor passes the "text stream" (an object contain all of the information in the document being edited) as the "host stream" argument. Other editing programs that want to use image objects may want to pass the data structure being edited to the IMAGEFNS methods as the "host stream" argument.

(**DISPLAYFN** *IMAGEOBJ IMAGESTREAM IMAGESTREAMTYPE HOSTSTREAM*) [IMAGEFNS Method]

The DISPLAYFN method is called to display the object *IMAGEOBJ* at the current position on *IMAGESTREAM*. The type of *IMAGESTREAM* indicates whether the device is the display or some other image stream.

Note: When the DISPLAYFN method is called, the offset and clipping regions for the stream are set so the object's image is at (0,0), and only that image area can be modified.

(**IMAGEBOXFN** *IMAGEOBJ IMAGESTREAM CURRENTX RIGHTMARGIN*) [IMAGEFNS Method]

The IMAGEBOXFN method should return the size of the object as an IMAGEBOX, which is a data structure that describes the image laid down when an *IMAGEOBJ* is displayed in terms of width, height, and descender height. An IMAGEBOX has four fields: XSIZE, YSIZE, YDESC, and XKERN. XSIZE and YSIZE are the width and height of the object image. YDESC and XKERN give the position of the baseline and the left edge of the image relative to where you want to position it. For characters, the YDESC is the descent (height of the descender) and the XKERN is the amount of left kerning (note: TEdit doesn't support left kerning).

The IMAGEBOXFN looks at the type of the stream to determine the output device if the object's size changes from device to device. (For example, a bit-map object may specify a scale factor that is ignored when the bit map is displayed on the screen.) *CURRENTX* and *RIGHTMARGIN* allow an object to take account of its environment when deciding how big it is. If these fields are not available, they are NIL.

Note: TEdit calls the IMAGEBOXFN only during line formatting, then caches the IMAGEBOX as the BOUNDBOX property of the *IMAGEOBJ*. This avoids the need to call the IMAGEBOXFN when incomplete position and margin information is available.

GRAPHICS OUTPUT OPERATIONS

(**PUTFN** *IMAGEOBJ FILESTREAM*) [IMAGEFNS Method]

The PUTFN method is called to save the object on a file. It prints a description on *FILESTREAM* that, when read by the corresponding GETFN method (see below), regenerates the image object. (TEdit and HPRINT take care of writing out the name of the GETFN.)

(**GETFN** *FILESTREAM*) [IMAGEFNS Method]

The GETFN method is called when the object is encountered on the file during input. It reads the description that was written by the PUTFN method and returns an IMAGEOBJ.

(**COPYFN** *IMAGEOBJ SOURCEHOSTSTREAM TARGETHOSTSTREAM*) [IMAGEFNS Method]

The COPYFN method is called during a copy-select operation. It should return a copy of *IMAGEOBJ*. If it returns the litatom DON'T, copying is suppressed.

(**BUTTONEVENTINFN** *IMAGEOBJ WINDOWSTREAM SELECTION RELX RELY WINDOW HOSTSTREAM BUTTON*) [IMAGEFNS Method]

The BUTTONEVENTINFN method is called when you press a mouse button inside the object. The BUTTONEVENTINFN decides whether or not to handle the button, to track the cursor in parallel with mouse movement, and to invoke selections or edits supported by the object (but see the COPYBUTTONEVENTINFN method below). If the BUTTONEVENTINFN returns NIL, TEdit treats the button press as a selection at its level. Note that when this function is first called, a button is down. The BUTTONEVENTINFN should also support the button-down protocol to descend inside of any composite objects with in it. In most cases, the BUTTONEVENTINFN relinquishes control (i.e., returns) when the cursor leaves its object's region.

When the BUTTONEVENTINFN is called, the window's clipping region and offsets have been changed so that the lower-left corner of the object's image is at (0, 0), and only the object's image can be changed. The selection is available for changing to fit your needs; the mouse button went down at (RELX, RELY) within the object's image. You can affect how TEdit treats the selection by returning one of several values. If you return NIL, TEdit forgets that you selected an object; if you return the atom DON'T, TEdit doesn't permit the selection; if you return the atom CHANGED, TEdit updates the screen. Use CHANGED to signal TEdit that the object has changed size or will have side effects on other parts of the screen image.

(**COPYBUTTONEVENTINFN** *IMAGEOBJ WINDOWSTREAM*) [IMAGEFNS Method]

The COPYBUTTONEVENTINFN method is called when you button inside an object while holding down a copy key. Many of the comments about BUTTONEVENTINFN apply here too. Also, see the discussion below about copying image objects between windows.

INTERLISP-D REFERENCE MANUAL

(**WHENMOVEDFN** *IMAGEOBJ* *TARGETWINDOWSTREAM* *SOURCEHOSTSTREAM*
TARGETHOSTSTREAM) [IMAGEFNS Method]

The WHENMOVEDFN method provides hooks by which the object is notified when TEdit performs an operation (MOVEing) on the whole object. It allows objects to have side effects.

(**WHENINSERTEDFN** *IMAGEOBJ* *TARGETWINDOWSTREAM* *SOURCEHOSTSTREAM*
TARGETHOSTSTREAM) [IMAGEFNS Method]

The WHENINSERTEDFN method provides hooks by which the object is notified when TEdit performs an operation (INSERTing) on the whole object. It allows objects to have side effects.

(**WHENDELETEDFN** *IMAGEOBJ* *TARGETWINDOWSTREAM*) [IMAGEFNS Method]

The WHENDELETEDFN method provides hooks by which the object is notified when TEdit performs an operation (DELETEing) on the whole object. It allows objects to have side effects.

(**WHENCOPIEDFN** *IMAGEOBJ* *TARGETWINDOWSTREAM* *SOURCEHOSTSTREAM*
TARGETHOSTSTREAM) [IMAGEFNS Method]

The WHENCOPIEDFN method provides hooks by which the object is notified when TEdit performs an operation (COPYing) on the whole object. The WHENCOPIEDFN method is called in addition to (and after) the COPYFN method above. It allows objects to have side effects.

(**WHENOPERATEDONFN** *IMAGEOBJ* *WINDOWSTREAM* *HOWOPERATEDON* *SELECTION*
HOSTSTREAM) [IMAGEFNS Method]

The WHENOPERATEDONFN method provides a hook for edit operations. HOWOPERATEDON should be one of SELECTED, DESELECTED, HIGHLIGHTED, and UNHIGHLIGHTED. The WHENOPERATEDONFN differs from the BUTTONEVENTINFN because it is called when you extend a selection through the object. That is, the object is treated in toto as a TEdit character. HIGHLIGHTED refers to the selection being highlighted on the screen, and UNHIGHLIGHTED means that the highlighting is being turned off.

(**PREPRINTFN** *IMAGEOBJ*) [IMAGEFNS Method]

The PREPRINTFN method is called to convert the object into something that can be printed for inclusion in documents. It returns an object that the receiving window can print (using either PRIN1 or PRIN2, its choice) to obtain a character representation of the object. If the PREPRINTFN method is NIL, the OBJECTDATUM field of *IMAGEOBJ* itself is used. TEdit uses this function when you indicate that you want to print the characters from an object rather than the object itself (presumably using PRIN1 case).

Registering Image Objects

Each legitimate GETFN needs to be known to the system, to prevent various Trojan-horse problems and to allow the automatic loading of the supporting code for infrequently used IMAGEOBJS. To this end, there is a global list, IMAGEOBJGETFNS, that contains an entry for each GETFN. The existence of the entry marks the GETFN as legitimate; the entry itself is a property list, which can hold information about the GETFN.

No action needs to be taken for GETFNs that are currently in use: the function IMAGEFNSCREATE automatically adds its GETFN argument to the list. However, packages that support obsolete versions of objects may need to explicitly add the obsolete GETFNs. For example, TEdit supports bit-map IMAGEOBJS. Recently, a change was made in the format in which objects are stored; to retain compatibility with the old object format, there are now two GETFNs. The current GETFN is automatically on the list, courtesy of IMAGEFNSCREATE. However, the code file that supports the old bit-map objects contains the clause: (ADDVARS (IMAGEOBJGETFNS (OLDGETFNNAME))), which adds the old GETFN to IMAGEOBJGETFNS.

For a given GETFN, the entry on IMAGEOBJGETFNS may be a property list of information. Currently the only recognized property is FILE.

FILE is the name of the file that can be loaded if the GETFN isn't defined. This file should define the GETFN, along with all the other functions needed to support that kind of IMAGEOBJ.

For example, the bit-map IMAGEOBJ implemented by TEdit use the GETFN BMOBJ.GETFN2. Its entry on IMAGEOBJGETFNS is (BMOBJ.GETFN2 FILE IMAGEOBJ), indicating that the support code for bit-map image objects resides on the file IMAGEOBJ, and that the GETFN for them is BMOBJ.GETFN2.

This makes it possible to have entries for GETFNs whose supporting code isn't loaded—you might, for instance, have your init file add entries to IMAGEOBJGETFNS for the kinds of image objects you commonly use. The system's default reading method will automatically load the code when necessary.

Reading and Writing Image Objects on Files

Image Objects can be written out to files using HPRINT and read back using HREAD. The following functions can also be used:

(WRITEIMAGEOBJ IMAGEOBJ STREAM) [Function]

Prints (using PRIN2) a call to READIMAGEOBJ, then calls the PUTFN for IMAGEOBJ to write it onto STREAM. During input, then, the call to READIMAGEOBJ is read and evaluated; it in turn reads back the object's description, using the appropriate GETFN.

INTERLISP-D REFERENCE MANUAL

(**READIMAGEOBJ** *STREAM GETFN NOERROR*)

[Function]

Reads an IMAGEOBJ from *STREAM*, starting at the current file position. Uses the function *GETFN* after validating it (and loading support code, if necessary).

If the *GETFN* can't be validated or isn't defined, READIMAGEOBJ returns an "encapsulated image object", an IMAGEOBJ that safely encapsulates all of the information in the image object. An encapsulated image object displays as a rectangle that says, "Unknown IMAGEOBJ Type" and lists the *GETFN*'s name. Selecting an encapsulated image object with the mouse causes another attempt to read the object from the file; this is so you can load any necessary support code and then get to the object.

Warning: You cannot save an encapsulated image object on a file because there isn't enough information to allow copying the description to the new file from the old one.

If *NOERROR* is non-NIL, READIMAGEOBJ returns NIL if it can't successfully read the object.

Copying Image Objects Between Windows

Copying between windows is implemented as follows: If a button event occurs in a window when a copy key is down, the window's COPYBUTTONEVENTFN window property is called. If this window supports copy-selection, it should track the mouse, indicating the item to be copied. When the button is released, the COPYBUTTONEVENTFN should create an image object out of the selected information, and call COPYINSERT to insert it in the current TTY window. COPYINSERT calls the COPYINSERTFN window property of the TTY window to insert this image object. Therefore, both the source and destination windows can determine how they handle copying image objects.

If the COPYBUTTONEVENTFN of a window is NIL, the BUTTONEVENTFN is called instead when a button event occurs in the window when a copy key is down, and copying from that window is not supported. If the COPYINSERTFN of the TTY window is NIL, COPYINSERT will turn the image object into a string (by calling the PREPRINTFN method of the image object) and insert it by calling BKSYSBUF.

COPYBUTTONEVENTFN

[Window Property]

The COPYBUTTONEVENTFN of a window is called (if it exists) when a button event occurs in the window and a copy key is down. If no COPYBUTTONEVENTFN exists, the BUTTONEVENTFN is called.

COPYINSERTFN

[Window Property]

The COPYINSERTFN of the "destination" window is called by COPYINSERT to insert something into the destination window. It is called with two arguments: the object to be inserted and the destination window. The object to be inserted can be a character string, an IMAGEOBJ, or a list of IMAGEOBJS and character strings. As a convention, the COPYINSERTFN should call BKSYSBUF if the object to be inserted insert is a character string.

GRAPHICS OUTPUT OPERATIONS

(**COPYINSERT** *IMAGEOBJ*)

[Function]

COPYINSERT inserts *IMAGEOBJ* into the window that currently has the TTY. If the current TTY window has a COPYINSERTFN, it is called, passing it *IMAGEOBJ* and the window as arguments.

If no COPYINSERTFN exists and if *IMAGEOBJ* is an image object, BKSYSBUF is called on the result of calling its PREPRINTFN on it. If *IMAGEOBJ* is not an image object, it is simply passed to BKSYSBUF. In this case, BKSYSBUF will call PRIN2 with a read table taken from the process associated with the TTY window. A window that wishes to use PRIN1 or a different read table must provide its own COPYINSERTFN to do this.

Implementation of Image Streams

Interlisp does all image creation through a set of functions and data structures for device-independent graphics, known popularly as DIG. DIG is implemented through the use of a special type of stream, known as an image stream.

An image stream, by convention, is any stream that has its IMAGEOPS field (described in detail below) set to a vector of meaningful graphical operations. Using image streams, you can write programs that draw and print on an output stream without regard to the underlying device, be it a window, a disk, or a printer.

To define a new image stream type, it is necessary to put information on the variable IMAGESTREAMTYPES:

IMAGESTREAMTYPES

[Variable]

This variable describes how to create a stream for a given image stream type. The value of IMAGESTREAMTYPES is an association list, indexed by the image stream type (e.g., DISPLAY, INTERPRESS, etc.). The format of a single association list item is:

```
(IMAGETYPE
  (OPENSTREAM OPENSTREAMFN)
  (FONTCREATE FONTCREATEFN)
  (FONTSAVAILABLE FONTSAVAILABLEFN) )
```

OPENSTREAMFN, FONTCREATEFN, and FONTSAVAILABLEFN are "image stream methods," device-dependent functions used to implement generic image stream operations. For Interpress image streams, the association list entry is:

```
(INTERPRESS
  (OPENSTREAM OPENIPSTREAM)
  (FONTCREATE \CREATEINTERPRESSFONT)
  (FONTSAVAILABLE \SEARCHINTERPRESSFONTS) )
```

INTERLISP-D REFERENCE MANUAL

(**OPENSTREAMFN** *FILE* *OPTIONS*) [Image Stream Method]

FILE is the file name as it was passed to OPENIMAGESTREAM, and *OPTIONS* is the *OPTIONS* property list passed to OPENIMAGESTREAM. The result must be a stream of the appropriate image type.

(**FONTCREATEFN** *FAMILY* *SIZE* *FACE* *ROTATION* *DEVICE*) [Image Stream Method]

FAMILY is the family name for the font, e.g., MODERN. *SIZE* is the body size of the font, in printer's points. *FACE* is a three-element list describing the weight, slope, and expansion of the face desired, e.g., (MEDIUM ITALIC EXPANDED). *ROTATION* is how much the font is to be rotated from the normal orientation, in minutes of arc. For example, to print a landscape page, fonts have the rotation 5400 (90 degrees). The function's result must be a FONTDESCRIPTOR with the fields filled in appropriately.

(**FONTSAVAILABLEFN** *FAMILY* *SIZE* *FACE* *ROTATION* *DEVICE*) [Image Stream Method]

This function returns a list of all fonts agreeing with the *FAMILY*, *SIZE*, *FACE*, and *ROTATION* arguments; any of them may be wild-carded (i.e., equal to *, which means any value is acceptable). Each element of the list should be a quintuple of the form (*FAMILY* *SIZE* *FACE* *ROTATION* *DEVICE*).

Where the function looks is an implementation decision: the FONTSAVAILABLEFN for the display device looks at DISPLAYFONTDIRECTORIES, the Interpress code looks on INTERPRESSFONTDIRECTORIES, and implementors of new devices should feel free to introduce new search path variables.

As indicated above, image streams use a field that no other stream uses: IMAGEOPS. IMAGEOPS is an instance of the IMAGEOPS data type and contains a vector of the stream's graphical methods. The methods contained in the IMAGEOPS object can make arbitrary use of the stream's IMAGEDATA field, which is provided for their use, and may contain any data needed.

IMAGETYPE [IMAGEOPS Field]

Value is the name of an image type. Monochrome display streams have an IMAGETYPE of DISPLAY; color display streams are identified as (COLOR DISPLAY). The IMAGETYPE field is informational and can be set to anything you choose.

IMFONTCREATE [IMAGEOPS Field]

Value is the device name to pass to FONTCREATE when fonts are created for the stream.

The remaining fields are all image stream methods, whose value should be a device-dependent function that implements the generic operation. Most methods are called by a similarly-named function, e.g. the function DRAWLINE calls the IMDRAWLINE method. All coordinates that refer to points in a display device's space are measured in the device's units. (The IMSCALE method provides access to a device's scale.) For arguments that have defaults (such as the BRUSH argument of DRAWCURVE), the default is substituted

GRAPHICS OUTPUT OPERATIONS

for the `NIL` argument before it is passed to the image stream method. Therefore, image stream methods do not have to handle defaults.

(**IMCLOSEFN** *STREAM*) [Image Stream Method]

Called before a stream is closed with `CLOSEF`. This method should flush buffers, write header or trailer information, etc.

(**IMDRAWLINE** *STREAM X Y X Y WIDTH OPERATION COLOR DASHING*) [Image Stream Method]

Draws a line of width *WIDTH* from (*X* , *Y*) to (*X* , *Y*). See `DRAWLINE`.

(**IMDRAWCURVE** *STREAM KNOTS CLOSED BRUSH DASHING*) [Image Stream Method]

Draws a curve through *KNOTS*. See `DRAWCURVE`.

(**IMDRAWCIRCLE** *STREAM CENTERX CENTERY RADIUS BRUSH DASHING*) [Image Stream Method]

Draws a circle of radius *RADIUS* around (*CENTERX*, *CENTERY*). See `DRAWCIRCLE`.

(**IMDRAWELLIPSE** *STREAM CENTERX CENTERY SEMIMINORRADIUS SEMIMAJORRADIUS ORIENTATION BRUSH DASHING*) [Image Stream Method]

Draws an ellipse around (*CENTERX*, *CENTERY*). See `DRAWELLIPSE`.

(**IMFILLPOLYGON** *STREAM POINTS TEXTURE*) [Image Stream Method]

Fills in the polygon outlined by *POINTS* on the image stream *STREAM*, using the texture *TEXTURE*. See `FILLPOLYGON`.

(**IMFILLCIRCLE** *STREAM CENTERX CENTERY RADIUS TEXTURE*) [Image Stream Method]

Draws a circle filled with texture *TEXTURE* around (*CENTERX*, *CENTERY*). See `FILLCIRCLE`.

(**IMBLTSHADE** *TEXTURE STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT OPERATION CLIPPINGREGION*) [Image Stream Method]

The texture-source case of `BITBLT`. *DESTINATIONLEFT*, *DESTINATIONBOTTOM*, *WIDTH*, *HEIGHT*, and *CLIPPINGREGION* are measured in *STREAM*'s units. This method is invoked by the functions `BITBLT` and `BLTSHADE`.

(**IMBITBLT** *SOURCEBITMAP SOURCELEFT SOURCEBOTTOM STREAM DESTINATIONLEFT DESTINATIONBOTTOM WIDTH HEIGHT SOURCETYPE OPERATION TEXTURE CLIPPINGREGION CLIPPEDSOURCELEFT CLIPPEDSOURCEBOTTOM SCALE*) [Image Stream Method]

Contains the bit-map-source cases of `BITBLT`. *SOURCELEFT*, *SOURCEBOTTOM*, *CLIPPEDSOURCELEFT*, *CLIPPEDSOURCEBOTTOM*, *WIDTH*, and *HEIGHT* are measured

INTERLISP-D REFERENCE MANUAL

in pixels; *DESTINATIONLEFT*, *DESTINATIONBOTTOM*, and *CLIPPINGREGION* are in the units of the destination stream.

(**IMSCALEDBITBLT** *SOURCEBITMAP* *SOURCELEFT* *SOURCEBOTTOM* *STREAM*
DESTINATIONLEFT *DESTINATIONBOTTOM* *WIDTH* *HEIGHT* *SOURCETYPE* *OPERATION*
TEXTURE *CLIPPINGREGION* *CLIPPEDSOURCELEFT* *CLIPPEDSOURCEBOTTOM* *SCALE*) [*I*
image Stream Method]

A scaled version of *IMBITBLT*. Each pixel in *SOURCEBITMAP* is replicated *SCALE* times in the X and Y directions; currently, *SCALE* must be an integer.

(**IMMOVETO** *STREAM* *X* *Y*) [Image Stream Method]

Moves to (*X*, *Y*). This method is invoked by the function *MOVETO*. If *IMMOVETO* is not supplied, a default method composed of calls to the *IMXPOSITION* and *IMYPOSITION* methods is used.

(**IMSTRINGWIDTH** *STREAM* *STR* *RDTBL*) [Image Stream Method]

Returns the width of string *STR* in *STREAM*'s units, using *STREAM*'s current font. This is invoked when *STRINGWIDTH* is passed a stream as its *FONT* argument. If *IMSTRINGWIDTH* is not supplied, it defaults to calling *STRINGWIDTH* on the default font of *STREAM*.

(**IMCHARWIDTH** *STREAM* *CHARCODE*) [Image Stream Method]

Returns the width of character *CHARCODE* in *STREAM*'s units, using *STREAM*'s current font. This is invoked when *CHARWIDTH* is passed a stream as its *FONT* argument. If *IMCHARWIDTH* is not supplied, it defaults to calling *CHARWIDTH* on the default font of *STREAM*.

(**IMCHARWIDTHY** *STREAM* *CHARCODE*) [Image Stream Method]

Returns the Y component of the width of character *CHARCODE* in *STREAM*'s units, using *STREAM*'s current font. This is invoked when *CHARWIDTHY* is passed a stream as its *FONT* argument. If *IMCHARWIDTHY* is not supplied, it defaults to calling *CHARWIDTHY* on the default font of *STREAM*.

(**IMBITMAPSIZE** *STREAM* *BITMAP* *DIMENSION*) [Image Stream Method]

Returns the size that *BITMAP* will be when *BITBLT*ed to *STREAM*, in *STREAM*'s units. *DIMENSION* can be one of *WIDTH*, *HEIGHT*, or *NIL*, in which case the dotted pair (*WIDTH* . *HEIGHT*) will be returned.

This is invoked by *BITMAPIMAGE*SIZE. If *IMBITMAPSIZE* is not supplied, it defaults to a method that multiplies the bitmap height and width by the scale of *STREAM*.

GRAPHICS OUTPUT OPERATIONS

(**IMNEWPAGE** *STREAM*) [Image Stream Method]

Causes a new page to be started. The X position is set to the left margin, and the Y position is set to the top margin plus the linefeed. If not supplied, defaults to (`\OUTCHAR STREAM (CHARCODE ^L)`). Invoked by DSPNEWPAGE.

(**IMTERPRI** *STREAM*) [Image Stream Method]

Causes a new line to be started. The X position is set to the left margin, and the Y position is set to the current Y position plus the linefeed. If not supplied, defaults to (`\OUTCHAR STREAM (CHARCODE EOL)`). Invoked by TERPRI.

(**IMRESET** *STREAM*) [Image Stream Method]

Resets the X and Y position of *STREAM*. The X coordinate is set to its left margin; the Y coordinate is set to the top of the clipping region minus the font ascent. Invoked by DSPRESET.

The following methods all have corresponding DSPxx functions (e.g., IMYPOSITION corresponds to DSPYPOSITION) that invoke them. They also have the property of returning their previous value; when called with NIL they return the old value without changing it.

(**IMCLIPPINGREGION** *STREAM REGION*) [Image Stream Method]

Sets a new clipping region on *STREAM*.

(**IMXPOSITION** *STREAM XPOSITION*) [Image Stream Method]

Sets the X-position on *STREAM*.

(**IMYPOSITION** *STREAM YPOSITION*) [Image Stream Method]

Sets a new Y-position on *STREAM*.

(**IMFONT** *STREAM FONT*) [Image Stream Method]

Sets *STREAM*'s font to be *FONT*.

(**IMLEFTMARGIN** *STREAM LEFTMARGIN*) [Image Stream Method]

Sets *STREAM*'s left margin to be *LEFTMARGIN*. The left margin is defined as the X-position set after the new line.

(**IMRIGHTMARGIN** *STREAM RIGHTMARGIN*) [Image Stream Method]

Sets *STREAM*'s right margin to be *RIGHTMARGIN*. The right margin is defined as the maximum X-position at which characters are printed; printing beyond it causes a new line.

INTERLISP-D REFERENCE MANUAL

(**IMTOPMARGIN** *STREAM YPOSITION*) [Image Stream Method]

Sets *STREAM*'s top margin (the Y-position of the tops of characters that is set after a new page) to be *YPOSITION*.

(**IMBOTTOMMARGIN** *STREAM YPOSITION*) [Image Stream Method]

Sets *STREAM*'s bottom margin (the Y-position beyond which any printing causes a new page) to be *YPOSITION*.

(**IMLINEFEED** *STREAM DELTA*) [Image Stream Method]

Sets *STREAM*'s line feed distance (distance to move vertically after a new line) to be *DELTA*.

(**IMSCALE** *STREAM SCALE*) [Image Stream Method]

Returns the number of device points per screen point (a screen point being $\sim 1/72$ inch). *SCALE* is ignored.

(**IMSPACEFACTOR** *STREAM FACTOR*) [Image Stream Method]

Sets the amount by which to multiply the natural width of all following space characters on *STREAM*; this can be used for the justification of text. The default value is 1. For example, if the natural width of a space in *STREAM*'s current font is 12 units, and the space factor is set to two, spaces appear 24 units wide. The values returned by *STRINGWIDTH* and *CHARWIDTH* are also affected.

(**IMOPERATION** *STREAM OPERATION*) [Image Stream Method]

Sets the default BITBLT *OPERATION* argument.

(**IMBACKCOLOR** *STREAM COLOR*) [Image Stream Method]

Sets the background color of *STREAM*.

(**IMCOLOR** *STREAM COLOR*) [Image Stream Method]

Sets the default color of *STREAM*.

In addition to the IMAGEOPS methods described above, there are two other important methods, which are contained in the stream itself. These fields can be installed using a form like (replace (STREAM OUTCHARFN) of STREAM with (FUNCTION MYOUTCHARFN)). Note: You need to have loaded the Interlisp-D system declarations to manipulate the fields of STREAMs. The declarations can be loaded by loading the Lisp Library package SYSEdit.

(**STRMBOUTFN** *STREAM CHARCODE*) [Stream Method]

The function called by BOUT.

GRAPHICS OUTPUT OPERATIONS

(**OUTCHARFN** *STREAM CHARCODE*)

[Stream Method]

The function that is called to output a single byte. This is like `STRMBOUTFN`, except for being one level higher: it is intended for text output. Hence, this function should convert (*CHARCODE* EOL) into the stream's actual end-of-line sequence and should adjust the stream's `CHARPOSITION` appropriately before invoking the stream's `STRMBOUTFN` (by calling `BOUT`) to actually put the character. Defaults to `\FILEOUTCHARFN`, which is probably incorrect for an image stream.

27. WINDOWS AND MENUS

Windows provide a means by which different programs can share a single display harmoniously. Rather than having every program directly manipulating the screen bitmap, all display input/output operations are directed towards windows, which appear as rectangular regions of the screen, with borders and titles. The Interlisp-D window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion (see the Windows section below). This allows existing Interlisp programs to be used without change, while providing a base for experimentation with more complex windows in new applications.

Menus are a special type of window provided by the window system, used for displaying a set of items to the user, and having the user select one using the mouse and cursor. The window system uses menus to provide the interactive interface for manipulating windows. The menu facility also allows users to create and use menus in interactive programs (see the Menus section below).

Sometimes, a program needs to use a number of windows, displaying related information. The attached window facility (see the Attached Windows section below) makes it easy to manipulate a group of windows as a single unit, moving and reshaping them together.

This chapter documents the Interlisp-D window system. First, it describes the default windows and menus supplied by the window system. Then, the programmatic facilities for creating windows. Next, the functions for using menus. Finally, the attached window facility.

Warning: The window system assumes that all programs follow certain conventions concerning control of the screen. All user programs should use perform display operations using windows and menus. In particular, user programs should not perform operate directly on the screen bitmap; otherwise the window system will not work correctly. For specialized applications that require taking complete control of the display, the window system can be turned off (and back on again) with the following function:

(**WINDOWWORLD** *FLAG*)

[NoSpread Function]

The window system is turned on if *FLAG* is T and off if *FLAG* is NIL. WINDOWWORLD returns the previous state of the window system (T or NIL). If WINDOWWORLD is given no arguments, it simply returns the current state without affecting the window system.

Using the Window System

When Medley is initially started, the display screen lights up, showing a number of windows, including the following:

INTERLISP-D REFERENCE MANUAL



This window is the "logo window," used to identify the system. The logo window is bound to the variable `LOGOW` until it is closed. The user can create other windows like this by calling the following function:

(LOGOW *STRING* *WHERE* *TITLE* *ANGLEDELTA*) [Function]

Creates a window formatted like the "logo window." *STRING* is the string to be printed in big type in the window; if `NIL`, "Medley" is used. *WHERE* is the position of the lower-left corner of the window; if `NIL`, the user is asked to specify a position. *TITLE* is the window title to use; if `NIL`, it defaults to the Xerox copyright notice and date. *ANGLEDELTA* specifies the angle (in degrees) between the boxes in the picture; if `NIL`, it defaults to 23 degrees.



This window is the "executive window," used for typing expressions and commands to the Interlisp-D executive, and for the executive to print any results (see Chapter 13). For example, in the above picture, the user typed in `(PLUS 3 4)`, the executive evaluated it, and printed out the result, 7. The upward-pointing arrow is the flashing caret, which indicates where the next keyboard typein will be printed (see the TTY Process and the Caret section in this chapter).



This window is the "prompt window," used for printing various system prompt messages. It is available to user programs through the following functions:

PROMPTWINDOW [Variable]

Global variable containing the prompt window.

(PROMPTPRINT *EXP* ... *EXP*) [NoSpread Function]

Clears the prompt window, and prints *EXP* through *EXP* in the prompt window.

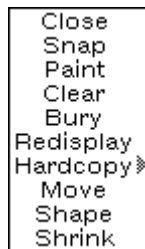
(CLRSPROMPT) [Function]

WINDOWS AND MENUS

Clears the prompt window.

The Medley window system allows the user to interactively manipulate the windows on the screen, moving them around, changing their shape, etc. by selecting various operations from a menu.

For most windows, pressing the RIGHT mouse button when the cursor is inside a window during I/O wait will cause the window to come to the top and a menu of window operations to appear.



If a command is selected from this menu (by releasing the right mouse key while the cursor is over a command), the selected operation will be applied to the window in which the menu was brought up. It is possible for an applications program to redefine the action of the RIGHT mouse button. In these cases, there is a convention that the default command menu may be brought up by depressing the RIGHT button when the cursor is in the header or border of a window (see the Mouse Activity in Windows section in this chapter). The operations are:

Close [Window Menu Command]

Closes the window, i.e, removes it from the screen. (See CLOSEW in the Opening and Closing Windows section in this chapter.)

Snap [Window Menu Command]

Prompts for a region on the screen and makes a new window whose bits are a snapshot of the bits currently in that region. Useful for saving some particularly choice image before the window image changes.

Paint [Window Menu Command]

Switches to a mode in which the cursor can be used like a paint brush to draw in a window. This is useful for making notes on a window. While the LEFT button is down, bits are added. While the MIDDLE button is down, they are erased. The RIGHT button pops up a command menu that allows changing of the brush shape, size and shade, changing the mode of combining the brush with the existing bits, or stopping paint mode.

Clear [Window Menu Command]

Clears the window and repositions it to the left margin of the first line of text (below the upper left corner of the window by the amount of the font ascent).

Bury [Window Menu Command]

INTERLISP-D REFERENCE MANUAL

Puts the window on the bottom of the occlusion stack, thereby exposing any windows that it was hiding.

Redisplay [Window Menu Command]

Redisplays the window. (See REDISPLAYW in the Redisplaying Windows section in this chapter.)

Hardcopy [Window Menu Command]

Prints the contents of the window to the printer. If the window has a window property `HARDCOPYFN`, it is called with two arguments, the window and an image stream to print to, and the `HARDCOPYFN` must do the printing. In this way, special windows can be set up that know how to print their contents in a particular way. If the window does not have a `HARDCOPYFN`, the bitmap image of the window (including the border and title) are printed on the file or printer.

To save the image in a Press or Interpress-format file, or to send it to a non-default printer, use the submenu of the Hardcopy command, indicated by a gray triangle on the right edge of the Hardcopy menu item. If the mouse is moved off of the right of the menu item, another pop-up menu will appear giving the choices "To a file" or "To a printer." If "To a file" is selected, the user is prompted to supply a file name, and the format of the file (Press, Interpress, etc.), and the specified region will be stored in the file.

If "To a printer" is selected, the user is prompted to select a printer from the list of known printers, or to type the name of another printer. If the printer selected is not the first printer on `DEFAULTPRINTINGHOST` (see Chapter 29), the user will be asked whether to move or add the printer to the beginning of this list, so that future printing will go to the new printer.

Move [Window Menu Command]

Moves the window to a location specified by pressing and then releasing the `LEFT` button. During this time a ghost frame will indicate where the window will reappear when the key is released. (See `GETBOXPOSITION` in the Interactive Display Functions section below.)

Shape [Window Menu Command]

Allows the user to specify a new region for the existing window contents. If the `LEFT` button is used to specify the new region, the reshaped window can be placed anywhere. If the `MIDDLE` button is used, the cursor will start out tugging at the nearest corner of the existing window, which is useful for making small adjustments in a window that is already positioned correctly. This is done by calling the function `SHAPEW` (see the Reshaping Windows section below).

Occasionally, a user will have a number of large windows on the screen, making it difficult to access those windows being used. To help with the problem of screen space management, the Interlisp-D window system allows the creation of "icons." An icon is a small rectangle (containing text or a bitmap) which is a "shrunken-down" form of a particular window. Using the Shrink and Expand

WINDOWS AND MENUS

commands, the user can shrink windows not currently being used into icons, and quickly restore the original windows at any time.

Shrink

[Window Menu Command]

Removes the window from the screen and brings up its icon. (See `SHRINKW` in the Shrinking Windows into Icons section in this chapter) The window can be restored by selecting Expand from the window command menu of the icon.

If the `RIGHT` button is pressed while the cursor is in an icon, the window command menu will contain a slightly different set of commands. The Redisplay and Clear commands are removed, and the Shrink command is replaced with the Expand command:

Expand

[Window Menu Command]

Restores the window associated with this icon and removes the icon. (See `EXPANDW` in the Shrinking Windows into Icons section in this chapter.)

If the `RIGHT` button is pressed while the cursor is not in any window, a "background menu" appears with the following operations:

Idle

[Background Menu Command]

Enters "idle mode" (see Chapter 12), which blacks out the display screen to save the phosphor. Idle mode can be exited by pressing any key on the keyboard or mouse. This menu command has subitems that allow the user to interactively set idle options to erase the password cache (for security), to request a password before exiting idle mode, to change the timeout before idle mode is entered automatically, etc.

SaveVM

[Background Menu Command]

Calls the function `SAVEVM` (see Chapter 12), which writes out all of the dirty pages of the virtual memory. After a `SAVEVM`, and until the pagefault handler is next forced to write out a dirty page, your virtual memory image will be continuable (as of the `SAVEVM`) should you experience a system crash or other disaster.

Snap

[Background Menu Command]

The same as the window menu command Snap described above.

Hardcopy

[Background Menu Command]

Prompts for a region on the screen, and sends the bitmap image to the printer by calling `HARDCOPYW` (see Chapter 29). Note that the region can cross window boundaries.

Like the Hardcopy window menu command (above), the user can print to a file or specify a printer by using a submenu.

PSW

[Background Menu Command]

Prompts the user for a position on the screen, and creates a "process status window" that allows the user to examine and manipulate all of the existing processes (see Chapter 23).

INTERLISP-D REFERENCE MANUAL

Various system utilities (TEdit, SEdit, TTYIN) allow information to be "copy-inserted" at the current cursor position by selecting it with the "copy" key held down (Normally the shift keys are the "copy" key; this action can be changed in the key action table.) To "copy-insert" the bitmap of a snap into a Tedit document. If the right mouse button is pressed in the background with the copy key held down, a menu with the single item "SNAP" appears. If this item is selected, the user is prompted to select a region, and a bitmap containing the bits in that region of the screen is inserted into the current tty process, if that process is able to accept image objects.

Some built-in facilities and Lispusers packages add commands to the background menu, to provide an easy way of calling the different facilities. The user can determine what these new commands do by holding the RIGHT button down for a few seconds over the item in question; an explanatory message will be printed in the prompt window.

Changing the Window System

The following functions provide a functional interface to the interactive window operations so that user programs can call them directly.

(**DOWINDOWCOM** *WINDOW*) [Function]

If *WINDOW* is a *WINDOW* that has a DOWINDOWCOMFN window property, it APPLYS that property to *WINDOW*. Shrunk windows have a DOWINDOWCOMFN property that presents a window command menu that contains "expand" instead of "shrink".

If *WINDOW* is a *WINDOW* that doesn't have a DOWINDOWCOMFN window property, it brings up the window command menu. The initial items in these menus are described above. If the user selects one of the items from the provided menu, that item is APPLyEd to *WINDOW*.

If *WINDOW* is NIL, DOBACKGROUNDCOM (below) is called.

If *WINDOW* is not a *WINDOW* or NIL, DOWINDOWCOM simply returns without doing anything.

(**DOBACKGROUNDCOM**) [Function]

Brings up the background menu. The initial items in this menu are described above. If the user selects one of the items from the menu, that item is EVALed.

The window command menu for unshrunk windows is cached in the variable WindowMenu. To change the entries in this menu, the user should change the change the menu "command lists" in the variable WindowMenuCommands, and set the appropriate menu variable to a non-MENU, so the menu will be recreated. This provides a way of adding commands to the menu, of changing its font or of restoring the menu if it gets clobbered. The window command menus for icons and the background have similar pairs of variables, documented below. The "command lists" are in the format of the ITEMS field of a menu (see the Menu Fields section below), except as specified below.

Note: Command menus are recreated using the current value of MENUFONT.

WINDOWS AND MENUS

WindowMenu [Variable]
WindowMenuCommands [Variable]

The menu that is brought up in response to a right button in an unshrunk window is stored on the variable `WindowMenu`. If `WindowMenu` is set to a non-MENU, the menu will be recreated from the list of commands `WindowMenuCommands`. The CADR of each command added to `WindowMenuCommands` should be a function name that will be APPLIED to the window.

IconWindowMenu [Variable]
IconWindowMenuCommands [Variable]

The menu that is brought up in response to a right button in a shrunken window is stored on the variable `IconWindowMenu`. If it is NIL, it is recreated from the list of commands `IconWindowMenuCommands`. The CADR of each command added a function name that will be APPLIED to the window.

BackgroundMenu [Variable]
BackgroundMenuCommands [Variable]

The menu that is brought up in response to a right button in the background is stored on the variable `BackgroundMenu`. If it is NIL, it is recreated from the list of commands `BackgroundMenuCommands`. The CADR of each command added to `BackgroundMenuCommands` should be a form that will be EVALed.

BackgroundCopyMenu [Variable]
BackgroundCopyMenuCommands [Variable]

The menu that is brought up in response to a right button in the background when the copy key is down is stored on the variable `BackgroundCopyMenu`. If it is NIL, it is recreated from the list of commands `BackgroundCopyMenuCommands`. The CADR of each command added to `BackgroundCopyMenuCommands` should be a form that will be EVALed.

Interactive Display Functions

The following functions can be used by programs to allow the user to interactively specify positions or regions on the display screen.

(GETPOSITION WINDOW CURSOR) [Function]

Returns a POSITION that is specified by the user. GETPOSITION waits for the user to press and release the left button of the mouse and returns the cursor position at the time of release. If *WINDOW* is a WINDOW, the position will be in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position will be in screen coordinates. If *CURSOR* is a CURSOR (see Chapter 30), the cursor will be changed to it while GETPOSITION is running. If *CURSOR* is NIL, the value of the system variable

CROSSHAIRS will be used as the cursor: 

INTERLISP-D REFERENCE MANUAL

(**GETBOXPOSITION** *BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Allows the user to position a "ghost" region of size *BOXWIDTH* by *BOXHEIGHT* on the screen, and returns the *POSITION* of the lower left corner of the region. If *PROMPTMSG* is non-NIL, GETBOXPOSITION first prints it in the PROMPTWINDOW. GETBOXPOSITION

then changes the cursor to a box (using the global variable BOXCURSOR: ). If *ORGX* and *ORGY* are numbers, they are taken to be the original position of the region, and the cursor is moved to the nearest corner of that region. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. If *ORGX* and *ORGY* are numbers, the corner of the region formed by (*ORGX ORGY BOXWIDTH BOXHEIGHT*) that is nearest the cursor position is locked, otherwise the lower left corner is locked. The user can change to another corner by holding down the right button. With the right button down, the cursor can be moved across the screen without effect on the ghost region frame. When the right button is released, the mouse will snap to the nearest corner, which will then become locked to the cursor. (The held corner can be changed after the left or middle button is down by holding both the original button and the right button down while the cursor is moved to the desired new corner, then letting up just the right button.) When the left or middle button is pressed and released, the lower left corner of the region at the time of release is returned. If *WINDOW* is a *WINDOW*, the returned position will be in *WINDOW*'s coordinate system; otherwise it will be in screen coordinates.

Example:

```
(GETBOXPOSITION 100 200 NIL NIL NIL
  "Specify the position of the command area.")
```

prompts the user for a 100 wide by 200 high region and returns its lower left corner in screen coordinates.

(**GETREGION** *MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS*) [Function]

Lets the user specify a new region and returns that region in screen coordinates. GETREGION prompts for a region by displaying a four-pronged box next to the cursor

arrow at one corner of a "ghost" region: . If the user presses the left button, the corner of a "ghost" region opposite the cursor is locked where it is. Once one corner has been fixed, the ghost region expands as the cursor moves.

To specify a region:

1. Move the ghost box so that the corner opposite the cursor is at one corner of the intended region.
2. Press the left button.
3. Move the cursor to the position of the opposite corner of the intended region while holding down the left button.
4. Release the left button.

WINDOWS AND MENUS

Before one corner has been fixed, one can switch the cursor to another corner of the ghost region by holding down the right button. With the right button down, the cursor changes to a "forceps"  and the cursor can be moved across the screen without effect on the ghost region frame. When the right button is released, the cursor will snap to the nearest corner of the ghost region.

After one corner has been fixed, one can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across the screen without effect on the ghost region frame. When the right button is released, the cursor will snap to the nearest corner, which will become the moving corner. In this way, the region may be moved all over the screen, before its size and position is finalized.

The size of the initial ghost region is controlled by the *MINWIDTH*, *MINHEIGHT*, *OLDREGION*, and *INITCORNERS* arguments.

If *INITCORNERS* is non-NIL, it should be a list specifying the initial corners of a ghost region of the form (*BASEX* *BASEY* *OPPX* *OPPY*), where (*BASEX*, *BASEY*) describes the anchored corner of the box, and (*OPPX*, *OPPY*) describes the trackable corner (in screen coordinates). The cursor is moved to (*OPPX*, *OPPY*).

If *INITCORNERS* is NIL, the ghost region will be *MINWIDTH* wide and *MINHEIGHT* high. If *MINWIDTH* or *MINHEIGHT* is NIL, 0 is used. Thus, for a call to *GETREGION* with no arguments specified, there will be no initial ghost region. The cursor will be in the lower right corner of the region, if there is one.

If *OLDREGION* is a region and the user presses the middle button, the corner of *OLDREGION* farthest from the cursor position is fixed and the corner nearest the cursor is locked to the cursor.

MINWIDTH and *MINHEIGHT*, if given, are the smallest *WIDTH* and *HEIGHT* that the returned region will have. The ghost image will not get any smaller than *MINWIDTH* by *MINHEIGHT*.

If *NEWREGIONFN* is non-NIL, it will be called to determine values for the positions of the corners. This provides a way of "filtering" prospective regions; for instance, by restricting the region to lie on an arbitrary grid. When the user is specifying a region, the region is determined by two of its corners, one that is fixed and one that is tracking the cursor. Each time the cursor moves or a mouse button is pressed, *NEWREGIONFN* is called with three arguments: *FIXEDPOINT*, the position of the fixed corner of the prospective region; *MOVINGPOINT*, the position of the opposite corner of the prospective region; and *NEWREGIONFNARG*. *NEWREGIONFNARG* allows the caller of *GETREGION* to pass information to the *NEWREGIONFN*.

The first time a button is pressed and when the user changes the moving corner via right buttoning, *MOVINGPOINT* is NIL and *FIXEDPOINT* is the position the user selected for the fixed corner of the new region. In this case, the position returned by *NEWREGIONFN* will be used for the fixed corner instead of the one proposed by the user. For all other calls, *FIXEDPOINT* is the position of the fixed corner (as returned by the previous call) and

INTERLISP-D REFERENCE MANUAL

MOVINGPOINT is the new position the user selected for the opposite corner. In these cases, the value of *NEWREGIONFN* is used for the opposite corner instead of the one proposed by the user. In all cases, the ghost region is drawn with the values returned by *NEWREGIONFN*. *NEWREGIONFN* can be a list of functions in which case they are called in order with each being passed the result of calling the previous and the value of the last one used as the point.

(**GETBOXREGION** *WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Performs the same prompting as *GETBOXPOSITION* and returns the *REGION* specified by the user instead of the *POSITION* of its lower left corner.

(**MOUSECONFIRM** *PROMPTSTRING HELPSTRING WINDOW DON'TCLEARWINDOWFLG*) [Function]

MOUSECONFIRM provides a simple way for the user to confirm or abort some action simply by using the mouse buttons. It prints the strings *PROMPTSTRING* and

HELPSTRING in the window *WINDOW*, changes the cursor to a "little mouse" cursor: , (stored in the variable *MOUSECONFIRMCURSOR*), and waits for the user to press the left button to confirm, or any other button to abort. If the left button was the last button released, returns T, else NIL.

If *PROMPTSTRING* is NIL, it is not printed out. If *HELPSTRING* is NIL, the string "Click LEFT to confirm, RIGHT to abort." is used. If *WINDOW* is NIL, the prompt window is used.

Normally, *MOUSECONFIRM* clears *WINDOW* before returning. If *DON'TCLEARWINDOWFLG* is non-NIL, the window is not cleared.

Windows

A window specifies a region of the screen, a display stream, functions that get called when the window undergoes certain actions, and various other items of information. The basic model is that a window is a passive collection of bits (on the screen). On top of this basic level, the system supports many different types of windows that are linked to the data structures displayed in them and provide selection and redisplaying routines. In addition, it is possible for the user to create new types of windows by providing selection and displaying functions for them.

Windows are ordered in depth from user to background. Windows in front of others obscure the latter. Operating on a window generally brings it to the top.

Windows are located at a certain position on the screen. Each window has a clipping region that confines all bits written to it to a region that allows a border around the window, and a title above it.

Each window has a display stream associated with it (see Chapter 27), and either a window or its display stream can be passed interchangeably to all system functions. There are dependencies

WINDOWS AND MENUS

between the window and its display stream that the user should not disturb. For instance, the destination bitmap of the display stream of a window must always be the screen bitmap. The X offset, Y offset, and Clipping Region fields of the display stream should not be changed.

Windows can be created by the user interactively, under program control, or may be created automatically by the system.

Windows are in one of two states: "open" or "closed". In an "open" state, a window is visible on the screen (unless it is covered by other open windows or off the edge of the screen) and accessible to mouse operations. In a "closed" state, a window is not visible and not accessible to mouse operations. Any attempt to print or draw on a closed window will open it.

Window Properties

The behavior of a window is controlled by a set of "window properties." Some of these are used by the system. However, any arbitrary property name may be used by a user program to associate information with a window. For many applications the user will associate the structure being displayed with its window using a property. The following functions provide for reading and setting window properties:

(**WINDOWPROP** *WINDOW PROP NEWVALUE*) [NoSpread Function]

Returns the previous value of *WINDOW*'s *PROP* aspect. If *NEWVALUE* is given, (even if given as NIL), it is stored as the new *PROP* aspect. Some aspects cannot be set by the user and will generate errors. Any *PROP* name that is not recognized is stored on a property list associated with the window.

(**WINDOWADDPROP** *WINDOW PROP ITEMTOADD FIRSTFLG*) [Function]

WINDOWADDPROP adds a new item to a window property. If *ITEMTOADD* is EQ to an element of the *PROP* property of the window *WINDOW*, nothing is added. If the current property is not a list, it is made a list before *ITEMTOADD* added. WINDOWADDPROP returns the previous property. If *FIRSTFLG* is non-NIL, the new item goes on the front of the list; otherwise, it goes on the end of the list. If *FIRSTFLG* is non-NIL and *ITEMTOADD* is already on the list, it is moved to the front.

Many window properties (*OPENFN*, *CLOSEFN*, etc.) can be a list of functions. WINDOWADDPROP is useful for adding additional functions to a window property without affecting any existing functions. Note that if the order of items in a window property is important, the list can be modified using WINDOWPROP.

(**WINDOWDELPROP** *WINDOW PROP ITEMTODELETE*) [Function]

WINDOWDELPROP deletes *ITEMTODELETE* from the window property *PROP* of *WINDOW* and returns the previous list if *ITEMTODELETE* was an element. If *ITEMTODELETE* was not a member of window property *PROP*, NIL is returned.

Creating Windows

INTERLISP-D REFERENCE MANUAL

(**CREATEW** *REGION TITLE BORDERSIZE NOOPENFLG*) [Function]

Creates a new window. *REGION* indicates where and how large the window should be by specifying the exterior region of the window. The usable height and width of the resulting window will be smaller than the height and width of the region by twice the border size and further less the height of the title, if any. If *REGION* is NIL, GETREGION is called to prompt the user for a region.

If *TITLE* is non-NIL, it is printed in the border at the top of the window. The *TITLE* is printed using the global display stream WindowTitleDisplayStream. Thus the height of the title will be (FONTPROP WindowTitleDisplayStream 'HEIGHT).

If *BORDERSIZE* is a number, it is used as the border size. If *BORDERSIZE* is not a number, the window will have a border WBorder (initially 4) bits wide.

If *NOOPENFLG* is non-NIL, the window will not be opened, i.e. displayed on the screen.

The initial X and Y positions of the window are set to the upper left corner by calling MOVETOUPPERLEFT (see Chapter 27).

(**DECODE.WINDOW.ARG** *WHERESPEC WIDTH HEIGHT TITLE BORDER NOOPENFLG*) [Function]

This is a useful function for creating windows. *WHERESPEC* can be a WINDOW, a REGION, a POSITION or NIL. If *WHERESPEC* is a WINDOW, it is returned. In all other cases, CREATEW is called with the arguments *TITLE BORDER* and *NOOPENFLG*. The REGION argument to CREATEW is determined from *WHERESPEC* as follows:

If *WHERESPEC* is a REGION, it is adjusted to be on the screen, then passed to CREATEW.

If *WIDTH* and *HEIGHT* are numbers and *WHERESPEC* is a POSITION, the region whose lower left corner is *WHERESPEC*, whose width is *WIDTH* and whose height is *HEIGHT* is adjusted to be on the screen, then passed to CREATEW.

If *WIDTH* and *HEIGHT* are numbers and *WHERESPEC* is not a POSITION, then GETBOXREGION is called to prompt the user for the position of a region that is *WIDTH* by *HEIGHT*.

If *WIDTH* and *HEIGHT* are not numbers, CREATEW is given NIL as a REGION argument.

If *WIDTH* and *HEIGHT* are used, they are used as interior dimensions for the window.

(**WINDOWP** *X*) [Function]

Returns *X* if *X* is a window, NIL otherwise.

Opening and Closing Windows

(**OPENWP** *WINDOW*) [Function]

Returns *WINDOW*, if *WINDOW* is an open window (has not been closed); NIL otherwise.

WINDOWS AND MENUS

(**OPENWINDOWS**) [Function]

Returns a list of all open windows.

(**OPENW** *WINDOW*) [Function]

If *WINDOW* is a closed window, **OPENW** calls the function or functions on the window property **OPENFN** of *WINDOW*, if any. If one of the **OPENFN**s is the atom **DON'T**, the window will not be opened. Otherwise the window is placed on the occlusion stack of windows and its contents displayed on the screen. If *WINDOW* is an open window, it returns **NIL**.

(**CLOSEW** *WINDOW*) [Function]

CLOSEW calls the function or functions on the window property **CLOSEFN** of *WINDOW*, if any. If one of the **CLOSEFN**s is the atom **DON'T** or returns the atom **DON'T** as a value, **CLOSEW** returns without doing anything further. Otherwise, **CLOSEW** removes *WINDOW* from the window stack and restores the bits it is obscuring. If *WINDOW* was closed, *WINDOW* is returned as the value. If it was not closed, (for example because its **CLOSEFN** returned the atom **DON'T**), **NIL** is returned as the value.

WINDOW can be restored in the same place with the same contents (reopened) by calling **OPENW** or by using it as the source of a display operation.

OPENFN [Window Property]

The **OPENFN** window property can be a single function or a list of functions. If one of the **OPENFN**s is the atom **DON'T**, the window will not be opened. Otherwise, the **OPENFN**s are called after a window has been opened by **OPENW**, with the window as a single argument.

CLOSEFN [Window Property]

The **CLOSEFN** window property can be a single function or a list of functions that are called just before a window is closed by **CLOSEW**. The function(s) will be called with the window as a single argument. If any of the **CLOSEFN**s are the atom **DON'T**, or if the value returned by any of the **CLOSEFN**s is the atom **DON'T**, the window will not be closed.

Note: If the **CAR** of the **CLOSEFN** list is a **LAMBDA** word, it is treated as a single function.

Note: A **CLOSEFN** should not call **CLOSEW** on its argument.

Redisplaying Windows

(**REDISPLAYW** *WINDOW* *REGION* *ALWAYSFLG*) [Function]

Redisplay the region *REGION* of the window *WINDOW*. If *REGION* is **NIL**, the entire window is redisplayed.

INTERLISP-D REFERENCE MANUAL

If *WINDOW* doesn't have a *REPAINTFN*, the action depends on the value of *ALWAYSFLG*. If *ALWAYSFLG* is *NIL*, *WINDOW* will not change and the message "Window has no *REPAINTFN*. Can't redisplay." will be printed in the prompt window. If *ALWAYSFLG* is non-*NIL*, *REDISPLAYW* acts as if *REPAINTFN* was *NILL*.

REPAINTFN

[Window Property]

The *REPAINTFN* window property can be a single function or a list of functions that are called to repaint parts of the window by *REDISPLAYW*. The *REPAINTFN*s are called with two arguments: the window and the region in the coordinates of the window's display stream of the area that should be repainted. Before the *REPAINTFN* is called, the clipping region of the window is set to clip all display operations to the area of interest so that the *REPAINTFN* can display the entire window contents and the results will be appropriately clipped.

Note: *CLEARW* (see the Miscellaneous Window Functions section below) should not be used in *REPAINTFN*s because it resets the window's coordinate system. If a *REPAINTFN* wants to clear its region first, it should use *DSPFILL* (see Chapter 27).

Reshaping Windows

(**SHAPEW** *WINDOW* *NEWREGION*)

[Function]

Reshapes *WINDOW*. If the window property *RESHAPEFN* is the atom *DON'T* or a list that contains the atom *DON'T*, a message is printed in the prompt window, *WINDOW* is not changed, and *NIL* is returned. Otherwise, *RESHAPEFN* window property can be a single function or a list of functions that are called when a window is reshaped, to reformat or redisplay the window contents (see below). If the *RESHAPEFN* window property is *NIL*, *RESHAPEBYREPAINTFN* is the default.

If the region *NEWREGION* is *NIL*, it prompts for a region with *GETREGION*. When calling *GETREGION*, the function *MINIMUMWINDOWSIZE* is called to determine the minimum height and width of the window, the function *WINDOWREGION* is called to get the region passed as the *OLDREGION* argument, the window property *NEWREGIONFN* is used as the *NEWREGIONFN* argument and *WINDOW* as the *NEWREGIONFNARG* argument. If the window property *INITCORNERSFN* is non-*NIL*, it is applied to the window, and the value is passed as the *INITCORNERS* argument to *GETREGION*, to determine the initial size of the "ghost region." These window properties allow the window to specify the regions used for interactive calls to *SHAPEW*.

If the region *NEWREGION* is a *REGION* and its *WIDTH* or *HEIGHT* less than the minimums returned by calling the function *MINIMUMWINDOWSIZE*, they will be increased to the minimums.

If *WINDOW* has a window property *DOSHAPEFN*, it is called, passing it *WINDOW* and *NEWREGION* (or the region returned by *GETREGION*). If *WINDOW* does not have a *DOSHAPEFN* window property, the function *SHAPEW1* is called to reshape the window.

WINDOWS AND MENUS

DOSHAPEFNs are provided to implement window groups and few users should ever write them. They are tricky to write and must call SHAPEW1 eventually. The RESHAPEFN window property is a simpler hook into reshape operations.

(**SHAPEW1** *WINDOW REGION*)

[Function]

Changes *WINDOW*'s size and position on the screen to be *REGION*. After clearing the region on the screen, it calls the window's RESHAPEFN, if any, passing it three arguments: *WINDOW*; a bitmap that contains *WINDOW*'s previous screen image; and the region of *WINDOW*'s old image within the bitmap.

RESHAPEFN

[Window Property]

The RESHAPEFN window property can be a single function or a list of functions that are called when a window is reshaped by SHAPEW. If the RESHAPEFN is DON'T or a list containing DON'T, the window will not be reshaped. Otherwise, the function(s) are called after the window has been reshaped, its coordinate system readjusted to the new position, the title and border displayed, and the interior filled with texture. The RESHAPEFN should display any additional information needed to complete the window's image in the new position and shape. The RESHAPEFN is called with four arguments: (1) the window in its reshaped form, (2) a bitmap with the image of the old window in its old shape, and (3) the region within the bitmap that contains the window's old image, and (4) the region of the screen previously occupied by this window. This function is provided so that users can reformat window contents or whatever. RESHAPEBYREPAINTFN (below) is the default and should be useful for many windows.

NEWREGIONFN

[Window Property]

If SHAPEW calls GETREGION to prompt the user for a region, the value of the NEWREGIONFN window property is passed as the NEWREGIONFN argument to GETREGION.

INITCORNERSFN

[Window Property]

If this window property is non-NIL, it should be a function of one argument, a window, that returns a list specifying the initial corners of a "ghost region" of the form (BASEX BASEY OPPX OPPY), where (BASEX, BASEY) describes the anchored corner of the box, and (OPPX, OPPY) describes the trackable corner. If SHAPEW calls GETREGION to prompt the user for a region, this function is applied to the window, and the list returned is passed as the INITCORNERS argument to GETREGION, to specify the initial ghost region.

DOSHAPEFN

[Window Property]

If this window property is non-NIL, it is called by SHAPEW to reshape the window (instead of SHAPEW1). It is called with two arguments: the window and the new region.

(**RESHAPEBYREPAINTFN** *WINDOW OLDIMAGE IMAGEREGION OLDSCREENREGION*) [Function
]

INTERLISP-D REFERENCE MANUAL

This is the default window *RESHAPEFN*. *WINDOW* is a window that has been reshaped from the screen region *OLDSCREENREGION* to its new region (available via *(WINDOWPROP WINDOW 'REGION)*). *OLDIMAGE* is a bitmap that contains the image of the window from its previous location. *IMAGEREGION* is the region within *OLDIMAGE* that contains the old image.

RESHAPEBYREPAINTFN *BITBLT*s the old region contents into the new region. If the new shape is larger in either or both dimensions, the newly exposed areas are redisplayed via calls *WINDOW*'s *REPAINTFN* window property. *RESHAPEBYREPAINTFN* may call the *REPAINTFN* up to four times during a single reshape.

The choice of which areas of the window to remove or extend is done as follows. If *WINDOW*'s new region shares an edge with *OLDSCREENREGION*, that edge of the window image will remain fixed and any addition or reduction in that dimension will be performed on the opposite side. If *WINDOW* has an *EXTENT* property and the newly exposed window area is outside of it, any extra will be added so as to show *EXTENT* that was previously not visible. An exception to these rules is that the current X,Y position is kept visible, if it was visible before the reshape.

Moving Windows

(*MOVEW WINDOW POSorX Y*) [Function]

Moves *WINDOW* to the position specified by *POSorX* and *Y* according to the following rules:

If *POSorX* is *NIL*, *GETBOXPOSITION* is called to read a position from the user. If *WINDOW* has a *CALCULATEREGION* window property, it will be called with *WINDOW* as an argument and should return a region which will be used to prompt the user with. If *WINDOW* does not have a *CALCULATEREGION* window property, the region of *WINDOW* is used to prompt with.

If *POSorX* is a *POSITION*, *POSorX* is used.

If *POSorX* and *Y* are both *NUMBERP*, a position is created using *POSorX* as the *XCOORD* and *Y* as the *YCOORD*.

If *POSorX* is a *REGION*, a position is created using its *LEFT* as the *XCOORD* and *BOTTOM* as the *YCOORD*.

If *WINDOW* is not open and *POSorX* is non-*NIL*, the window will be moved without being opened. Otherwise, it will be opened.

If *WINDOW* has the atom *DON'T* as a *MOVEFN* window property, the window will not be moved. If *WINDOW* has any other non-*NIL* value as a *MOVEFN* property, it should be a function or list of functions that will be called before the window is moved with the *WINDOW* and the new position as its arguments. If it returns the atom *DON'T*, the window will not be moved. If it returns a position, the window will be moved to that position.

WINDOWS AND MENUS

instead of the new one. If there are more than one MOVEFNs, the last one to return a value is the one that determines where the window is moved to.

If *WINDOW* is moved and *WINDOW* has an AFTERMOVEFN window property, it should be a function or a list of functions that will be called after the window is moved with *WINDOW* as an argument.

MOVEW returns the new position, or NIL if the window could not be moved.

Note: If MOVEW moves any part of the window from off-screen onto the screen, that part is redisplayed (by calling REDISPLAYW).

(**RELMOVEW** *WINDOW* *POSITION*)

[Function]

Like MOVEW for moving windows but the *POSITION* is interpreted relative to the current position of *WINDOW*. Example: The following code moves *WINDOW* to the right one screen point.

```
(RELMOVEW WINDOW (create POSITION XCOORD ← 1 YCOORD  
← 0))
```

CALCULATEREGION

[Window Property]

If MOVEW calls GETBOXPOSITION to prompt the user for a region, the CALCULATEREGION window property is called (passing the window as an argument. The CALCULATEREGION should return a region to be used to prompt the user with. If CALCULATEREGION is NIL, the region of the window is used to prompt with.

MOVEFN

[Window Property]

If the MOVEFN is DON'T, the window will not be moved by MOVEW. Otherwise, if the MOVEFN is non-NIL, it should be a function or a list of functions that will be called before a window is moved with two arguments: the window being moved and the new position of the lower left corner in screen coordinates. If the MOVEFN returns DON'T, the window will not be moved. If the MOVEFN returns a POSITION, the window will be moved to that position. Otherwise, the window will be moved to the specified new position.

AFTERMOVEFN

[Window Property]

If non-NIL, it should be a function or a list of functions that will be called after the window is moved (by MOVEW) with the window as an argument.

Exposing and Burying Windows

(**TOTOPW** *WINDOW* *NOCALLTOTOPFNFLG*)

[Function]

Brings *WINDOW* to the top of the stack of overlapping windows, guaranteeing that it is entirely visible. If *WINDOW* is closed, it is opened. This is done automatically whenever a printing or drawing operation occurs to the window.

INTERLISP-D REFERENCE MANUAL

If `NOCALLTOTOPFNFLG` is `NIL`, the `TOTOPFN` of `WINDOW` is called. If `NOCALLTOTOPFNFLG` is `T`, it is not called, which allows a `TOTOPFN` to call `TOTOPW` without causing an infinite loop.

(**BURYW** *WINDOW*) [Function]

Puts *WINDOW* on the bottom of the stack by moving all the windows that it covers in front of it.

TOTOPFN [Window Property]

If non-`NIL`, whenever the window is brought to the top, the `TOTOPFN` is called (with the window as a single argument). This function may be used to bring a collection of windows to the top together.

If the `NOCALLTOPWFN` argument of `TOTOPW` is non-`NIL`, the `TOTOPFN` of the window is not called, which provides a way of avoiding infinite loops when using `TOTOPW` from within a `TOTOPFN`.

Shrinking Windows Into Icons

Occasionally, a user will have a number of large windows on the screen, making it difficult to access those windows being used. To help with the problem of screen space management, the Interlisp-D window system allows the creation of Icons. An icon is a small rectangle (containing text or a bitmap) which is a "shrunk-down" form of a particular window. Using the Shrink and Expand window menu commands (see the beginning of this chapter), the user can shrink windows not currently being used into icons, and quickly restore the original windows at any time. This facility is controlled by the following functions and window properties:

(**SHRINKW** *WINDOW TOWHAT ICONPOSITION EXPANDFN*) [Function]

`SHRINKW` makes a small icon which represents *WINDOW* and removes *WINDOW* from the screen. Icons have a different window command menu that contains "EXPAND" instead of "SHRINK". The `EXPAND` command calls `EXPANDW` which returns the shrunk window to its original size and place. The icon can also be moved by pressing the `LEFT` button in it, or expanded by pressing the `MIDDLE` button in it.

The `SHRINKFN` property of the window *WINDOW* affects the operation of `SHRINKW`. If the `SHRINKFN` property of *WINDOW* is the atom `DON'T`, `SHRINKW` returns. Otherwise, the `SHRINKFN` property of the window is treated as a (list of) function(s) to apply to *WINDOW*; if any returns the atom `DON'T`, `SHRINKW` returns.

TOWHAT, if given, indicates the image the icon window will have. If *TOWHAT* is a string, atom or list, the icon's image will be that string (currently implemented as a title-only window with *TOWHAT* as the title.) If *TOWHAT* is a `BITMAP`, the icon's image will be a copy of the bitmap. If *TOWHAT* is a *WINDOW*, that window will be used as the icon.

If *TOWHAT* is not given (as is the case when invoked from the `SHRINK` window command), then the following apply in turn:

WINDOWS AND MENUS

1. If the window has an `ICONFN` property, it gets called with the two arguments `WINDOW` and `OLDICON`, where `WINDOW` is the window being shrunk and `OLDICON` is the previously created icon, if any. The `ICONFN` should return one of the `TOWHAT` entities described above or return the `OLDICON` if it does not want to change it.
2. If the window has an `ICON` property, it is used as the value of `TOWHAT`.
3. If the window has neither an `ICONFN` or `ICON` property, the icon will be `WINDOW`'s title or, if `WINDOW` doesn't have a title, the date and time of the icon creation.

`ICONPOSITION` gives the position that the new icon will be on the screen. If it is `NIL`, the icon will be in the corner of the window furthest from the center of the screen.

In all but the default case, the icon is cached on the property `ICONWINDOW` of `WINDOW` so repeating `SHRINKW` reuses the same icon (unless overridden by the `ICONFN` described above). Thus to change the icon it is necessary to remove the `ICONWINDOW` property or call `SHRINKW` explicitly giving a `TOWHAT` argument.

(**EXPANDW** *ICONW*)

[Function]

Restores the window for which *ICONW* is an icon, and removes the icon from the screen. If the `EXPANDFN` window property of the main window is the atom `DON'T`, the window won't be expanded. Otherwise, the window will be restored to its original size and location and the `EXPANDFN` (or list of functions) will be applied to it.

SHRINKFN

[Window Property]

The `SHRINKFN` window property can be a single function or a list of functions that are called just before a window is shrunk by `SHRINKW`, with the window as a single argument. If any of the `SHRINKFN`s are the atom `DON'T`, or if the value returned by any of the `SHRINKFN`s is the atom `DON'T`, the window will not be shrunk.

EXPANDREGIONFN

[Window property]

`EXPANDREGIONFN`, if non-`NIL`, should be the function to be called (with the window as its argument) before the window is actually expanded.

The `EXPANDREGIONFN` must return `NIL` or a valid region, and must not do any window operations (e.g., redisplaying). If `NIL` is returned, the window is expanded normally, as if the `EXPANDREGIONFN` had not existed. The region returned specifies the new region for the main window only, not for the group including any of its attached windows. The window will be opened in its new shape, and any attached windows will be repositioned or rejustified appropriately. The main window must have a `REPAINTFN` which can repaint the entire window under these conditions.

As with expanding windows normally, the `OPENFN` for the main window is not called.

INTERLISP-D REFERENCE MANUAL

Also, the window is reshaped without checking for a special shape function (e.g., a DOSHAPEFN).

ICONFN [Window Property]

If SHRINKW is called without begin given a TOWHAT argument (as is the case when invoked from the SHRINK window command) and the window's ICONFN property is non-NIL, then it gets called with two arguments, the window being shrunk and the previously created icon, if any. The ICONFN should return one of the TOWHAT entities described above or return the previously created icon if it does not want to change it.

ICON [Window Property]

If SHRINKW is called without being given a TOWHAT argument, the window's ICONFN property is NIL, and the ICON property is non-NIL, then it is used as the value of TOWHAT.

ICONWINDOW [Window Property]

Whenever an icon is created, it is cached on the property ICONWINDOW of the window, so calling SHRINKW again will reuse the same icon (unless overridden by the ICONFN.

Thus, to change the icon it is necessary to remove the ICONWINDOW property or call SHRINKW explicitly giving a TOWHAT argument.

DEFAULTICONFN [Variable]

Changes how an icon is created when a window having no ICONFN is shrunk or when SHRINKW, with a TOWHAT argument of a string, is called. The value of DEFAULTICONFN is a function of two arguments (window text); text is either NIL or a string. DEFAULTICONFN returns an icon window.

The initial value of DEFAULTICONFN is MAKETITLEBARICON. It creates a window that is a title bar only; the title is either the text argument, the window's title, or "Icon made <date>" for titleless windows. MAKETITLEBARICON places the title bar at some corner of the main window.

An alternative behavior is available by setting DEFAULTICONFN to be TEXTICON. TEXTICON creates a titled icon window from the text or window's title.

You can now copy-select titled icons such as those used by FileBrowser, SEdit, TEdit, Sketch. The default behavior is that the icon's title is unread (via BKSYSBUF), but if the icon window has a COPYFN property, that gets called instead, with the icon window as its argument. For example, if the name displayed in an icon is really a symbol, and you want copy selection to cause the name to be unread correctly with respect to the package and read table of the exec you are copying into, you could put the following COPYFN property on the icon window:

```
(LAMBDA (WINDOW)
```

WINDOWS AND MENUS

```
(IL:BKSYSEBUF <fetch symbolic name from window> T )
```

EXPANDFN

[Window Property]

The EXPANDFN window property can be a single function or a list of functions. If one of the EXPANDFNs is the atom DON'T, the window will not be expanded. Otherwise, the EXPANDFNs are called after the window has been expanded by EXPANDW, with the window as a single argument.

Creating Icons with ICONW

ICONW is a group of functions available for building small windows of arbitrary shape. These windows are principally for use as icons for shrinking windows; i.e., these functions are likely to be invoked from within the ICONFN of a window. An icon is specified by supplying its image (a bitmap) and a mask that specifies its shape. The mask is a bitmap of the same dimensions as the image whose bits are on (black) in those positions considered to be in the image, and off (white) in those positions where the background should show through. By using the mask and appropriate window functions, ICONW maintains the illusion that the icon window is nonrectangular, even though the actual window itself is rectangular. The illusion is not complete, of course. For example, if you try to select what looks like the background (or an occluded window) around the icon but still within its rectangular perimeter, the icon window itself is selected. Also, if you move a window occluded by an icon, the icon never notices that the background changed behind it. Icons created with ICONW can also have titles; some part of the image can be filled with text computed at the time the icon is created, or text may be changed after creation.

Creating Icons

Two types of icons can be created with ICONW, a borderless window containing an image defined by a mask and a window with a title.

(**ICONW** *IMAGE MASK POSITION NOOPENFLG*)

[Function]

Creates a window at *POSITION*, or prompts for a position if *POSITION* is NIL. The window is borderless, and filled with *IMAGE*, as cookie-cut by *MASK*. If *MASK* is NIL, the image is considered rectangular (i.e., *MASK* defaults to a black bitmap of the same dimensions as *IMAGE*). If *NOOPENFLG* is T, the window is returned unopened.

(**TITLEDICONW** *ICON TITLE FONT POSITION NOOPENFLG JUST BREAKCHARS OPERATION*)

[Function]

INTERLISP-D REFERENCE MANUAL

Creates a titled icon at *POSITION*, or prompts for a position if *POSITION* is *NIL*. If *NOOPENFLG* is *T*, the window is returned unopened. The argument *ICON* is an instance of the record *TITLEDICON*, which specifies the icon image and mask, as with *ICONW*, and a region within the image to be used for displaying the title. Thus, the *ICON* argument is usually of the form

```
(create TITLEDICON ICON ← someIconImage  
MASK ← iconMask TITLEREG ← someRegionWithinICON)
```

The title region is specified in coordinates relative to the icon, i.e., the lower-left corner of the image bitmap is (0, 0). The mask can be *NIL* if the icon is rectangular. The image should be white where it is covered by the title region. *TITLEDICONW* clears the region before printing on it. The title is printed into the specified region in the image, using *FONT*. If *FONT* is *NIL* it defaults to the value of *DEFAULTICONFONT*, initially Helvetica 10. The title is broken into multiple lines if necessary; *TITLEDICONW* attempts to place the breaks at characters that are in the list of character codes *BREAKCHARS*. *BREAKCHARS* defaults to (CHARCODE (SPACE *ÿ*)). In addition, line breaks are forced by any carriage returns in *TITLE*, independent of *BREAKCHARS*. *BREAKCHARS* is ignored if a long title would not otherwise fit in the specified region. For convenience, *BREAKCHARS* = *FILE* means the title is a file name, so break at file name field delimiters. The argument *JUST* indicates how the text should be justified relative to the region. It is an atom or list of atoms chosen from *TOP*, *BOTTOM*, *LEFT*, or *RIGHT*, which indicate the vertical positioning (flush to top or bottom) and/or horizontal positioning (flush to left edge or right). If *JUST* = *NIL*, the text is centered. The argument *OPERATION* is a display stream operation indicating how the title should be printed. If *OPERATION* is *INVERT*, then the title is printed white-on-black. The default *OPERATION* is *REPLACE*, meaning black-on-white. *ERASE* is the same as *INVERT*; *PAINT* is the same as *REPLACE*.

For convenience, *TITLEDICONW* can also be used to create icons that consist solely of a title, with no special image. If the argument *ICON* is *NIL*, *TITLEDICONW* creates a rectangular icon large enough to contain *TITLE*, with a border the same width as that on a regular window. The remaining arguments are as described above, except that a *JUST* of *TOP* or *BOTTOM* is not meaningful.

In the Medley release, *TITLEDICONW* can create icons with white text on a black background. To get this effect, your icon image must be black in the correct area, and you must specify the *OPERATION* argument as *INVERT*.

In Medley, you can copy- select the title of an icon.

Modifying Icons

(*ICONW*.*TITLE* *ICON* *TITLE*)

[Function]

WINDOWS AND MENUS

Returns the current title of the window *ICON*, which must be a window returned by *TITLEDICONW*. In addition, if *TITLE* is non-NIL, makes *TITLE* the new title of the window and repaints it accordingly. To erase the current title, make *TITLE* a null string.

(**ICONW.SHADE** *WINDOW SHADE*) [Function]

Returns the current shading of the window *ICON*, which must be a window returned by *ICONW* or *TITLEDICONW*. In addition, if *SHADE* is non-NIL, paints the texture *SHADE* on *WINDOW*. A typical use for this function is to communicate a change of state in a window that is shrunk, without reopening the window. To remove any shading, make *SHADE* be *WHITESHAE*.

Default Icons

When you shrink a window that has no *ICONFN*, the system currently creates an icon that looks like the window's title bar. You can make the system instead create titled icons by setting the global variable *DEFAULTICONFN* to the value *TEXTICON*.

(**TEXTICON** *WINDOW TEXT*) [Function]

Creates a titled icon window for the main window *WINDOW* containing the text *TEXT*, or the window's title if *TEXT* is NIL.

DEFAULTTEXTICON [Variable]

The value that *TEXTICON* passes to *TITLEDICONW* as its *ICON* argument. Initially it is NIL, which creates an unadorned rectangular window. However, you can set it to a *TITLEDICON* record of your choosing if you would like default icons to have a different appearance.

Coordinate Systems, Extents, And Scrolling

Note: The word "scrolling" has two distinct meanings when applied to Interlisp-D windows. This section documents the use of "scroll bars" on the left and bottom of a window to move an object displayed in the window. "Scrolling" also describes the feature where trying to print text off the bottom of a window will cause the contents to "scroll up." This second feature is controlled by the function *DSPSCROLL* (see Chapter 27).

One way of thinking of a window is as a "view" onto an object (e.g. a graph, a file, a picture, etc.) The object has its own natural coordinate system in terms of which its subparts are laid out. When the window is created, the *X Offset* and *Y Offset* of the window's display stream are set to map the origin of the object's coordinate system into the lower left point of the window's interior region. At the same time, the *Clipping Region* of the display stream is set to correspond to the interior of the window. From then on, the display stream's coordinate system is translated and its clipping region adjusted whenever the window is moved, scrolled or reshaped.

INTERLISP-D REFERENCE MANUAL

There are several distinct regions associated with a window viewing an object. First, there is a region in the window's coordinate system that contains the complete image of the object. This region (which can only be determined by application programs with knowledge of the "semantics" of the object) is stored as the `EXTENT` property of the window (below). Second, the clipping region of the display stream (obtainable with the function `DSPCLIPPINGREGION`, see Chapter 27) specifies the portion of the object that is actually visible in the window. This is set so that it corresponds to the interior of the window (not including the border or title). Finally, there is the region on the screen that specifies the total area that the window occupies, including the border and title. This region (in screen coordinates) is stored as the `REGION` property of the window (see the Miscellaneous Window Properties section below).

The window system supports the idea of scrolling the contents of a window. Scrolling regions are on the left and the bottom edge of each window. The `LEFT` button is used to indicate upward or leftward scrolling by the amount necessary to move the selected position to the top or the left edge. The `RIGHT` button is used to indicate downward or rightward scrolling by the amount necessary to move the top or left edge to the selected position. The `MIDDLE` button is used to indicate global placement of the object within the window (similar to "thumbing" a book). In the scroll region, the part of the object that is being viewed by the window is marked with a gray shade. If the whole scroll bar is thought of as the entire object, the shaded portion is the portion currently being viewed. This will only occur when the window "knows" how big the object is (see window property `EXTENT`, below).

When the button is released in a scroll region, the function `SCROLLW` is called. `SCROLLW` calls the scrolling function associated with the window to do the actual scrolling and provides a programmable entry to the scrolling operation.

(**SCROLLW** *WINDOW DELTAX DELTAY CONTINUOUSFLG*) [Function]

Calls the `SCROLLFN` window property of the window *WINDOW* with arguments *WINDOW*, *DELTAX*, *DELTAY* and *CONTINUOUSFLG*. See `SCROLLFN` window property below.

(**SCROLL.HANDLER** *WINDOW*) [Function]

This is the function that tracks the mouse while it is in the scroll region. It is called when the cursor leaves a window in either the left or downward direction. If *NWINDOW* does not have a scroll region for this direction (e.g. the window has moved or reshaped since it was last scrolled), a scroll region is created that is `SCROLLBARWIDTH` wide. It then waits for `SCROLLWAITTIME` milliseconds and if the cursor is still inside the scroll region, it opens a window the size of the scroll region and changes the cursor to indicate the scrolling is taking place.

When a button is pressed, the cursor shape is changed to indicate the type of scrolling (up, down, left, right or thumb). After the button is held for `WAITBEFORESCROLLTIME` milliseconds, until the button is released `SCROLLW` is called each `WAITBETWEENSCROLLTIME` milliseconds. These calls are made with the `CONTINUOUSFLG` argument set to `T`. If the button is released before `WAITBEFORESCROLLTIME` milliseconds, `SCROLLW` is called with the `CONTINUOUSFLG` argument set to `NIL`.

WINDOWS AND MENUS

The arguments passed to `SCROLLW` depend on the mouse button. If the `LEFT` button is used in the vertical scroll region, `DY` is distance from cursor position at the time the button was released to the top of the window and `DX` is 0. If the `RIGHT` button is used, the inverse of this quantity is used for `DY` and 0 for `DX`. If the `LEFT` button is used in the horizontal scroll region, `DX` is distance from cursor position to left of the window and `DY` is 0. If the `RIGHT` button is used, the inverse of this quantity is used for `DX` and 0 for `DY`.

If the `MIDDLE` button is pressed, the distance argument to `SCROLLW` will be a `FLOATP` between 0.0 and 1.0 that indicates the proportion of the distance the cursor was from the left or top edge to the right or bottom edge.

Note: The scrolling regions will not come up if the window has a `SCROLLFN` window property of `NIL`, has a non-`NIL` `NOSCROLLBARS` window property, or if its `SCROLLEXTENTUSE` property has certain values and its `EXTENT` is fully visible.

(**SCROLLBYREPAINTFN** *WINDOW DELTAX DELTAY CONTINUOUSFLG*) [Function]

`SCROLLBYREPAINTFN` is the standard scrolling function which should be used as the `SCROLLFN` property for most scrolling windows.

This function, when used as a `SCROLLFN`, `BITBLT`s the bits that will remain visible after the scroll to their new location, fills the newly exposed area with texture, adjusts the window's coordinates and then calls the window's `REPAINTFN` on the newly exposed region. Thus this function will scroll any window that has a repaint function.

If *WINDOW* has an `EXTENT` property, `SCROLLBYREPAINTFN` will limit scrolling in the `X` and `Y` directions according to the value of the window property `SCROLLEXTENTUSE`.

If *DELTAX* or *DELTAY* is a `FLOATP`, `SCROLLBYREPAINTFN` will position the window so that its top or left edge will be positioned at that proportion of its `EXTENT`. If the window does not have an `EXTENT`, `SCROLLBYREPAINTFN` will do nothing.

If *CONTINUOUSFLG* is non-`NIL`, this indicates that the scrolling button is being held down. In this case, `SCROLLBYREPAINTFN` will scroll the distance of one linefeed height (as returned by `DSPLINEFEED`, see Chapter 27).

Scrolling is controlled by the following window properties:

EXTENT [Window Property]

Used to limit scrolling operations. Accesses the extent region of the window. If non-`NIL`, the `EXTENT` is a region in the window's display stream that contains the complete image of the object being viewed by the window. User programs are responsible for updating the `EXTENT`. The functions `UNIONREGIONS`, `EXTENDREGION`, etc. (see Chapter 27) are useful for computing a new extent region.

In some situations, it is useful to define an `EXTENT` that only exists in one dimension. This may be done by specifying an `EXTENT` region with a width or height of -1.

INTERLISP-D REFERENCE MANUAL

SCROLLFN handling recognizes this situation as meaning that the negative EXTENT dimension is unknown.

SCROLLFN

[Window Property]

If the SCROLLFN property is NIL, the window will not scroll. Otherwise, it should be a function of four arguments: (1) the window being scrolled, (2) the distance to scroll in the horizontal direction (positive to right, negative to left), (3) the distance to scroll in the vertical direction (positive up, negative down), and (4) a flag which is T if the scrolling button is being held down. For more information, see SCROLL.HANDLER. For most scrolling windows, the SCROLLFN function should be SCROLLBYREPAINTFN.

NOSCROLLBARS

[Window Property]

If the NOScrollBARS property is non-NIL, scroll bars will not be brought up for this window. This disables mouse-driven scrolling of a window. This window can still be scrolled using SCROLLW.

SCROLLEXTENTUSE

[Window Property]

SCROLLBYREPAINTFN uses the SCROLLEXTENTUSE window property to limit how far scrolling can go in the X and Y directions. The possible values for SCROLLEXTENTUSE and their interpretations are:

- NIL This will keep the extent region visible or near visible. It will not scroll the window so that the top of the extent is below the top of the window, the bottom of the extent is more than one point above the top of the window, the left of the extent is to the right of the window and the right of the extent is to the left of the window. The EXTENT can be scrolled to just above the window to provide a way of "hiding" the contents of a window. In this mode the extent is either in the window or just of the top of the window.
- T The extent is not used to control scrolling. The user can scroll the window to anywhere. Having the EXTENT window property does all thumb scrolling to be supported so that the user can get back to the EXTENT by thumb scrolling.
- LIMIT This will keep the extent region visible. The window is only allowed to view within the extent.
- + This will keep the extent region visible or just off in the positive direction in either X or Y (i.e., the image will be either be visible or just off to the top and /or right.)
- This will keep the extent region visible or just off in the negative direction in either X or Y (i.e., the image will be either be visible or just off to the left and /or bottom).
- + -

WINDOWS AND MENUS

- + This will keep the extent region visible or just off in the window (i.e. the image will be either be visible or just off to the left, bottom, top or right).

(XBEHAVIOR . YBEHAVIOR) If the SCROLLEXTENTUSE is a list, the CAR is interpreted as the scrolling limit in the X behavior and the CDR as the scrolling limit in the Y behavior. XBEHAVIOR and YBEHAVIOR should each be one of the atoms (NIL T LIMIT + - +- -+). The interpretations of the atoms is the same as above except that NIL is equivalent to LIMIT.

Note: The NIL value of SCROLLEXTENTUSE is equivalent to (LIMIT . +).

Example: If the SCROLLEXTENTUSE window property of a window (with an extent defined) is (LIMIT . T), the window will scroll uncontrolled in the Y dimension but be limited to the extent region in the X dimension.

Mouse Activity in Windows

The following window properties allow the user to control the response to mouse activity in a window. The value of these properties, if non-NIL, should be a function that will be called (with the window as argument) when the specified event occurs.

These functions should be "self-contained", communicating with the outside world solely via their window argument, e.g., by setting window properties. In particular, these functions should not expect to access variables bound on the stack, as the stack context is formally undefined at the time these functions are called. Since the functions are invoked asynchronously, they perform any terminal input/output operations from their own window.

WINDOWENTRYFN

[Window Property]

Whenever a button goes down in the window and the process associated with the window is not the tty process, the WINDOWENTRYFN is called. The default is GIVE.TTY.PROCESS which gives the process associated with the window the tty and calls the BUTTONEVENTFN. WINDOWENTRYFN can be a list of functions and all will be called.

CURSORINFN

[Window Property]

Whenever the mouse moves into the window, the CURSORINFN is called. If CURSORINFN is a list of functions, all will be called.

CURSOROUTFN

[Window Property]

The CURSOROUTFN is called when the cursor leaves the window. If CURSOROUTFN is a list of functions, all will be called.

CURSORMOVEDFN

[Window Property]

INTERLISP-D REFERENCE MANUAL

The `CURSORMOVEDFN` is called whenever the cursor has moved and is inside the window. `CURSORMOVEDFN` can be a list of functions and all will be called. This allows a window function to implement "active" regions within itself by having its `CURSORMOVEDFN` determine if the cursor is in a region of interest, and if so, perform some action.

BUTTONEVENTFN

[Window Property]

The `BUTTONEVENTFN` is called whenever there is a change in the state (up or down) of the mouse buttons inside the window. Changes to the mouse state while the `BUTTONEVENTFN` is running will not be interpreted as new button events, and the `BUTTONEVENTFN` will not be re-invoked.

RIGHTBUTTONFN

[Window Property]

The `RIGHTBUTTONFN` is called in lieu of the standard window menu operation (`DOWINDOWCOM`) when the `RIGHT` button is depressed in a window. More specifically, the `RIGHTBUTTONFN` is called instead of the `BUTTONEVENTFN` when (`MOUSESTATE (ONLY RIGHT)`). If the `RIGHT` button is to be treated like any other key in a window, supply `RIGHTBUTTONFN` and `BUTTONEVENTFN` with the same function.

When an application program defines its own `RIGHTBUTTONFN`, there is a convention that the default `RIGHTBUTTONFN`, `DOWINDOWCOM`, may be executed by pressing the `RIGHT` button when the cursor is in the header or border of a window. User `RIGHTBUTTONFN`s are encouraged to follow this convention, by calling `DOWINDOWCOM` if the cursor is not in the interior region of the window.

BACKGROUND BUTTONEVENTFN

[Variable]

BACKGROUND CURSORINFN

[Variable]

BACKGROUND CURSOROUTFN

[Variable]

BACKGROUND CURSORMOVEDFN

[Variable]

These variables provide a way of taking action when there is cursor action and the cursor is in the background. They are interpreted like the corresponding window properties. If set to the name of a function, that function will be called, respectively, whenever the cursor is in the background and a button changes, when the cursor moves into the background from a window, when the cursor moved from the background into a window and when the cursor moves from one place in the background to another.

Terminal I/O and Page Holding

Each process has its own terminal i/o stream (accessed as the stream `T`, see Chapter 25). The terminal i/o stream for the current process can be changed to point to a window by using the function `TTYDISPLAYSTREAM`, so that output and echoing of type-in is directed to a window.

(`TTYDISPLAYSTREAM` *DISPLAYSTREAM*)

[Function]

Selects the display stream or window *DISPLAYSTREAM* to be the terminal output channel, and returns the previous terminal output display stream. `TTYDISPLAYSTREAM` puts

WINDOWS AND MENUS

DISPLAYSTREAM into scrolling mode and calls *PAGEHEIGHT* with the number of lines that will fit into *DISPLAYSTREAM* given its current Font and Clipping Region. The line length of *TTYDISPLAYSTREAM* is computed (like any other display stream) from its Left Margin, Right Margin, and Font. If one of these fields is changed, its line length is recalculated. If one of the fields used to compute the number of lines (such as the Clipping Region or Font) changes, *PAGEHEIGHT* is not automatically recomputed. (*TTYDISPLAYSTREAM* (*TTYDISPLAYSTREAM*)) will cause it to be recomputed.

If the window system is active, the line buffer is saved in the old TTY window, and the line buffer is set to the one saved in the window of the new display stream, or to a newly created line buffer (if it does not have one). Caution: It is possible to move the *TTYDISPLAYSTREAM* to a nonvisible display stream or to a window whose current position is not in its clipping region.

(**PAGEHEIGHT** *N*)

[Function]

If *N* is greater than 0, it is the number of lines of output that will be printed to *TTYDISPLAYSTREAM* before the page is held. A page is held before the *N*+1 line is printed to *TTYDISPLAYSTREAM* without intervening input if there is no terminal input waiting to be read. The output is held with the screen video reversed until a character is typed. Output holding is disabled if *N* is 0. *PAGEHEIGHT* returns the previous setting.

PAGEFULLFN

[Window Property]

If the *PAGEFULLFN* window property is non-NIL, it will be called with the window as a single argument when the window is full (i.e., when enough has been printed since the last TTY interaction so that the next character printed will cause information to be scrolled off the top of the window.)

If the *PAGEFULLFN* window property is NIL, the system function *PAGEFULLFN* is called. *PAGEFULLFN* simply returns if there are characters in the type-in buffer for *WINDOW*, otherwise it inverts the window and waits for the user to type a character. *PAGEFULLFN* is user advisable.

Note: The *PAGEFULLFN* window property is only called on windows which are the *TTYDISPLAYSTREAM* of some process.

TTY Process and the Caret

At any time, one process is designated as the TTY process, which is used for accepting keyboard input. The TTY process can be changed to a given process by calling *GIVE.TTY.PROCESS* (see Chapter 23), or by clicking the mouse in a window associated with the process. The latter mechanism is implemented with the following window property:

PROCESS

[Window Property]

If the *PROCESS* window property is non-NIL, it should be a *PROCESS* and will be made the TTY process by *GIVE.TTY.PROCESS* (see Chapter 23), the default

INTERLISP-D REFERENCE MANUAL

WINDOWENTRYFN property (see above). This implements the mechanism by which the keyboard is associated with different processes.

The window system uses a flashing caret  to indicate the position of the next window typeout. There is only one caret visible at any one time. The caret in the current TTY process is always visible; if it is hidden by another window, its window is brought to the top. An exception to this rule is that the flashing caret's window is not brought to the top if the user is buttoning or has a shift key down. This prevents the destination window (which has the tty and caret flashing) from interfering with the window one is trying to select text to copy from.

(**CARET** *NEWCARET*) [Function]

Sets the shape that blinks at the location of the next output to the current process. *NEWCARET* should be one of the following:

- a **CURS**OR object If *NEWCARET* is a **CURS**OR object (see Chapter 30), it is used to give the new caret shape
- OFF** Turns the caret off
- NIL** The caret is not changed. **CARET** returns a **CURS**OR representing the current caret
- T** Reset the caret to the value of **DEFAULTCARET**. **DEFAULTCARET** can be set to change the initial caret for new processes.

The hotspot of *NEWCARET* indicates which point in the new caret bitmap should be located at the current output position. The previous caret is returned. Note: the bitmap for the caret is not limited to the dimensions **CURS**OR**WIDTH** by **CURS**OR**HEIGHT**.

(**CARETRATE** *ONRATE OFFRATE*) [Function]

Sets the rate at which the caret for the current process will flash. The caret will be visible for *ONRATE* milliseconds, then not visible for *OFFRATE* milliseconds. If *OFFRATE* is **NIL** then it is set to be the same as *ONRATE*. If *ONRATE* is **T**, both the "on" and "off" times are set to the value of the variable **DEFAULTCARETRATE** (initially 333). The previous value of **CARETRATE** is returned. If the caret is off, **CARETRATE** return **NIL**.

Miscellaneous Window Functions

(**CLEARW** *WINDOW*) [Function]

Fills *WINDOW* with its background texture, changes its coordinate system so that the origin is the lower left corner of the window, sets its X position to the left margin and sets its Y position to the base line of the uppermost line of text, ie. the top of the window less the font ascent.

(**INVERTW** *WINDOW SHADE*) [Function]

WINDOWS AND MENUS

Fills the window *WINDOW* with the texture *SHADE* in INVERT mode. If *SHADE* is NIL, BLACKSHADE is used. INVERTW returns *WINDOW* so that it can be used inside RESETFORM.

(**FLASHWINDOW** *WIN? N FLASHINTERVAL SHADE*) [Function]

Flashes the window *WIN?* by "inverting" it twice. *N* is the number of times to flash the window (default is 1). *FLASHINTERVAL* is the length of time in milliseconds to wait between flashes (default is 200). *SHADE* is the shade that will be used to invert the window (default is BLACKSHADE).

If *WIN?* is NIL, the whole screen is flashed. In this case, the *SHADE* argument is ignored (can only invert the screen).

(**WHICHW** *X Y*) [Function]

Returns the window which contains the position in screen coordinates of *X* if *X* is a POSITION , the position (*X*, *Y*) if *X* and *Y* are numbers, or the position of the cursor if *X* is NIL. Returns NIL if the coordinates are not in any window. If they are in more than one window, it returns the uppermost.

Example: (WHICHW) returns the window that the cursor is in.

(**DECODE/WINDOW/OR/DISPLAYSTREAM** *DSORW WINDOWVAR TITLE BORDER*) [Function]

Returns a display stream as determined by the *DSORW* and *WINDOWVAR* arguments. If *DSORW* is a display stream, it is returned. If *DSORW* is a window, its display stream is returned. If *DSORW* is NIL, the litatom *WINDOWVAR* is evaluated. If its value is a window, its display stream is returned. If its value is not a window, *WINDOWVAR* is set to a newly created window (prompting user for region) whose display stream is then returned. If *DSORW* is NEW, the display stream of a newly created window is returned. If a *window* is involved in the decoding, it is opened and if *TITLE* or *BORDER* are given, the *TITLE* or *BORDER* property of the window are reset. The *DSORW* = NIL case is most useful for programs that want to display their output in a window, but want to reuse the same window each time they are called. The non-NIL cases are good for decoding a display stream argument passed to a function.

(**WIDTHIFWINDOW** *INTERIORWIDTH BORDER*) [Function]

Returns the width of the window necessary to have *INTERIORWIDTH* points in its interior if the width of the border is *BORDER*. If *BORDER* is NIL, the default border size WBorder is used.

(**HEIGHTIFWINDOW** *INTERIORHEIGHT TITLEFLG BORDER*) [Function]

Returns the height of the window necessary to have *INTERIORHEIGHT* points in its interior with a border of *BORDER* and, if *TITLEFLG* is non-NIL, a title. If *BORDER* is NIL, the default border size WBorder is used.

INTERLISP-D REFERENCE MANUAL

WIDTHIFWINDOW and HEIGHTIFWINDOW are useful for calculating the width and height for a call to GETBOXPOSITION for the purpose of positioning a prospective window.

(MINIMUMWINDOWSIZE WINDOW) [Function]

Returns a dotted pair, the CAR of which is the minimum width *WINDOW* needs and the CDR of which is the minimum height *WINDOW* needs.

The minimum size is determined by the value of the window property MINSIZE of *WINDOW*. If the value of the MINSIZE window property is NIL, the width is 26 and the height is the height *WINDOW* needs to have its title, border and one line of text visible. If MINSIZE is a dotted pair, it is returned. If it is a litatom, it should be a function which is called with *WINDOW* as its first argument, which should return a dotted pair.

Miscellaneous Window Properties

TITLE [Window Property]

Accesses the title of the window. If a title is added to a window whose title is NIL or the title is removed (set to NIL) from a window with a title, the window's exterior (its region on the screen) is enlarged or reduced to accomodate the change without changing the window's interior. For example, (WINDOWPROP WINDOW 'TITLE "Results") changes the title of WINDOW to be "Results". (WINDOWPROP WINDOW 'TITLE NIL) removes the title of WINDOW.

BORDER [Window Property]

Accesses the width of the border of the window. The border will have at most 2 point of white (but never more than half) and the rest black. The default border is the value of the global variable WBorder (initially 4).

WINDOWTITLESHAD [Window Property]

Accesses the window title shade of the window. If non-NIL, it should be a texture which is used as the "background texture" for the title bar on the top of the window. If it is NIL, the value of the global variable WINDOWTITLESHAD (initially BLACKSHAD) is used. Note that black is always used as the background of the title printed in the title bar, so that the letters can be read. The remaining space is painted with the "title shade".

HARDCOPYFN [Window Property]

If non-NIL, it should be a function that is called by the window menu command Hardcopy to print the contents of a window. The HARDCOPYFN property is called with two arguments, the window and an image stream to print to. If the window does not have a HARDCOPYFN, the bitmap image of the window (including the border and title) are printed on the file or printer.

DSP [Window Property]

WINDOWS AND MENUS

Value is the display stream of the window. All system functions will operate on either the window or its display stream. This window property cannot be changed using WINDOWPROP.

HEIGHT
WIDTH

[Window Property]
[Window Property]

Value is the height and width of the interior of the window (the usable space not counting the border and title). These window properties cannot be changed using WINDOWPROP.

REGION

[Window Property]

Value is a region (in screen coordinates) indicating where the window (counting the border and title) is located on the screen. This window property cannot be changed using WINDOWPROP.

Example: A Scrollable Window

The following is a simple example showing how one might create a scrollable window.

CREATE.PPWINDOW creates a window that displays the pretty printed expression EXPR. The window properties PPEXPR, PPORIGX, and PPORIGY are used for saving this expression, and the initial window position. Using this information, REPAINT.PPWINDOW simply reinitializes the window position, and prettyprints the expression again. Note that the whole expression is reformatted every time, even if only a small part actually lies within the window. If this window was going to be used to display very large structures, it would be desirable to implement a more sophisticated REPAINTFN that only redisplay that part of the expression within the window. However, this scheme would be satisfactory if most of the items to be displayed are small.

RESHAPE.PPWINDOW resets the window (and stores the initial window position), calls REPAINT.PPWINDOW to display the window's expression, and then sets the EXTENT property of the window so that SCROLLBYREPAINTFN will be able to handle scrolling and "thumbing" correctly.

```
(DEFINEQ
  (CREATE.PPWINDOW
    [LAMBDA (EXPR)
      (* rrb " 4-OCT-82 12:06" )
      (* creates a window that displays
        a pretty printed expression.)

    (PROG (WINDOW)
      (* ask the user for a piece of the
        screen and make it into a window.)
      (SETQ WINDOW (CREATEW NIL "PP window"))
      (* put the expression on the
        property list of the window so that
        the repaint and reshape functions
        can access it.)
      (WINDOWPROP WINDOW (QUOTE PPEXPR) EXPR)
      (* set the repaint and reshape
        functions.)
```

INTERLISP-D REFERENCE MANUAL

```

(WINDOWPROP WINDOW (QUOTE REPAINTFN)
 (FUNCTION REPAINT.PPWINDOW))
(WINDOWPROP WINDOW (QUOTE RESHAPEFN)
 (FUNCTION RESHAPE.PPWINDOW))
      (* make the scroll function
        SCROLLBYREPAINTFN, a system
        function that uses the repaint
        function to do scrolling.)
(WINDOWPROP WINDOW (QUOTE SCROLLFN)
 (FUNCTION SCROLLBYREPAINTFN))
      (* call the reshape function to
        initially print the expression and
        calculate its extent.)
(RESHAPE.PPWINDOW WINDOW)
(RETURN WINDOW))

(REPAINT.PPWINDOW
 [LAMBDA (WINDOW REGION)      (* rrb " 4-OCT-82 11:52")

  (* the repainting function for a window with a
    pretty printed expression. This repainting
    function ignores the region to be repainted
    and repaints the entire window.)

    (* set the window position to the
      beginning of the pretty printing
      of the expression.)
    (MOVETO (WINDOWPROP WINDOW (QUOTE PPORIGX))
      (WINDOWPROP WINDOW (QUOTE PPORIGY))
      WINDOW)
    (PRINTDEF (WINDOWPROP WINDOW (QUOTE PPEXPR))
      0 NIL NIL NIL WINDOW])

  (RESHAPE.PPWINDOW
    [LAMBDA (WINDOW)      (* rrb " 4-OCT-82 12:01")
      (* the reshape function for a
        window with a pretty printed
        expression.)

      (PROG (BTM)

        (* set the position of the window so that the
          first character appears in the upper left corner
          and save the X and Y for the repaint function.)

        (DSPRESET WINDOW)
        (WINDOWPROP WINDOW (QUOTE PPORIGX)
          (DSPXPOSITION NIL WINDOW))
        (WINDOWPROP WINDOW (QUOTE PPORIGY)
          (DSPYPOSITION NIL WINDOW))

        (* call the repaint function to
          pretty print the expression in
```

WINDOWS AND MENUS

```
                                the newly cleared window.)
(REPAINT.PPWINDOW WINDOW)

(* save the region actually covered by the pretty
   printed expression so that the scrolling routines
   will know where to stop. The pretty printing of
   the expression does a carriage return after the
   last piece of the expression printed so that the
   current position is the base line of the next line
   of text. Hence the last visible piece of the
   expression (BTM) is the ending position plus the
   height of the font above the base line (its ASCENT).)

(WINDOWPROP WINDOW (QUOTE EXTENT)
  create REGION
    LEFT ← 0
    BOTTOM ← [SETQ BTM (IPLUS
                      (DSPYPOSITION NIL WINDOW)
                      (FONTPROP WINDOW (QUOTE ASCENT))
    WIDTH ← (WINDOWPROP WINDOW (QUOTE WIDTH))
    HEIGHT ← (IDIFFERENCE
              (WINDOWPROP WINDOW (QUOTE HEIGHT))
              BTM)] )
)
```

Menus

A menu is basically a means of selecting from a list of items. The system provides common layout and interactive user selection mechanisms, then calls a user-supplied function when a selection has been confirmed. The two major constituents of a menu are a list of items and a "when selected function." The label that appears for each item is the item itself for non-lists, or its CAR if the item is a list. In addition, there are a multitude of different formatting parameters for specifying font, size, and layout. When a menu is created, its unspecified fields are filled with defaults and its screen image is computed and saved.

Menus can be either pop up or fixed. If fixed menus are used, the menu must be included in a window.

(**MENU** MENU POSITION RELEASECONTROLFLG -)

[Function]

This function provides menus that pop up when they are used. It displays *MENU* at *POSITION* (in screen coordinates) and waits for the user to select an item with a mouse key. Before any mouse key is pressed, the item the mouse is over is boxed. After any key is down, the selected menu item is video reversed. When all keys are released, *MENU*'s *WHENSELECTEDFN* field is called with four arguments: (1) the item selected, (2) the menu, (3) the last mouse key released (*LEFT*, *MIDDLE*, or *RIGHT*), and (4) the reverse list of superitems rolled through when selecting the item and *MENU* returns its

INTERLISP-D REFERENCE MANUAL

value. If no item is selected, *MENU* returns NIL. If *POSITION* is NIL, the menu is brought up at the value from *MENU*'s MENUPOSITION field, if it is a *POSITION*, or at the current cursor position. The orientation of *MENU* with respect to the specified position is determined by its MENUOFFSET field.

If *RELEASECONTROLFLG* is NIL, this process will retain control of the mouse. In this case, if the user lets the mouse key up outside of the menu, *MENU* return NIL. (Note: this is the standard way of allowing the user to indicate that they do not want to make the offered choice.) If *RELEASECONTROLFLG* is non-NIL, this process will give up control of the mouse when it is outside of the menu so that other processes can be run. In this case, clicking outside the menu has no effect on the call to *MENU*. If the menu is closed (for example, by right buttoning in it and selecting "Close" from the window menu), *MENU* returns NIL. Programmers are encouraged to provide a menu item such as "cancel" or "abort" which gives users a positive way of indicating "no choice".

Note: A "released" menu will stay visible (on top of the window stack) until it is closed or an item is selected.

(**ADDMENU** *MENU WINDOW POSITION DONTOPENFLG*)

[Function]

This function provides menus that remain active in windows. **ADDMENU** displays *MENU* at *POSITION* (in window coordinates) in *WINDOW*. If the window is too small to display the entire menu, the window is made scrollable. When an item is selected, the value of the WHENSELECTEDFN field of *MENU* is called with three arguments: (1) the item selected, (2) the menu, and (3) the mouse button that the item was selected with (LEFT, MIDDLE, or RIGHT). More than one menu can be put in a window, but a menu can only be added to one window at a time. **ADDMENU** returns the window into which *MENU* is placed.

If *WINDOW* is NIL, a window is created at the position specified by *POSITION* (in screen coordinates) that is the size of *MENU*. If a window is created, it will be opened unless *DONTOPENFLG* is non-NIL. If *POSITION* is NIL, the menu is brought up at the value of *MENU*'s MENUPOSITION field (in window coordinates), if it is a position, or else in the lower left corner of *WINDOW*. If both *WINDOW* and *POSITION* are NIL, a window is created at the current cursor position.

Warning: **ADDMENU** resets several of the window properties of *WINDOW*. The CURSORINFN, CURSORMOVEDFN, and BUTTONEVENTFN window properties are replaced with MENUBUTTONFN, so that *MENU* will be active. MENUREPAINTFN is added to the REPAINTFN window property to update the menu image if the window is redisplayed. The SCROLLFN window property is changed to SCROLLBYREPAINTFN if the window is too small for the menu, to make the window scroll.

(**DELETEMENU** *MENU CLOSEFLG FROMWINDOW*)

[Function]

This function removes *MENU* from the window *FROMWINDOW*. If *MENU* is the only menu in the window and *CLOSEFLG* is non-NIL, its window will be closed (by **CLOSEW**).

If *FROMWINDOW* is NIL, the list of currently open windows is searched for one that contains *MENU*. If none is found, *DELETEMENU* does nothing.

Menu Fields

A menu is a datatype with the following fields:

ITEMS

[Menu Field]

The list of items to appear in the menu. If an item is a list, its CAR will appear in the menu. If the item (or its CAR) is a bitmap, the bitmap will be displayed in the menu. The default selection functions interpret each item as a list of three elements: a label, a form whose value is returned upon selection, and a help string that is printed in the prompt window when the user presses a mouse key with the cursor pointing to this item. The default subitem function interprets the fourth element of the list. If it is a list whose CAR is the litatom SUBITEMS, the CDR is taken as a list of subitems.

SUBITEMFN

[Menu Field]

A function to be called to determine if an item has any subitems. If an item has subitems and the user rolls the cursor out the right of that item, a submenu with that item's subitems in it pops up. If the user selects one of the items from the submenu, the selected subitem is handled as if it were selected from the main menu. If the user rolls out of the submenu to the left, the submenu is taken down and selection resumes from the main menu.

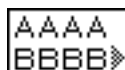
An item with subitems is marked in the menu by a grey, right pointing triangle following the label.

The function is called with two arguments: (1) the menu and (2) the item. It should return a list of the subitems of this item if any. (It is called twice to compute the menu image and each time the user rolls out of the item box so it should be moderately efficient. The default SUBITEMFN, DEFAULTSUBITEMFN, checks to see if the item is a list whose fourth element is a list whose CAR is the litatom SUBITEMS and if so, returns the CDR of it.

For example:

```
(create MENU
  ITEMS ← '(AAAA (BBBB 'BBBB "help string for
  BBBB"
              (SUBITEMS BBBB1 BBBB2 BBBB3))) )
```

will create a menu with items A and B in which B will have subitems B1, B2 and B3. The following picture below shows this menu as it first appears:



INTERLISP-D REFERENCE MANUAL

The following picture shows the submenu, with the item BBBB3 selected by the cursor



WHENSELECTEDFN

[Menu Field]

A function to be called when an item is selected. The function is called with three arguments: (1) the item selected, (2) the menu, and (3) the mouse key that the item was selected with (LEFT, MIDDLE, or RIGHT). The default function DEFAULTWHENSELECTEDFN evaluates and returns the value of the second element of the item if the item is a list of at least length 2. If the item is not a list of at least length 2, DEFAULTWHENSELECTEDFN returns the item.

Note: If the menu is added to a window with ADDMENU, the default WHENSELECTEDFN is BACKGROUNDWHENSELECTEDFN, which is the same as DEFAULTWHENSELECTEDFN except that EVAL.AS.PROCESS is used to evaluate the second element of the item, instead of tying up the mouse process.

WHENHELDNFN

[Menu Field]

The function which is called when the user has held a mouse key on an item for MENUHELDWAIT milliseconds (initially 1200). The function is called with three arguments: (1) the item selected, (2) the menu, and (3) the mouse key that the item was selected with (LEFT, MIDDLE, or RIGHT). WHENHELDNFN is intended for prompting users. The default is DEFAULTMENUHELDNFN which prints (in the prompt window) the third element of the item or, if there is not a third element, the string "This item will be selected when the button is released."

WHENUNHELDNFN

[Menu Field]

If WHENHELDNFN was called, WHENUNHELDNFN will be called: (1) when the cursor leaves the item, (2) when a mouse key is released, or (3) when another key is pressed. The function is called with the same three argument values used to call WHENHELDNFN. The default WHENUNHELDNFN is the function CLRPROMPT, which just clears the prompt window.

MENUPOSITION

[Menu Field]

The position of the menu to be used if the call to MENU or ADDMENU does not specify a position. For popup menus, this is in screen coordinates. For fixed menus, it is in the coordinates of the window the menu is in. The point within the menu image that is placed at this position is determined by MENUOFFSET. If MENUPOSITION is NIL, the menu will be brought up at the cursor position.

MENUOFFSET

[Menu Field]

WINDOWS AND MENUS

The position in the menu image that is to be located at MENUPOSITION. The default offset is (0,0). For example, to bring up a menu with the cursor over a particular menu item, set its MENUOFFSET to a position within that item and set its MENUPOSITION to NIL.

MENUFONT [Menu Field]

The font in which the items will be appear in the menu. Default is the value of MENUFONT.

TITLE [Menu Field]

If non-NIL, the value of this field will appear as a title in a line above the menu.

MENUTITLEFONT [Menu Field]

The font in which the title of the menu will be appear. If this is NIL, the title will be in the same font as window titles. If it is T, it will be in the same font as the menu items.

CENTERFLG [Menu Field]

If non-NIL, the menu items are centered; otherwise they are left-justified.

MENUROWS [Menu Field]

MENUCOLUMNS [Menu Field]

These fields control the shape of the menu in terms of rows and columns. If MENUROWS is given, the menu will have that number of rows. If MENUCOLUMNS is given, the menu will have that number of columns. If only one is given, the other one will be calculated to generate the minimal rectangular menu. (Normally only one of MENUROWS or MENUCOLUMNS is given.) If neither is given, the items will be in one column.

ITEMHEIGHT [Menu Field]

The height of each item box in the menu. If not specified, it will be the maximum of the height of the MENUFONT and the heights of any bitmaps appearing as labels.

ITEMWIDTH [Menu Field]

The width of each item box in the menu. If not specified, it will be the width of the largest item in the menu.

MENUBORDERSIZE [Menu Field]

The size of the border around each item box. If not specified, 0 (no border) is used.

MENUOUTLINESIZE [Menu Field]

The size of the outline around the entire menu. If not specified, a maximum of 1 and the MENUBORDERSIZE is used.

CHANGEOFFSETFLG [Menu Field]

INTERLISP-D REFERENCE MANUAL

(popup menus only) If `CHANGEOFFSETFLG` is non-NIL, the position of the menu offset is set each time a selection is confirmed so that the menu will come up next time in the same position relative to the cursor. This will cause the menu to reappear in the same place on the screen if the cursor has not moved since the last selection. This is implemented by changing the `MENUOFFSET` field on each use. If `CHANGEOFFSETFLG` is the atom `X` or the atom `Y`, only the `X` or the `Y` coordinate of the `MENUOFFSET` field will be changed. For example, by setting the `MENUOFFSET` position to `(-1,0)` and setting `CHANGEOFFSETFLG` to `Y`, the menu will pop up so that the cursor is just to the left of the last item selected. This is the setting of the window command menus.

The following fields are read only.

IMAGEHEIGHT [Menu Field]

Returns the height of the entire menu.

IMAGEWIDTH [Menu Field]

Returns the width of the entire menu.

Miscellaneous Menu Functions

(**MAXMENUITEMWIDTH** *MENU*) [Function]

Returns the width of the largest menu item label in the menu *MENU*.

(**MAXMENUITEMHEIGHT** *MENU*) [Function]

Returns the height of the largest menu item label in the menu *MENU*.

(**MENUREGION** *MENU*) [Function]

Returns the region covered by the image of *MENU* in its window.

(**WFROMMENU** *MENU*) [Function]

Returns the window *MENU* is located in, if it is in one; NIL otherwise.

(**DOSELECTEDITEM** *MENU ITEM BUTTON*) [Function]

Calls *MENU*'s `WHENSELECTEDFN` on *ITEM* and *BUTTON*. It provides a programmatic way of making a selection. It does not change the display.

(**MENUITEMREGION** *ITEM MENU*) [Function]

Returns the region occupied by *ITEM* in *MENU*.

(**SHADEITEM** *ITEM MENU SHADE DS/W*) [Function]

Shades the region occupied by *ITEM* in *MENU*. If *DS/W* is a display stream or a window, it is assumed to be where *MENU* is displayed. Otherwise, `WFROMMENU` is called to locate the

WINDOWS AND MENUS

window *MENU* is in. Shading is persistent, and is reapplied when the window the menu is in gets redisplayed. To unshade an item, call with a *SHADE* of 0.

(**PUTMENUPROP** *MENU* *PROPERTY* *VALUE*) [Function]

Stores the property *PROPERTY* with the value *VALUE* on a property list in the menu *MENU*. The user can use this property list for associating arbitrary data with a menu object.

(**GETMENUPROP** *MENU* *PROPERTY*) [Function]

Returns the value of the *PROPERTY* property of the menu *MENU*.

Examples of Menu Use

Example: A simple menu:

```
(MENU (create MENU ITEMS _ ' ((YES T) (NO (QUOTE
NIL))) ) )
```

Creates a menu with items YES and NO in a single vertical column:



If YES is selected, T will be returned. Otherwise, NIL will be returned.

Example: A simple menu, with centering:

```
(MENU (create MENU TITLE ← "Foo?"
ITEMS ← ' ((YES T "Adds the Foo feature.")
(NO 'NO "Removes the Foo feature.")
CENTERFLG ← T))
```

Creates a menu with a title Foo? and items YES and NO centered in a single vertical column:



The strings following the YES and NO are help strings and will be printed if the cursor remains over one of the items for a period of time. This menu differs from the one above in that it distinguishes the NO case from the case where the user clicked outside of the menu. If the user clicks outside of the menu, NIL is returned.

Example: A multi-column menu:

```
(create MENU ITEMS ← ' (1 2 3 4 5 6 7 8 9 * 0 #)
CENTERFLG ← T
```

INTERLISP-D REFERENCE MANUAL

```
MENUCOLUMNS ← 3
MENUFONT ← (FONTCREATE 'MODERN 10 'BOLD)
ITEMHEIGHT ← 15
ITEMWIDTH ← 15
CHANGEOFFSETFLG ← T)
```

Creates a touch-tone-phone number pad with the items in 15 by 15 boxes printed in Modern 10 bold font:

1	2	3
4	5	6
7	8	9
*	0	#

If used in pop up mode, its first use will have the cursor in the middle. Subsequent use will have the cursor in the same relative location as the previous selection.

Example: A program using a previously-saved menu:

```
(SELECTQ [MENU
  (COND ((type? MENU FOOMENU)
    (* use previously computed menu.)
    FOOMENU)
    (T (* create and save the menu)
      (SETQ FOOMENU
        (create MENU
          ITEMS ← '( (A 'A-SELECTED "prompt string
for A")
                    (B 'B-SELECTED "prompt string for B")
                    (A-SELECTED (* if A is selected) (DOATHING) )
                    (B-SELECTED (* if B is selected) (DOBTHING) )
                    (PROGN (* user selected outside the menu) NIL) ) )
          (B 'B-SELECTED "prompt string for B"]
        (A-SELECTED (* if A is selected) (DOATHING) )
        (B-SELECTED (* if B is selected) (DOBTHING) )
        (PROGN (* user selected outside the menu) NIL) ) )
```

This expression displays a pop up menu with two items, A and B, and waits for the user to select one. If A is selected, DOATHING is called. If B is selected, DOBTHING is called. If neither of these is selected, the form returns NIL.

The purpose of this example is to show some good practices to follow when using menus. First, the menu is only created once, and saved in the variable FOOMENU. This is more efficient if the menu is used more than once. Second, all of the information about the menu is kept in one place, which makes it easy to understand and edit. Third, the forms evaluated as a result of selecting something from the menu are part of the code and hence will be known to masterscope (as opposed to the situation if the forms were stored as part of the items). Fourth, the items in the menu have help strings for the user. Finally, the code is commented (always worth the trouble).

Free Menus

Free Menus are powerful and flexible menus that are useful for applications needing menus with different types of items, including command items, state items, and items that can be edited. A Free Menu is part of a window. It can be opened and closed as desired, or attached as a control menu to the application window.

Making a Free Menu

A Free Menu is built from a description of the contents and layout of the menu. As a Free Menu is simply a group of items, a Free Menu Description is simply a specification of a group of items. Each group has properties associated with it, as does each Free Menu Item. These properties specify the format of the items in the group, and the behavior of each item. The function `FREEMENU` takes a Free Menu Description, and returns a closed window with the Free Menu in it.

The easiest way to make a Free Menu is to define a specific function which calls `FREEMENU` with the Free Menu Description in the function. This function can then also set up the Free Menu window as required by the application. The Free Menu Description is saved as part of the specific function when the application is saved. Alternately, the Free Menu Description can be saved as a variable in your file; then just call `FREEMENU` with the name of the variable. This may be a more difficult alternative if the backquote facility is used to build the Free Menu Description.

Free Menu Formatting

A Free Menu can be formatted in one of four ways. The items in any group can be automatically laid out in rows, in columns, or in a table, or else the application can specify the exact location of each item in the group. Free Menu keeps track of the region that a group of items occupies, and items can be justified within that region. This way an item can be automatically positioned at one of the nine justification locations, top-left, top-center, top-right, middle-left, etc.

Free Menu Description

A Free Menu Description, specifying a group of items, is a list structure. The first entry in the list is an optional list of the properties for this group of items. This entry is in the form:

```
(PROPS  <PROP> <VALUE> <PROP> <VALUE> ...)
```

The keyword `PROPS` determines whether or not the optional group properties list is specified..

One important group property is `FORMAT`. The four types of formatting, `ROW`, `TABLE`, `COLUMN`, or `EXPLICIT`, determine the syntax of the rest of the Free Menu Description. When using `EXPLICIT` formatting, the rest of the description is any number of Item Descriptions which have `LEFT` and `BOTTOM` properties specifying the position of the item in the menu. The syntax is:

INTERLISP-D REFERENCE MANUAL

```
((PROPS FORMAT EXPLICIT ...)
  <ITEM DESCRIPTION>
  <ITEM DESCRIPTION> ...)
```

When using ROW or TABLE formatting, the rest of the description is any number of item groups, each group corresponding to a row in the menu. These groups are identical in syntax to an EXPLICIT group description. The groups have an optional PROPS list and any number of Item Descriptions. The items need not have LEFT and BOTTOM properties, as the location of each item is determined by the formatter. However, the order of the rows and items is important. The menu is laid out top to bottom by row, and left to right within each row. The syntax is:

```
((PROPS FORMAT ROW ...)      ; props of this group
  (<ITEM DESCRIPTION>        ; items in first row
   <ITEM DESCRIPTION> ...)
  ((PROPS ...)               ; props of second row
   <ITEM DESCRIPTION>        ; items in second row
   <ITEM DESCRIPTION> ...))
```

(The comments above only describe the syntax.)

For COLUMN formatting, the syntax is identical to that of ROW formatting. However, each group of items corresponds to a column in the menu, rather than a row. The menu is laid out left to right by column, top to bottom within each column.

Finally, a Free Menu Description can have recursively nested groups. Anywhere the description can take an Item Description, it can take a group, marked by the keyword GROUP. A nested group inherits all of the properties of its mother group, by default. However, any of these properties can be overridden in the nested groups PROPS list, including the FORMAT. The syntax is:

```
(      ; no PROPS list, default row format
<ITEM DESCRIPTION>      ; first in row
(GROUP      ; nested group, second in row
  (PROPS FORMAT COLUMN ...) ; optional props
  (<ITEM DESCRIPTION> ...)  ; first column
  (<ITEM DESCRIPTION> ...))
  <ITEM DESCRIPTION>))      ; third in row
```

Here is an example of a simple Free Menu Description for a menu which might provide access to a simple data base:

```
(( (LABEL LOOKUP SELECTEDFN MYLOOKUPFN)
  (LABEL EXIT SELECTEDFN MYEXITFN))
  ((LABEL Name: TYPE DISPLAY) (LABEL "" TYPE EDIT ID NAME))
  ((LABEL Address: TYPE DISPLAY) (LABEL "" TYPE EDIT ID ADDRESS))
  ((LABEL Phone: TYPE DISPLAY)
```

WINDOWS AND MENUS

```
(LABEL "" TYPE EDIT LIMITCHARS MYPHONEP ID PHONE))
```

This menu has two command buttons, LOOKUP and EXIT, and three edit fields, with IDs NAME, PHONE, and ADDRESS. The Edit items are initialized to the empty string, as in this example they need no other initial value. The user could select the Name: prompt, type a person's name, and then press the LOOKUP button. The function MYLOOKUPFN would be called. That function would look at the NAME Edit item, look up that name in the data base, and fill in the rest of the fields appropriately. The PHONE item has MYPHONEP as a LIMITCHARS function. This function would be called when editing the phone number, in order to restrict input to a valid phone number. After looking up Perry, the Free Menu might look like:

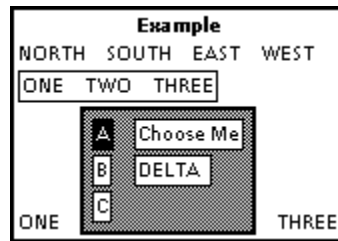
```
LOOKUP EXIT
Name: Herbert Q Perry
Address: 13 Middleperry Dr
Phone: (411) 767-1234
```

Here is a more complicated example:

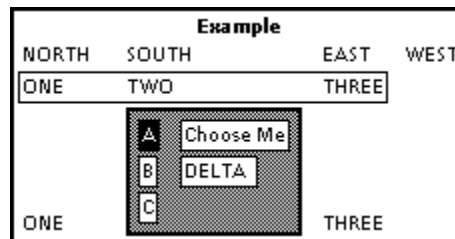
```
((PROPS FONT (MODERN 10))
 (LABEL Example FONT (MODERN 10 BOLD) HJUSTIFY CENTER))
 (LABEL NORTH) (LABEL SOUTH) (LABEL EAST) (LABEL WEST))
 (PROPS ID ROW3 BOX 1)
 (LABEL ONE) (LABEL TWO) (LABEL THREE))
 (PROPS ID ROW4)
 (LABEL ONE ID ALPHA)
 (GROUP (PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4)
  ((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT
T))
  (TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
  (TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
  ((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
   INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
  (TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)))
 (LABEL THREE)))
```

which will produce the following Free Menu:

INTERLISP-D REFERENCE MANUAL



And if the Free Menu were formatted as a Table, instead of in Rows, it would look like:



The following breakdown of the example explains how each part contributes to the Free Menu shown above.

```
(PROPS FONT (MODERN 10))
```

This line specifies the properties of the group that is the entire Free Menu. These properties are described in Section 28.7.4, Free Menu Group Properties. In this example, all items in the Free Menu, unless otherwise specified, will be in Modern 10.

```
((LABEL Example FONT (MODERN 10 BOLD) HJUSTIFY CENTER))
```

This line of the Free Menu Description describes the first row of the menu. Since the FORMAT specification of a Free Menu is, by default, ROW formatting, this line sets the first row in the menu. If the menu were in COLUMN formatting, this position in the description would specify the first column in the menu.

In this example the first row contains only one item. The item is, by default, a type MOMENTARY item. It has its own Font declaration (FONT (MODERN 10 BOLD)), that overrides the font specified for the Free Menu as a whole, so the item appears bolded.

Finally, the item is justified, in this case centered. The HJUSTIFY Item Property indicates that the item is to be centered horizontally within its row.

```
((LABEL NORTH) (LABEL SOUTH) (LABEL EAST) (LABEL WEST))
```

WINDOWS AND MENUS

This line specifies the second row of the menu. The second row has four very simple items, labeled NORTH, SOUTH, EAST, and WEST next to each other within the same row.

```
((PROPS ID ROW3 BOX 1)
 (LABEL ONE) (LABEL TWO) (LABEL THREE))
```

The third row in the menu is similar to the second row, except that it has a box drawn around it. The box is specified in the PROPS declaration for this row. Rows (and columns) are just like Groups in that the first thing in the declaration can be a list of properties for that row. In this case the row is named by giving it an ID property of ROW3. It is useful to name your groups if you want to be able to access and modify their properties later (via the function FM.GROUPPROP). It is boxed by specifying the BOX property with a value of 1, meaning draw the box one dot wide.

```
((PROPS ID ROW4)
 (LABEL ONE ID ALPHA)
 (GROUP (PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4)
  ((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT T))
   (TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
   (TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
  ((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
   INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
   (TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)))
 (LABEL THREE))
```

This part of the description specifies the fourth row in the menu. This row consists of: an item labelled ONE, a group of items, and an item labelled THREE. That is, Free Menu thinks of the group as an entry, and formats the rest of the row just as it it were a large item.

```
(GROUP (PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4)
 ((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT T))
  (TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
  (TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
 ((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
  INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
  (TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)))
```

The second part of this row is a nested group of items. It is declared as a group by placing the keyword GROUP as the first word in the declaration. A group can be declared anywhere a Free Menu Description can take a Free Menu Item Description (as opposed to a row or column declaration).

The first thing in what would have been the second item declaration in this row is the keyword GROUP. Following this keyword comes a normal group description, starting with an optional list of properties, and followed by any number of things to go in the group (based on the format of the group).

INTERLISP-D REFERENCE MANUAL

This group's Props declaration is:

```
(PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4) .
```

It specifies that the group is to be formatted as a number of columns (instead of rows, the default). The entire group will have a background shade of 23130, and a box of width 2 around it, as you can see in the sample menu. The BOXSPACE declaration tells Free Menu to leave an extra four dots of room between the edge of the group (ie the box around the group) and the items in the group.

The first column of this group is a Collection of NWAY items:

```
((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT T))
 (TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
 (TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
```

The three items, labelled A, B, and C are all declared as NWAY items, and are also specified to belong to the same NWAY Collection, Col1. This is how a number of NWAY items are collected together. The property NWAYPROPS (DESELECT T) on the first NWAY item specifies that the Col1 Collection is to have the Deselect property enabled. This simply means that the NWAY collection can be put in the state where none of the items (A, B, or C) are selected (highlighted). Additionally, each item is declared with a box whose width is one dot (pixel) around it.

The second column in this nested group is specified by:

```
((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
  INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
 (TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35))
```

Column two contains two items, a STATE item and a DISPLAY item. The STATE item is labelled "Choose Me." A Label can be a string or a bitmap, as well as an atom. Selecting the STATE item will cause a pop-up menu to appear with two choices for the state of the item, BRAVO and DELTA. The items to go in the pop-up menu are designated by the MENUITEMS property.

The pop-up menu would look like:

Choose Me
BRAVO
DELTA

The initial state of the "Choose Me" item is designated to be DELTA by the INITSTATE Item Property. The initial state can be anything; it does not have to be one of the items in the pop-up menu.

Next, the STATE item is Linked to a DISPLAY item, so that the current state of the item will be displayed in the Free Menu. The link's name is DISPLAY (a special link name for

WINDOWS AND MENUS

STATE items), and the item linked to is described by the Link Description, (GROUP ALPHA). Normally the linked item can just be described by its ID. But in this case, there is more than one item whose ID is ALPHA (for the sake of this example), specifically the first item in the fourth row and the display item in this nested group. The form (GROUP ALPHA) tells Free Menu to search for an item whose ID is ALPHA, limiting the search to the items that are within this lexical group. The lexical group is the smallest group that is declared with the GROUP keyword (i.e., not row and column groups) that contains this item declaration. So in this case, Free Menu will link the STATE item to the DISPLAY item, rather than the first item in the fourth row, since *that* item is outside of the nested group. For further discussion of linking items, see Section 28.7.12, Free Menu Item Links.

Now, establish the DISPLAY item:

```
(TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)
```

We have given it the ID of Alpha that the above STATE item uses in finding the proper DISPLAY item to link to. This display item is used to display the current state of the item "Choose Me." Every item is required to have a Label property specified, but the label for this DISPLAY item will depend on the state of "Choose Me." That is, when the state of the "Choose Me" item is changed from DELTA to BRAVO, the label of the DISPLAY item will also change. The null string serves to hold the place for the changeable label.

A box is specified for this item. Since the label is the empty string, Free Menu would draw a very small box. Instead, the MAXWIDTH property indicates that the label, whatever it becomes, will be limited to a stringwidth of 35. The width restriction of 35 was chosen because it is big enough for each of the possible labels for this display item. So Free Menu draws the box big enough to enclose any item within this width restriction.

Finally we specify the final item in row four:

```
(LABEL THREE)
```

Free Menu Group Properties

Each group has properties. Most group properties are relevant and should be set in the group's PROPS list in the Free Menu Description. User properties can be freely included in the PROPS list. A few other properties are set up by the formatter. The macros FM.GROUPPROP or FM.MENUPROP allow access to group properties after the Free Menu is created.

ID The identifier of this group. Setting the group ID is desirable, for example, if the application needs to get handles on items in particular groups, or access group properties.

FORMAT One of ROW, COLUMN, TABLE, or EXPLICIT. The default is ROW.

FONT A font description of the form (FAMILY SIZE FACE), or a FONTDESCRIPTOR data type. This will be the default font for each item

INTERLISP-D REFERENCE MANUAL

in this group. The default font of the top group is the value of the variable `DEFAULTFONT`.

<code>COORDINATES</code>	One of <code>GROUP</code> or <code>MENU</code> . This property applies only to <code>EXPLICIT</code> formatting. If <code>GROUP</code> , the items in the <code>EXPLICIT</code> group are positioned in coordinates relative to the lower left corner of the group, as determined by the mother group. If <code>MENU</code> , which is the default, the items are positioned relative to the lower left corner of the menu.
<code>LEFT</code>	Specifies a left offset for this group, pushing the group to the right.
<code>BOTTOM</code>	Specifies a bottom offset for this group, pushing the group up.
<code>ROWSPACE</code>	Specifies the number of dots between rows in this group.
<code>COLUMNSPACE</code>	Specifies the number of dots between columns in this group.
<code>BOX</code>	Specifies the number of dots in the box around this group of items.
<code>BOXSHADE</code>	Specifies the shade of the box.
<code>BOXSPACE</code>	Specifies the number of bits between the box and the items.
<code>BACKGROUND</code>	The background shade of this group. Nested groups inherit this background shade, but items in this group and nested groups do not. This is because, in general, it is difficult to read text on a background, so items appear on a white background by default. This can be overridden by the <code>BACKGROUND</code> Item Property.

Other Group Properties

The following group properties are set up and maintained by Free Menu. The application should probably not change any of these properties.

<code>ITEMS</code>	A list of the items in the group.
<code>REGION</code>	The region that is the extent of the items in the group.
<code>MOTHER</code>	The ID of the group that is the mother of this group.
<code>DAUGHTERS</code>	A list of ID of groups which are daughters to this group.

Free Menu Items

Each Free Menu Item is stored as an instance of the data type `FREEMENUITEM`. Free Menu Items can be thought of as objects, each item having its own particular properties, such as its type, label, and mouse event functions. A number of useful item types, described in Section 28.7.11, Predefined Item Types, are predefined by Free Menu. New types of items can be defined by the application, using

Display items as a base. Each Free Menu Item is created from a Free Menu Item Description when the Free Menu is created.

CAUTION: Edit (and thus Number) Freemenu Items do not perform well when boxed or when there is another item to the right in the same row. The display to the right of the edit item may be corrupted under editing and fm.changelabel operations.

Free Menu Item Descriptions

A Free Menu Item Description is a list in property list format, specifying the properties of the item. For example:

```
(LABEL Refetch SELECTEDFN MY.REFETCHFN)
```

describes a MOMENTARY item labelled Refetch, with the function MY.REFETCHFN to be called when the item is selected. None of the property values in an item description are evaluated. When constructing Free Menu descriptions that incorporate evaluated expressions (for example labels that are bitmaps) it is helpful to use the backquote facility. For instance, if the value of the variable MYBITMAP is a bitmap, then

```
(FREEMENU `(( (LABEL A) (LABEL ,MYBITMAP) ) ) )
```

would create a Free Menu of one row, with two items in that row, the second of which has the value of MYBITMAP as its label.

Free Menu Item Properties

The following Free Menu Item Properties can be set in the Item Description. Any other properties given in an Item Description will be treated as user properties, and will be saved on the USERDATA property of the item.

- TYPE** The type of the item. Choose from one of the Free Menu Item type keywords MOMENTARY, TOGGLE, 3STATE, STATE, NWAY, EDITSTART, EDIT, NUMBER, or DISPLAY. The default is MOMENTARY.
- LABEL** An atom, string, or bitmap. Bitmaps are always copied, so that the original will not be changed. This property must be specified for every item.
- FONT** The font in which the item appears. The default is the font specified for the group containing this item. Can be a font description of the form (FAMILY SIZE FACE), or a FONTDESCRIPTOR data type.
- ID** May be used to specify a unique identifier for this item, but is not necessary.

INTERLISP-D REFERENCE MANUAL

LEFT and BOTTOM	When ROW, COLUMN, or TABLE formatting, these specify offsets, pushing the item right and up, respectively, from where the formatter would have put the item. In EXPLICIT formatting, these are the actual coordinates of the item, in the coordinate system given by the group's COORDINATES property.
HJUSTIFY	Indicates horizontal justification type: LEFT, CENTER, or RIGHT. Specifies that this item is to be horizontally justified within the extent of its group. Note that the main group, as opposed to the smaller row or column group, is used.
VJUSTIFY	Specifies that this item is to be vertically justified. Values are TOP, MIDDLE, or BOTTOM.
HIGHLIGHT	Specifies the highlighted looks of the item, that is, how the item changes when a mouse event occurs on it. See Section 28.7.12, Free Menu Item Highlighting, for more details on highlighting.
MESSAGE	Specifies a string that will be printed in the prompt window after a mouse cursor selects this item for MENUHELDWAIT milliseconds. Or, if an atom, treated as a function to get the message. The function is passed three arguments, ITEM, WINDOW, and BUTTONS, and should return a string. The default is a message appropriate to the type of the item.
INITSTATE	Specifies the initial state of the item. This is only appropriate to TOGGLE, 3STATE, and STATE items.
MAXWIDTH	Specifies the width allowed for this item. The formatter will leave enough space after the item for the item to grow to this width without collisions.
MAXHEIGHT	Similar to MAXWIDTH, but in the vertical dimension.
BOX	Specifies the number of bits in the box around this item. Boxes are made around MAXWIDTH and MAXHEIGHT dimensions. If unspecified, no box is drawn.
BOXSHADE	Specifies the shade that the box is drawn in. The default is BLACKSHADE.
BOXSPACE	Specifies the number of bits between the box and the label. The default is one bit.
BACKGROUND	Specifies the background shade on which the item appears. The default is WHITESHADE, regardless of the group's background.
LINKS	Can be used to link this item to other items in the Free Menu.

Mouse Properties

The following properties provide a way for application functions to be called under certain mouse events. These functions are called with the ITEM, the WINDOW, and the BUTTONS passed as arguments. These application functions do not interfere with any Free Menu system functions that take care of handling the different item types. In each case, though, the application function is called

WINDOWS AND MENUS

after the system function. The default for all of these functions is `NILL`. The value of each of the following properties can be the name of a function, or a lambda expression.

<code>SELECTEDFN</code>	Specifies the function to be called when this item is selected. The <code>Edit</code> and <code>EditStart</code> items cannot have a <code>SELECTEDFN</code> . See the <code>Edit Free Menu</code> item description in Section 28.7.11, <i>Predefined Item Types</i> , for more information.
<code>DOWNFN</code>	Specifies the function to be called when the item is selected with the mouse cursor.
<code>HELDFN</code>	Specifies the function to be called repeatedly when the item is selected with the mouse cursor.
<code>MOVEDFN</code>	Specifies the function to be called when the mouse cursor moves off this item (mouse buttons are still depressed).

System Properties

The following `Free Menu Item` properties are set and maintained by `Free Menu`. The application should probably not change these properties directly.

<code>GROUPLD</code>	Specifies the ID of the smallest group that the item is in. For example, in a row formatted group, the item's <code>GROUPLD</code> will be set to the ID of the row that the item is in, not the ID of the whole group.
<code>STATE</code>	Specifies the current state of <code>TOGGLE</code> , <code>3STATE</code> , or <code>STATE</code> items. The state of an <code>NWAY</code> item behaves like that of a toggle item.
<code>BITMAP</code>	Specifies the bitmap from which the item is displayed.
<code>REGION</code>	Specifies the region of the item, in window coordinates. This is used for locating the display position, as well as determining the mouse sensitive region of the item.
<code>MAXREGION</code>	Specifies the maximum region the item may occupy, determined by the <code>MAXWIDTH</code> and <code>MAXHEIGHT</code> properties (see Section 28.7.8, <i>Free Menu item Properties</i>). This is used by the formatter and the display routines.
<code>SYSDOWNFN</code>	
<code>SYSMOVEDFN</code>	
<code>SYSSELECTEDFN</code>	These are the system mouse event functions, set up by <code>Free Menu</code> according to the item type. These functions are called before the mouse event functions, and are used to implement highlighting, state changes, editing, etc.
<code>USERDATA</code>	Specifies how any other properties are stored on this list in property list format. This list should probably not need to be manipulated directly.

Predefined Item Types

INTERLISP-D REFERENCE MANUAL

MOMENTARY [Free Menu Item]

MOMENTARY items are like command buttons. When the button is selected, its associated function is called.

TOGGLE [Free Menu Item]

Toggle items are simple two-state buttons. When pressed, the button is highlighted; it stays that way until pressed again. The states of a toggle button are T and NIL; the initial state is NIL.

3STATE [Free Menu Item]

3STATE items rotate through NIL, T, and OFF, states each time they are pressed. The default looks of the OFF state are with a diagonal line through the button, while T is highlighted, and NIL is normal. The default initial state is NIL.

The following Item Property applies to 3STATE items:

OFF Specifies the looks of a 3STATE item in its OFF state. Similar to HIGHLIGHT. The default is that the label gets a diagonal slash through it.

NOTE: If you specify special highlighting (a different bitmap of string) for Toggle or 3State items AND use this item in a group formatted as a Column or a Table, the highlight looks of the item may not appear in the correct place.

STATE [Free Menu Item]

STATE items are general multiple state items. The following Item Property determines how the item changes state:

CHANGESTATE This Item Property can be changed at any time to change the effect of the item. If a MENU data type, this menu pops up when the item is selected, and the user can select the new state. Otherwise, if this property is given, it is treated as a function name, which is passed three arguments, ITEM, WINDOW, and BUTTONS. This function can do whatever it wants, and is expected to return the new state (an atom, string, or bitmap), or NIL, indicating the state should not change. The state of the item can automatically be indicated in the Free Menu, by setting up a DISPLAY link to a DISPLAY item in the menu (see Section 28.7.13, Free Menu Item Links). If such a link exists, the label of the DISPLAY item will be changed to the new state. The possible states are not restricted at all, with the exception of selections from a pop-up menu. The state can be changed to any atom, string, or bitmap, manually via FM.CHANGESTATE.

The following Item Properties are relevant to STATE items when building a Free Menu:

MENUIITEMS If specified, should be a list of items to go in a pop-up menu for this item. Free Menu will build the menu and save it as the CHANGESTATE property of the item.

WINDOWS AND MENUS

MENUFONT The font of the items in the pop-up menu.

MENUTITLE The title of the pop-up menu. The default title is the label of the **STATE** item.

NWAY [Free Menu Item]

NWAY items provide a way to collect any number of items together, in any format within the Free Menu. Only one item from each Collection can be selected at a time, and that item is highlighted to indicate this. The following Item Properties are particular to **NWAY** items:

COLLECTION An identifier that specifies which **NWAY** Collection this item belongs to.

NWAYPROPS A property list of information to be associated with this collection. This property is only noticed in the Free Menu Description on the first item in a **COLLECTION**. **NWAY** Collections are formed by creating a number of **NWAY** items with the same **COLLECTION** property. Each **NWAY** item acts individually as a Toggle item, and can have its own mouse event functions. Each **NWAY** Collection itself has properties, its state for instance. After the Free Menu is created, these Collection properties can be accessed by the macro **FM.NWAYPROPS**. Note that **NWAY** Collections are different from Free Menu Groups. There are three **NWAY** Collection properties that Free Menu looks at:

DESELECT If given, specifies that the Collection can be deselected, yielding a state in which no item in the Collection is selected. When this property is set, the Collection can be deselected by selecting any item in the Collection and pressing the right mouse button.

STATE The current state of the Collection, which is the actual item selected.

INITSTATE Specifies the initial state of the Collection. The value of this property is an Item Link Description

EDIT [Free Menu Item]

EDIT items are textual items that can be edited. The label for an **EDIT** item cannot be a bitmap. When the item is selected an edit caret appears at that cursor position within the item, allowing insertion and deletion of characters at that point. If selected with the right mouse button, the item is cleared before editing starts. While editing, the left mouse button moves the caret to a new position within the item. The right mouse button deletes from the caret to the cursor. **CONTROL-W** deletes the previous word. Editing is stopped when another item is selected, when the user moves the cursor into another TTY window and clicks the cursor, or when the Free Menu function **FM.ENDEDIT** is called (called when the Free Menu is reset, or the window is closed). The Free Menu editor will time out after about a minute, returning automatically. Because of the many ways in which editing can terminate, **EDIT** items are not allowed to have a **SELECTEDFN**, as it is not clear when this function should be called. Each **EDIT** item should have an ID specified, which is used when getting the state of the Free Menu, since the string being edited is defined as the state of the item, and thus cannot distinguish edit items. The following Item Properties are specific to **EDIT** items.

INTERLISP-D REFERENCE MANUAL

- MAXWIDTH** Specifies the maximum string width of the item, in bits, after which input will be ignored. If **MAXWIDTH** is not specified, the item becomes infinitely wide and input is never restricted.
- INFINITEWIDTH** This property is set automatically when **MAXWIDTH** is not specified. This tells Free Menu that the item has no right end, so that the item becomes mouse sensitive from its left edge to the right edge of the window, within the vertical space of the item.
- In Medley, Changestate of an infinite width Edit item to a smaller item clears the old item properly.
- LIMITCHARS** The input characters allowed can be restricted in two ways: If this item property is a list, it is treated as a list of legal characters; any character not in the list will be ignored. If it is an atom, it is treated as the name of a test predicate, which is passed three arguments, **ITEM**, **WINDOW**, and **CHARACTER**, when each character is typed. This predicate should return **T** if the character is legal, **NIL** otherwise. The **LIMITCHARS** function can also call **FM.ENDEDIT** to force the editor to terminate, or **FM.SKIPNEXT**, to cause the editor to jump to the next edit item in the menu.
- ECHOCHAR** This item property can be set to any character. This character will be echoed in the window, regardless of what character is typed. However the item's label contains the actual string typed. This is useful for operations like password prompting. If **ECHOCHAR** is used, the font of the item must be fixed pitch. Unrestricted **EDIT** items should not have other items to their right in the menu, as they will be replaced. If the item is boxed, input is restricted to what will fit in the box. Typing off the edge of the window will cause the window to scroll appropriately. Control characters can be edited, including the carriage return and line feed, and they are echoed as a black box. While editing, the Skip/Next key ends editing the current item, and starts editing the next **EDIT** item in the Free Menu.

NUMBER [Free Menu Item]

NUMBER items are **EDIT** items that are restricted to numerals. The state of the item is coerced to the number itself, not a string of numerals. There is one **NUMBER**-specific Item Property:

- NUMBERTYPE** If **FLOATP** (or **FLOAT**), then decimals are accepted. Otherwise only whole numbers can be edited.

EDITSTART [Free Menu Item]

EDITSTART items serve the purpose of starting editing on another item when they are selected. The associated Edit item is linked to the EditStart item by an **EDIT** link (see Free Menu Item Links below). If the **EDITSTART** item is selected with the right mouse button, the Edit item is cleared before editing is started. Similar to **EDIT** items, **EDITSTART** items cannot have a **SELECTEDFN**, as it is not clear when the associated editing will terminate.

WINDOWS AND MENUS

In Medley, `EDITSTART` items linked to a Number item properly set number state when editing has completed.

DISPLAY

[Free Menu Item]

`DISPLAY` items serve two purposes. First, they simply provide a way of putting dummy text in a Free Menu, which does nothing when selected. The item's label can be changed, though. Secondly, `DISPLAY` items can be used as the base for new item types. The application can create new item types by specifying `DOWNFN`, `HELDFN`, `MOVEDFN`, and `SELECTEDFN` for a `DISPLAY` item, making it behave as desired.

Free Menu Item Highlighting

Each Free Menu Item can specify how it wants to be highlighted. First of all, if the item does not specify a `HIGHLIGHT` property, there are two default highlights. If the item is not boxed, the label is simply inverted, as in normal menus. If the item is boxed, it is highlighted in the shade of the box. Alternatively, the value of the `HIGHLIGHT` property can be a `SHADE`, which will be painted on top of the item when a mouse event occurs on it. Or the `HIGHLIGHT` property can be an alternate label, which can be an atom, string or bitmap. If the highlight label is a different size than the item label, the formatter will leave enough space for the larger of the two. In all of these cases, the looks of the highlighted item are determined when the Free Menu is built, and a bitmap of the item with these looks is created. This bitmap is stored on the item's `HIGHLIGHT` property, and simply displayed when a mouse event occurs. The value of the highlight property in the Item Description is copied to the `USERDATA` list, in case it is needed later for a label change.

Free Menu Item Links

Links between items are useful for grouping items in abstract ways. In particular, links are used for associating `EDITSTART` items with their item to edit, and `STATE` items with their state display. The Free Menu Item property `LINKS` is a property list, where the value of each Link Name property is a pointer to another item. In the Item Description, the value of the `LINK` property should be a property list as above. The value of each Link Name property is a Link Description. A Link Description can be one of the following forms:

- `<ID>` An ID of an item in the Free Menu. This is acceptable if items can be distinguished by ID alone.
- `(<GROUPID> <ID>)` A list whose first element is a `GROUPID`, and whose second element is the ID of an item in that group. This way items with similar purposes, and thus similar ID's, can be distinguished across groups.
- `(GROUP <ID>)` A list whose first element is the keyword `GROUP`, and whose second element is an item ID. This form describes an item with ID, in the same group that this item is in. This way you do not need to know the `GROUPID`, just which group it is in.

INTERLISP-D REFERENCE MANUAL

Then after the entire menu is built, the links are set up, turning the Link Descriptions into actual pointers to Free Menu Items. There is no reason why circular Item Links cannot be created, although such a link would probably not be very useful. If circular links are created, the Free Menu will not be garbage collected after it is not longer being used. The application is responsible for breaking any such links that it creates.

Free Menu Window Properties

FM.PROMPTWINDOW	Specifies the window that Free Menu should use for displaying the item's messages. If not specified, PROMPTWINDOW is used.
FM.BACKGROUND	The background shade of the entire Free Menu. This property can be set automatically by specifying a BACKGROUND argument to the function FREEMENU. The window border must be 4 or greater when a Free Menu background is used, due to the way the Window System handles window borders.
FM.DONTRESHAPE	Normally, Free Menu will attempt to use empty space in a window by pushing items around to fill the space. When a Free Menu window is reshaped, the items are repositioned in the new shape. This can be disabled by setting the FM.DONTRESHAPE window property.

Free Menu Interface Functions

(**FREEMENU** *DESCRIPTION TITLE BACKGROUND BORDER*) [Function]

Creates a Free Menu from a Free Menu Description, returning the window. This function will return quickly unless new display fonts have to be created.

Accessing Functions

(**FM.GETITEM** *ID GROUP WINDOW*) [Function]

Gets item *ID* in *GROUP* of the Free Menu in *WINDOW*. This function will search the Free Menu for an item whose *ID* property matches, or secondly whose *LABEL* property matches *ID*. If *GROUP* is *NIL*, then the entire Free Menu is searched. If no matching item is found, *NIL* is returned.

(**FM.GETSTATE** *WINDOW*) [Function]

Returns in property list format the ID and current *STATE* of every *NWAY* Collection and item in the Free Menu. If an item's or Collection's state is *NIL*, then it is not included in the list. This provides an easy way of getting the state of the menu all at once. If the state of only one item or Collection is needed, the application can directly access the *STATE* property of that object using the Accessing Macros described in Section 28.7.20, Free Menu Macros. This function can be called when editing is in progress, in which case it will provide the label of the item being edited at that point.

Changing Free Menus

Many of the following functions operate on Free Menu Items, and thus take the item as an argument. The *ITEM* argument to these functions can be the Free Menu Item itself, or just a reference to the item. In the second case, `FM.GETITEM` (see Section 28.7.16, Accessing Functions) will be used to find the item in the Free Menu. The reference can be in one of the following forms:

<ID> Specifies the first item in the Free Menu whose ID or LABEL property matches <ID>.

(<GROUPID> <ID>) Specifies the item whose ID or LABEL property matches <ID> within the group specified by <GROUPID>.

(**FM.CHANGELABEL** *ITEM NEWLABEL WINDOW UPDATEFLG*) [Function]

Changes an *ITEM*'s label after the Free Menu has been created. It works for any type of item, and *STATE* items will remain in their current state. If the window is open, the item will be redisplayed with its new appearance. *NEWLABEL* can be an atom, a string, or a bitmap (except for *EDIT* items), and will be restricted in size by the *MAXWIDTH* and *MAXHEIGHT* Item Properties. If these properties are unspecified, the *ITEM* will be able to grow to any size. *UPDATEFLG* specifies whether or not the regions of the groups in the menu are recalculated to take into account the change of size of this item. The application should not change the label of an *EDIT* item while it is being edited. The following Item Property is relevant to changing labels:

CHANGELABELUPDATE Exactly like *UPDATEFLG* except specified on the item, rather than as a function parameter.

(**FM.CHANGESTATE** *X NEWSTATE WINDOW*) [Function]

Programmatically changes the state of items and *NWAY* Collections. *X* is either an item or a Collection name. For items *NEWSTATE* is a state appropriate to the type of the item. For *NWAY* Collections, *NEWSTATE* should be the desired item in the Collection, or *NIL* to deselect. For *EDIT* and *NUMBER* items, this function just does a label change. If the window is open, the item will be redisplayed.

(**FM.RESETSTATE** *ITEM WINDOW*) [Function]

Sets an *ITEM* back to its initial state.

(**FM.RESETMENU** *WINDOW*) [Function]

Resets every item in the menu back to its initial state.

(**FM.RESETSHAPE** *WINDOW ALWAYSFLG*) [Function]

Reshapes the *WINDOW* to its full extent, leaving the lower-left corner unmoved. Unless *ALWAYSFLG* is *T*, the window will only be increased in size as a result of resetting the shape.

(**FM.RESETGROUPS** *WINDOW*) [Function]

INTERLISP-D REFERENCE MANUAL

Recalculates the extent of each group in the menu, updating group boxes and backgrounds appropriately.

(**FM.HIGHLIGHTITEM** *ITEM WINDOW*) [Function]

Programmatically forces an *ITEM* to be highlighted. This might be useful for *ITEMs* which have a direct effect on other *ITEMs* in the menu. The *ITEM* will be highlighted according to its `HIGHLIGHT` property, as described in Section 28.7.12, Free Menu Item Highlighting. This highlight is temporary, and will be lost if the *ITEM* is redisplayed, by scrolling for example.

Editor Functions

(**FM.EDITITEM** *ITEM WINDOW CLEARFLG*) [Function]

Starts editing an `EDIT` or `NUMBER` *ITEM* at the beginning of the *ITEM*, as long as the *WINDOW* is open. This function will most likely be useful for starting editing of an *ITEM* that is currently the null string. If *CLEARFLG* is set, the *ITEM* is cleared first.

(**FM.SKIPNEXT** *WINDOW CLEARFLG*) [Function]

Causes the editor to jump to the beginning of the next `EDIT` item in the Free Menu. If *CLEARFLG* is set, then the next item will be cleared first. If there is not another `EDIT` item in the menu, this function will simply cause editing to stop. If this function is called when editing is not in progress, editing will begin on the first `EDIT` item in the menu. This function can be called from any process, and can also be called from inside the editor, in a `LIMITCHARS` function.

(**FM.ENDEDIT** *WINDOW WAITFLG*) [Function]

Stops any editing going on in *WINDOW*. If *WAITFLG* is `T`, then block until the editor has completely finished. This function can be called from another process, or from a `LIMITCHARS` function.

(**FM.EDITP** *WINDOW*) [Function]

If an item is in the process of being edited in the Free Menu *WINDOW*, that item is returned. Otherwise, `NIL` is returned.

Miscellaneous Functions

(**FM.REDISPLAYMENU** *WINDOW*) [Function]

Redisplays the entire Free Menu in its *WINDOW*, if the *WINDOW* is open.

(**FM.REDISPLAYITEM** *ITEM WINDOW*) [Function]

Redisplays a particular Free Menu *ITEM* in its *WINDOW*, if the *WINDOW* is open.

WINDOWS AND MENUS

(**FM.SHADE** *X SHADE WINDOW*)

[Function]

X can be an item, or a group ID. *SHADE* is painted on top of the item or group. Note that this is a temporary operation, and will be undone by redisplaying. For more permanent shading, the application may be able to add a *REDEDISPLAYFN* and *SCROLLFN* for the window as necessary to update the shading.

(**FM.WHICHITEM** *WINDOW POSorX Y*)

[Function]

Locates and identifies an item from its known location within the *WINDOW*. If *WINDOW* is NIL, (*WHICHW*) is used, and if *POSorX* is NIL, the current cursor location is used.

(**FM.TOPGROUPID** *WINDOW*)

[Function]

Returns the ID of the top group of this Free Menu.

Free Menu Macros

These Accessing Macros are provided to allow the application to get and set information in the Free Menu data structures. They are implemented as macros so that the operation will compile into the actual access form, rather than figuring that out at run time.

(**FM.ITEMPROP** *ITEM PROP {VALUE}*)

[Macro]

Similar to *WINDOWPROP*, this macro provides an easy access to the fields of a Free Menu Item. The function *FM.GETITEM* gets the *ITEM*, described in Section 28.7.16, Accessing Function. *VALUE* is optional, and if not given, the current value of the *PROP* property will be returned. If *VALUE* is given, it will be used as the new value for that *PROP*, and the old value will be returned. When a call to *FM.ITEMPROP* is compiled, if the *PROP* is known (quoted in the calling form), the macro figures out what field to access, and the appropriate Data Type access form is compiled. However, if the *PROP* is not known at compile time, the function *FM.ITEMPROP*, which goes through the necessary property selection at run time, is compiled. The *TYPE* and *USERDATA* properties of a Free Menu Item are Read Only, and an error will result from trying to change the value of one of these properties.

(**FM.GROUPPROP** *WINDOW GROUP PROP {VALUE}*)

[Macro]

Provides access to the Group Properties set up in the *PROPS* list for each group in the Free Menu Description. *GROUP* specifies the ID of the desired group, and *PROP* the name of the desired property. If *VALUE* is specified, it will become the new value of the property, and the old value will be returned. Otherwise, the current value is returned.

(**FM.MENUPROP** *WINDOW PROP {VALUE}*)

[Macro]

Provides access to the group properties of the top-most group in the Free Menu, that is to say, the entire menu. This provides an easy way for the application to attach properties to the menu as a whole, as well as access the Group Properties for the entire menu.

INTERLISP-D REFERENCE MANUAL

(**FM.NWAYPROP** *WINDOW COLLECTION PROP {VALUE}*) [Macro]

This macro works just like `FM.GROUPPROP`, except it provides access to the `NWay` Collections.

Attached Windows

The attached window facility makes it easy to manipulate a group of window as a unit. Standard window operations like moving, reshaping, opening, and closing can be done so that it appears to the user as if the windows are a single entity. Each collection of attached windows has one main window and any number of other windows that are "attached" to it. Moving or reshaping the main window causes all of the attached windows to be moved or reshaped as well. Moving or reshaping an attached window does not affect the main window.

Attached windows can have other windows attached to them. Thus, it is possible to attach window A to window B when B is already attached to window C. Similarly, if A has other windows attached to it, it can still be attached to B.

(**ATTACHWINDOW** *WINDOWTOATTACH MAINWINDOW EDGE POSITIONONEDGE WINDOWCOMACTION*) [Function]

Associates *WINDOWTOATTACH* with *MAINWINDOW* so that window operations done to *MAINWINDOW* are also done to *WINDOWTOATTACH* (the exact set of window operations passed between main windows and attached windows is described in the Window Operations and Attached Windows section below). **ATTACHWINDOW** moves *WINDOWTOATTACH* to the correct position relative to *MAINWINDOW*.

Note: A window can be attached to only one other window. Attaching a window to a second window will detach it from the first. Attachments can not form loops. That is, a window cannot be attached to itself or to a window that is attached to it. **ATTACHWINDOW** will generate an error if this is attempted.

EDGE determines which edge of *MAINWINDOW* the attached window is positioned along: it should be one of TOP, BOTTOM, LEFT, or RIGHT. If *EDGE* is NIL, it defaults to TOP.

POSITIONONEDGE determines where along *EDGE* the attached window is positioned. It should be one of the following:

- LEFT The attached window is placed on the left (of a TOP or BOTTOM edge).
- RIGHT The attached window is placed on the right (of a TOP or BOTTOM edge).
- BOTTOM The attached window is placed on the bottom (of a LEFT or RIGHT edge).
- TOP The attached window is placed on the top (of a LEFT or RIGHT edge).
- CENTER The attached window is placed in the center of the edge.

WINDOWS AND MENUS

JUSTIFY

or NIL The attached window is placed to fill the entire edge. ATTACHWINDOW reshapes the window if necessary.

Note: The width or height used to justify an attached window includes any other windows that have already been attached to *MAINWINDOW*. Thus (ATTACHWINDOW BBB AAA 'RIGHT' JUSTIFY) followed by (ATTACHWINDOW CCC AAA 'TOP' JUSTIFY) will put CCC across the top of both BBB and AAA:



WINDOWCOMACTION provides a convenient way of specifying how *WINDOWTOATTACH* responds to right button menu commands. The window property *PASSTOMAINCOMS* determines which right button menu commands are directly applied to the attached window, and which are passed to the main window (see the Window Operations and Attached Windows section below). Depending on the value of *WINDOWCOMACTION*, the *PASSTOMAINCOMS* window property of *WINDOWTOATTACH* is set as follows:

- NIL *PASSTOMAINCOMS* is set to (CLOSEW MOVEW SHAPEW SHRINKW BURYW), so right button menu commands to close, move, shape, shrink, and bury are passed to the main window, and all others are applied to the attached window.
- LOCALCLOSE *PASSTOMAINCOMS* is set to (MOVEW SHAPEW SHRINKW BURYW), which is the same as when *WINDOWCOMACTION* is NIL, except that the attached window can be closed independently.
- HERE *PASSTOMAINCOMS* is set to NIL, so all right button menu commands are applied to the attached window.
- MAIN *PASSTOMAINCOMS* is set to T, so all right button menu commands are passed to the main window.

Note: If the user wants to set the *PASSTOMAINCOMS* window property of an attached window to something else, it must be done after the window is attached, since ATTACHWINDOW modifies this window property.

(DETACHWINDOW WINDOWTODETACH)

[Function]

INTERLISP-D REFERENCE MANUAL

Detaches *WINDOWTODETACH* from its main window. Returns a dotted pair (*EDGE* . *POSITIONONEDGE*) if *WINDOWTODETACH* was an attached window, *NIL* otherwise. This does not close *WINDOWTODETACH*.

(**DETACHALLWINDOWS** *MAINWINDOW*) [Function]

Detaches and closes all windows attached to *MAINWINDOW*.

(**FREEATTACHEDWINDOW** *WINDOW*) [Function]

Detaches the attached window *WINDOW*. In addition, other attached windows above (in the case of a *TOP* attached window) or below (in the case of a *BOTTOM* attached window) are moved closer to the main window to fill the gap.

Note: Attached windows that "reject" the move operation (see *REJECTMAINCOMS* below) are not moved.

Note: *FREEATTACHEDWINDOW* currently doesn't handle *LEFT* or *RIGHT* attached windows.

(**REMOVEWINDOW** *WINDOW*) [Function]

Closes *WINDOW*, and calls *FREEATTACHEDWINDOW* to move other attached windows to fill any gaps.

(**REPOSITIONATTACHEDWINDOWS** *WINDOW*) [Function]

Repositions every window attached to *WINDOW*, in the order that they were attached. This is useful as a *RESHAPEFN* for main windows with attached window that don't want to be reshaped, but do want to keep their position relative to the main window when the main window is reshaped.

Note: Attached windows that "reject" the move operation (see *REJECTMAINCOMS* below) are not moved.

(**MAINWINDOW** *WINDOW RECURSEFLG*) [Function]

If *WINDOW* is not a window, it generates an error. If *WINDOW* is closed, it returns *WINDOW*. If *WINDOW* is not attached to another window, it returns *WINDOW* itself. If *RECURSEFLG* is *NIL* and *WINDOW* is attached to a window, it returns that window. If *RECURSEFLG* is *T*, it returns the first window up the "main window" chain starting at *WINDOW* that is not attached to any other window.

(**ATTACHEDWINDOWS** *WINDOW COM*) [Function]

Returns the list of windows attached to *WINDOW*.

If *COM* is non-*NIL*, only those windows attached to *WINDOW* that do not reject the window operation *COM* are returned (see *REJECTMAINCOMS*).

(**ALLATTACHEDWINDOWS** *WINDOW*) [Function]

WINDOWS AND MENUS

Returns a list of all of the windows attached to *WINDOW* or attached to a window attached to it.

(**WINDOWREGION** *WINDOW COM*) [Function]

Returns the screen region occupied by *WINDOW* and its attached windows, if it has any.

If *COM* is non-NIL, only those windows attached to *WINDOW* that do not reject the window operation *COM* are considered in the calculation (see REJECTMAINCOMS).

(**WINDOWSIZE** *WINDOW*) [Function]

Returns the size of *WINDOW* and its attached windows (if any), as a dotted pair (WIDTH . HEIGHT).

(**MINATTACHEDWINDOWEXTENT** *WINDOW*) [Function]

Returns the minimum size that *WINDOW* and its attached windows (if any) will accept, as a dotted pair (WIDTH . HEIGHT).

Attaching Menus To Windows

The following functions are provided to associate menus to windows.

(**MENUWINDOW** *MENU VERTFLG*) [Function]

Returns a closed window that has the menu *MENU* in it. If *MENU* is a list, a menu is created with *MENU* as its ITEMS menu field. Otherwise, *MENU* should be a menu. The returned window has the appropriate RESHAPEFN, MINSIZE and MAXSIZE window properties to allow its use in a window group.

If both the MENUROWS and MENCOLUMNS fields of *MENU* are NIL, *VERTFLG* is used to set the default menu shape. If *VERTFLG* is non-NIL, the MENCOLUMNS field of *MENU* will be set to 1 (the menu items will be listed vertically); otherwise the MENUROWS field of *MENU* will be set to 1 (the menu items will be listed horizontally).

(**ATTACHMENU** *MENU MAINWINDOW EDGE POSITIONONEDGE NOOPENFLG*) [Function]

Creates a window that contains the menu *MENU* (by calling *MENUWINDOW*) and attaches it to the window *MAINWINDOW* on edge *EDGE* at position *POSITIONONEDGE*. The menu window is opened unless *MAINWINDOW* is closed, or *NOOPENFLG* is T.

If *EDGE* is either LEFT or RIGHT, *MENUWINDOW* will be called with *VERTFLG* = T, so the menu items will be listed vertically; otherwise the menu items will be listed horizontally. These defaults can be overridden by specifying the MENUROWS or MENCOLUMNS fields in *MENU*.

(**CREATEMENUEDWINDOW** *MENU WINDOWTITLE LOCATION WINDOWSPEC*) [Function]

INTERLISP-D REFERENCE MANUAL

Creates a window with an attached menu and returns the main window. *MENU* is the only required argument, and may be a menu or a list of menu items. *WINDOWTITLE* is a string specifying the title of the main window. *LOCATION* specifies the edge on which to place the menu; the default is TOP. *WINDOWSPEC* is a region specifying a region for the aggregate window; if NIL, the user is prompted for a region.

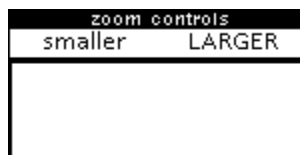
Examples:

```
(SETQ MENUW
  (MENUWINDOW
    (create MENU
      ITEMS ← '(smaller LARGER)
      MENUFONT ← '(MODERN 12)
      TITLE ← "zoom controls"
      CENTERFLG ← T
      WHENSELECTEDFN ← (FUNCTION ZOOMMAINWINDOW))))
```

creates (but does not open) a menu window that contains the two items "smaller" and "LARGER" with the title "zoom controls" and that calls the function ZOOMMAINWINDOW when an item is selected. Note that the menu items will be listed horizontally, because MENUWINDOW is called with VERTFLG = NIL, and the menu does not specify either a MENUROWS or MENCOLUMNS field.

```
(ATTACHWINDOW MENUW
  (CREATEW '(50 50 150 50))
  'TOP
  'JUSTIFY)
```

creates a window on the screen and attaches the above created menu window to its top:



```
(CREATEMENUEDWINDOW
  (create MENU
    ITEMS ← '(smaller LARGER)
    MENUFONT ← '(MODERN 12)
    TITLE ← "zoom controls"
    CENTERFLG ← T
    WHENSELECTEDFN ← (FUNCTION ZOOMMAINWINDOW))))
```

creates the same sort of window in one step, prompting the user for a region.

Attached Prompt Windows

Many packages have a need to display status information or prompt for small amounts of user input in a place outside their standard window. A convenient way to do this is to attach a small window to the top of the program's main window. The following functions do so in a uniform way that can be depended on among diverse applications.

WINDOWS AND MENUS

(GETPROMPTWINDOW MAINWINDOW #LINES FONT DONTCREATE) [Function]

Returns the attached prompt window associated with *MAINWINDOW*, creating it if necessary. The window is always attached to the top of *MAINWINDOW*, has *DSPSCROLL* set to T, and has a *PAGEFULLFN* of NIL to inhibit page holding. The window is at least *#LINES* lines high (default 1); if a pre-existing window is shorter than that, it is reshaped to make it large enough. *FONT* is the font to give the prompt window (defaults to the font of *MAINWINDOW*), and applies only when the window is first created. If *DONTCREATE* is true, returns the window if it exists, otherwise NIL without creating any prompt window.

(REMOVEPROMPTWINDOW MAINWINDOW) [Function]

Detaches the attached prompt window associated with *MAINWINDOW* (if any), and closes it.

Window Operations And Attached Windows

When a window operation, such as moving or clearing, is performed on a window, there is a question about whether or not that operation should also be performed on the windows attached to it or performed on the window it is attached to. The "right" thing to do depends on the window operation: it makes sense to independently redisplay a single window in a collection of windows, whereas moving a single window usually implies moving the whole group of windows. The interpretation of window operations also depends on the application that the window group is used for. For some applications, it may be desirable to have a window group where individual windows can be moved away from the group, but still be conceptually attached to the group for other operations. The attached window facility is flexible enough to allow all of these possibilities.

The operation of window operations can be specified by each attached window, by setting the following two window properties:

PASSTOMAINCOMS [Window Property]

Value is a list of window commands (e.g. CLOSEW, MOVEW) which, when selected from the attached window's right-button menu, are actually applied to the central window in the group, instead of being applied to the attached window itself. The "central window" is the first window up the "main window" chain that is not attached to any other window.

If *PASSTOMAINCOMS* is NIL, all window operations are directly applied to the attached window. If *PASSTOMAINCOMS* is T, all window operations are passed to the central window.

Note: ATTACHWINDOW allows this window property to be set to commonly-used values by using its WINDOWCOMACTION argument. ATTACHWINDOW always sets this window property, so users must modify it directly only after attaching the window to another window.

REJECTMAINCOMS [Window Property]

INTERLISP-D REFERENCE MANUAL

Value is a list of window commands that the attached window will not allow the main window to apply to it. This is how a window can say "leave me out of this group operation."

If REJECTMAINCOMS is NIL, all window commands may be applied to this attached window. If REJECTMAINCOMS is T, no window commands may be applied to this attached window.

The PASSTOMAINCOMS and REJECTMAINCOMS window properties affect right-button menu operations applied to main windows or attached windows, and the action of programmatic window functions (SHAPEW, MOVEW, etc.) applied to main windows. However, these window properties do not affect the action of window functions applied to attached windows.

The following list describes the behavior of main and attached windows under the window operations, assuming that all attached windows have their REJECTMAINCOMS window property set to NIL and PASSTOMAINCOMS set to (CLOSEW MOVEW SHAPEW SHRINKW BURYW) (the default if ATTACHWINDOW is called with WINDOWCOMACTION = NIL).

The behavior for any particular operation can be changed for particular attached windows by setting the standard window properties (e.g., MOVEFN or CLOSEFN) of the attached window. An exception is the TOTOPFN property of an attached window, that is set to bring the whole window group to the top and should not be set by the user (although users can add functions to the TOTOPFN window property).

Move If the main window moves, all attached windows move with it, and the relative positioning between the main window and the attached windows is maintained. If the region is determined interactively, the prompt region for the move is the union of the extent of the main window and all attached windows (excluding those with MOVEW in their REJECTMAINCOMS window property).

If an attached window is moved by calling the function MOVEW, it is moved without affecting the main window. If the right-button window menu command Move is called on an attached window, it is passed on to the main window, so that all windows in the group move.

Reshape If the main window is reshaped, the minimum size of it and all of its attached windows is used as the minimum of the space for the result. Any space greater than the minimum is distributed among the main window and its attached windows. Attached windows with SHAPEW on their REJECTMAINCOMS window property are ignored when finding the minimum size, creating a "ghost" region, or distributing space after a reshape.

If an attached window is reshaped by calling the function SHAPEW, it is reshaped independently. If the right-button window menu command Shape is called on an attached window, it is passed on to the main window, so the whole group is reshaped.

WINDOWS AND MENUS

Note: Reshaping the main window will restore the conditions established by the call to ATTACHWINDOW, whereas moving the main window does not. Thus, if A is attached to the top of B and then moved by the user, its new position relative to B will be maintained if B is moved. If B is reshaped, A will be reshaped to the top of B. Additionally, if, while A is moved away from the top of B, C is attached to the top of B, C will position itself above where A used to be.

Close If the main window is closed, all of the attached windows are closed also and the links from the attached windows to the main window are broken. This is necessary for the windows to be garbage collected.

If an attached window is closed by calling the function CLOSEW, it is closed without affecting the main window. If the right-button window menu command Close is called on an attached window, it is passed on to the main window. Note that closing an attached window detaches it.

Open If the main window is opened, it opens all attached windows and reestablishes links from them to the main window.

Attached windows can be opened independently and this does not affect the main window. Note that it is possible to reopen a closed attached window and not have it linked to its main window.

Shrink The collection of windows shrinks as a group. The SHRINKFNs of the attached windows are evaluated but the only icon displayed is the one for the main window.

Redisplay The main or attached windows can be redisplayed independently.

Totop If any main or attached window is brought to the top, all of the other windows are brought to the top also.

Expand Expanding any of the windows expands the whole collection.

Scrolling All of the windows involved in the group scroll independently.

Clear All windows clear independently of each other.

Window Properties Of Attached Windows

Windows that are involved in a collection either as a main window or as an attached window have properties stored on them. The only properties that are intended to be set by the user are the MINSIZE, MAXSIZE, PASSTOMAINCOMS, and REJECTMAINCOMS window properties. The other properties should be considered read only.

MINSIZE
MAXSIZE

[Window Property]
[Window Property]

INTERLISP-D REFERENCE MANUAL

Each of these window properties should be a dotted pair (WIDTH . HEIGHT) or a function to apply to the window that returns a dotted pair. The numbers are used when the main window is reshaped. The MINSIZE is used to determine the size of the smallest region acceptable during reshaping. Any amount greater than the collective minimum is spread evenly among the windows until each reaches MAXSIZE. Any excess is given to the main window.

If you give the main window of an attached window group a MINSIZE or MAXSIZE property, its value is moved to the MAINWINDOWMINSIZE or MAINWINDOWMAXSIZE property, so that the main window can be given a size function that computes the minimum or maximum size of the entire group. Thus, if you want to change the main window's minimum or maximum size after attaching windows to it, you should change the MAINWINDOWMINSIZE or MAINWINDOWMAXSIZE property instead.

This doesn't address the hard problem of overlapping attached windows side to side, for example if window A was attached as [TOP, LEFT] and B as [TOP, RIGHT]. Currently, the attached window functions do not worry about the overlap.

The default MAXSIZE is NIL, which will let the region grow indefinitely.

MAINWINDOW

[Window Property]

Pointer from attached windows to the main window of the group. This link is not available if the main window is closed. The function MAINWINDOW is the preferred way to access this property.

ATTACHEDWINDOWS

[Window Property]

Pointer from a window to its attached windows. The function ATTACHEDWINDOWS is the preferred way to access this property.

WHEREATTACHED

[Window Property]

For attached windows, a dotted pair (EDGE . POSITIONONEDGE) giving the edge and position on the edge that determine how the attached window is placed relative to its main window.

The TOTOPFN window property on attached windows and the properties TOTOPFN, DOSHAPEFN, MOVEFN, CLOSEFN, OPENFN, SHRINKFN, EXPANDFN and CALCULATEREGIONFN on main windows contain functions that implement the attached window manipulation facilities. Care should be used in modifying or replacing these properties.

Communication of Window Menu Commands between Attached Windows is dependent on the name of function used to implement the window command, e.g., CLOSEW implements CLOSE (refer to PASSTOMAINCOMS documentation under Attached Windows). Consequently, if an application intercepts a window command by changing WHENSELECTEDFN for an item in the WindowMenu (for example, to advise the application that a window is being closed), windows may not behave correctly when attached to other windows.

WINDOWS AND MENUS

To get around this problem, the Medley release provides the variable `*attached-window-command-synonyms*`. This variable is an alist, where each element is of the form (new-command-function-name . old-command-function-name).

For example, if an application redefines the WindowMenu to call `my-close-window` when `CLOSE` is selected, that application should:

```
(cl:push '(my-close-window . il:closew) il:*attached-window-command-synonyms*)
```

in order to tell the attached window system that `my-close-window` is a synonym function for `CLOSEW`.

28. HARDCOPY FACILITIES

Interlisp-D includes facilities for generating hardcopy in "Interpress" format and "Press" format. Interpress is a file format used for communicating documents to Xerox Network System printers such as the Xerox 8044 and Xerox 5700. Press is a file format used for communicating documents to Xerox laser Xerographic printers known by the names "Dover", "Spruce", "Penguin", and "Raven". There are also library packages available for supporting other types of printer formats (4045, FX-80, C150, etc.). The hardcopy facilities are designed to allow the user to support new types of printers with minimal changes to the user interface.

Files can be in a number of formats, including Interpress files, plain text files, and formatted Tedit files. In order to print a file on a given printer, it is necessary to identify the format of the file, convert the file to a format that the printer can accept, and transmit it. Rather than require that the user explicitly determine file types and do the conversion, the Interlisp-D hardcopy functions generate Interpress or other format output depending on the appropriate choice for the designated printer. The hardcopy functions use the variables `PRINTERTYPES` and `PRINTFILETYPES` (described below) to determine the type of a file, how to convert it for a given printer, and how to send it. By changing these variables, the user can define other kinds of printers and print to them using the normal hardcopy functions.

`(SEND.FILE.TO.PRINTER FILE HOST PRINTOPTIONS)` [Function]

The function `SEND.FILE.TO.PRINTER` causes the file *FILE* to be sent to the printer *HOST*. If *HOST* is `NIL`, the first host in the list `DEFAULTPRINTINGHOST` which can print *FILE* is used.

PRINTOPTIONS is a property list of the form `(PROP VALUE PROP VALUE . . .)`. The properties accepted depends on the type of printer. For Interpress printers, the following properties are accepted:

`DOCUMENT.NAME` The document name to appear on the header page (a string). Default is the full name of the file.

`DOCUMENT.CREATION.DATE` The creation date to appear on the header page (a Lisp integer date, such as returned by `IDATE`). The default value is the creation date of the file.

`SENDER.NAME` The name of the sender to appear on the header page (a string). The default value is the name of the user.

`RECIPIENT.NAME` The name of the recipient to appear on the header page (a string). The default is none.

`MESSAGE` An additional message to appear on the header page (a string). The default is none.

INTERLISP-D REFERENCE MANUAL

- #COPIES** The number of copies to be printed. The default value is 1.
- PAGES . TO . PRINT** The pages of the document that should be printed, represented as a list (FIRSTPAGE# LASTPAGE#). For example, if this option is (3 5), this specifies that pages 3 through 5, inclusive, should be printed. Note that the page numbering used for this purpose has no connection to any page numbers that may be printed on the document. The default is to print all of the pages in the document.
- MEDIUM** The medium on which the master is to be printed. If omitted, this defaults to the value of NSPRINT.DEFAULT.MEDIUM, as follows: NIL means to use the printer's default; T means to use the first medium reported available by the printer; any other value must be a Courier value of type MEDIUM. The format of this type is a list (PAPER (KNOWN.SIZE TYPE)) or (PAPER (OTHER.SIZE (WIDTH LENGTH))). The paper TYPE is one of US.LETTER, US.LEGAL, A0 through A10, ISO.B0 through ISO.B10, and JIS.B0 through JIS.B10. For users who use A4 paper exclusively, it should be sufficient to set NSPRINT.DEFAULT.MEDIUM to (PAPER (KNOWN.SIZE "A4")).
- When using different paper sizes, it may be necessary to reset the variable DEFAULTPAGEREGION, the region on the page used for printing (measured in microns from the lower-left corner).
- STAPLE?** True if the document should be stapled.
- #SIDES** 1 or 2 to indicate that the document should be printed on one or two sides, respectively. The default is the value of EMPRESS#SIDES.
- PRIORITY** The priority of this print request, one of LOW, NORMAL, or HIGH. The default is the printer's default.

Note: Press printers only recognize the options #COPIES, #SIDES, DOCUMENT.CREATION.DATE, and DOCUMENT.NAME.

For example,

```
(SEND.FILE.TO.PRINTER 'FOO NIL
  (#COPIES 3 #SIDES 2 DOCUMENT.NAME "For John"))
```

SEND.FILE.TO.PRINTER calls PRINTERTYPE and PRINTFILETYPE to determine the printer type of *HOST* and the file format of *FILE*. If *FILE* is a formatted file already in a form that the printer can print, it is transmitted directly. Otherwise, CONVERT.FILE.TO.TYPE.FOR.PRINTER is called to do the conversion. [Note: If the file is converted, PRINTOPTIONS is passed to the formatting function, so it can include properties such as HEADING, REGION, and FONTS.] All of these functions use

HARDCOPY FACILITIES

the lists `PRINTERTYPES` and `PRINTFILETYPES` to actually determine how to do the conversion.

`LISTFILES` (Chapter 17) calls the function `LISTFILES1` to send a single file to a hardcopy printing device. Interlisp-D is initialized with `LISTFILES1` defined to call `SEND.FILE.TO.PRINTER`.

(**HARDCOPYW** *WINDOW/BITMAP/REGION FILE HOST SCALEFACTOR ROTATION*
PRINTERTYPE HARDCOPYTITLE) [Function]

Creates a hardcopy file from a bitmap and optionally sends it to a printer. Note that some printers may have limitations concerning how big or how "complicated" the bitmap may be printed.

WINDOW/BITMAP/REGION can either be a *WINDOW* (open or closed), a *BITMAP*, or a *REGION* (interpreted as a region of the screen). If *WINDOW/BITMAP/REGION* is `NIL`, the user is prompted for a screen region using `GETREGION`.

If *FILE* is non-`NIL`, it is used as the name of the file for output. If *HOST* = `NIL`, this file is not printed. If *FILE* is `NIL`, a temporary file is created, and sent to *HOST*.

To save an image on a file without printing it, perform (`HARDCOPYW IMAGE FILE`). To print an image to the printer `PRINTER` without saving the file, perform (`HARDCOPYW IMAGE NIL PRINTER`).

If both *FILE* and *HOST* are `NIL`, the default action is to print the image, without saving the file. The printer used is determined by the argument *PRINTERTYPE* and the value of the variable `DEFAULTPRINTINGHOST`. If *PRINTERTYPE* is non-`NIL`, the first host on `DEFAULTPRINTINGHOST` of the type *PRINTERTYPE* is used. If *PRINTERTYPE* is `NIL`, the first printer on `DEFAULTPRINTINGHOST` that implements the `BITMAPSCALE` (as determined by `PRINTERTYPES`) operation is used, if any. Otherwise, the first printer on `DEFAULTPRINTINGHOST` is used.

The type of hardcopy file produced is determined by *HOST* if non-`NIL`, else by *PRINTERTYPE* if non-`NIL`, else by the value of `DEFAULTPRINTINGHOST`, as described above.

SCALEFACTOR is a reduction factor. If not given, it is computed automatically based on the size of the bitmap and the capabilities of the printer type. This may not be supported for some printers.

ROTATION specifies how the bitmap image should be rotated on the printed page. Most printers (including Interpress printers) only support a *ROTATION* of multiples of 90.

PRINTERTYPE specifies what type of printer to use when *HOST* is `NIL`. `HARDCOPYW` uses this information to select which printer to use or what print file format to convert the output into, as described above.

The background menu contains a "Hardcopy" command (Chapter 28) that prompts the user for a region on the screen, and sends the image to the default printer.

INTERLISP-D REFERENCE MANUAL

Hardcopy output may also be obtained by writing a file on the printer device LPT, e.g. (COPYFILE 'FOO '{LPT})). When a file on this device is closed, it is converted to Interpress or some other format (if necessary) and sent to the default printer (the first host on DEFAULTPRINTINGHOST). One can include the printer name directly in the file name, e.g. (COPYFILE 'FOO {LPT}TREMOR:) will send the file to the printer TREMOR:.

HARDCOPYTITLE is a string specifying a title to print on the page containing the screen image. If NIL, the string "Window Image" is used. To omit a title, specify the null string.

(**PRINTERSTATUS** *PRINTER*) [Function]

Returns a list describing the current status of the printer named *PRINTER*. The exact form of the value returned depends on the type of printer. For InterPress printers, the status describes whether the printer is available or busy or needs attention, and what type of paper is loaded in the printer.

Returns NIL if the printer does not respond in a reasonable time, which can occur if the printer is very busy, or does not implement the printer status service.

DEFAULTPRINTINGHOST [Variable]

The variable DEFAULTPRINTINGHOST is used to designate the default printer to be used as the output of printing operations. It should be a list of the known printer host names, for example, (QUAKE LISPPRINT:). If an element of DEFAULTPRINTINGHOST is a list, is interpreted as (PRINTERTYPE HOST), specifying both the host type and the host name. The type of the printer, which determines the protocol used to send to it and the file format it requires, is determined by the function PRINTERTYPE.

If DEFAULTPRINTINGHOST is a single printer name, it is treated as if it were a list of one element.

(**PRINTFILETYPE** *FILE* —) [Function]

Returns the format of the file *FILE*. Possible values include INTERPRESS, TEDIT, etc. If it cannot determine the file type, it returns NIL. Uses the global variable PRINTFILETYPES.

(**PRINTERTYPE** *HOST*) [Function]

Returns the type of the printer *HOST*. Currently uses the following heuristic:

1. If *HOST* is a list, the CAR is assumed to be the printer type and CADR the name of the printer
2. If *HOST* is a litatom with a non-NIL PRINTERTYPE property, the property value is returned as the printer type

3. If *HOST* contains a colon (e.g., `PRINTER : PARC : XEROX`) it is assumed to be an INTERPRESS printer
4. If *HOST* is the CADR of a list on `DEFAULTPRINTINGHOST`, the CAR is returned as the printer type
5. Otherwise, the value of `DEFAULTPRINTERTYPE` is returned as the printer type.

Low-level Hardcopy Variables

The following variables are used to define how Interlisp should generate hardcopy of different types. The user should only need to change these variables when it is necessary to access a new type of printer, or define a new hardcopy document type (not often).

PRINTERTYPES

[Variable]

The characteristics of a given printer are determined by the value of the list `PRINTERTYPES`. Each element is a list of the form

```
(TYPES (PROPERTY VALUE) (PROPERTY VALUE)
...)
```

`TYPES` is a list of the printer types that this entry addresses. The `(PROPERTY VALUE)` pairs define properties associated with each printer type.

The printer properties include the following:

<code>CANPRINT</code>	Value is a list of the file types that the printer can print directly.
<code>STATUS</code>	Value is a function that knows how to find out the status of the printer, used by <code>PRINTERSTATUS</code> .
<code>PROPERTIES</code>	Value is a function which returns a list of known printer properties.
<code>SEND</code>	Value is a function which invokes the appropriate protocol to send a file to the printer.
<code>BITMAPSCALE</code>	Value is a function of arguments <code>WIDTH</code> and <code>HEIGHT</code> in bits which returns a scale factor for scaling a bitmap.
<code>BITMAPFILE</code>	Value is a form which, when evaluated, converts a bitmap to a file format that the printer will accept.

Note: The name `8044` is defined on `PRINTERTYPES` as a synonym for the INTERPRESS printer type. The names `SPRUCE`, `PENGUIN`, and `DOVER` are defined on `PRINTERTYPES` as synonyms for the PRESS printer type. The printer types `FULLPRESS` and `RAVEN` are also defined the same as `PRESS`, except that these printer types indicate

INTERLISP-D REFERENCE MANUAL

that the printer is a "Full Press" printer that is able to scale bitmap images, in addition to the normal Press printer facilities.

PRINTFILETYPES

[Variable]

The variable PRINTFILETYPES contains information about various file formats, such as Tedit files and Interpress files. The format is similar to PRINTERTYPES. The properties that can be specified include:

TEST	Value is a function which tests a file if it is of the given type. Note that this function is passed an open stream.
CONVERSION	Value is a property list of other file types and functions that convert from the specified type to the file format.
EXTENSION	Value is a list of possible file extensions for files of this type.

29. TERMINALINPUT/OUTPUT

Most input/output operations in Interlisp can be simply modeled as reading or writing on a linear stream of bytes. However, the situation is much more complex when it comes to controlling the user's "terminal," which includes the keyboard, the mouse, and the display screen. For example, Interlisp coordinates the operation of these separate I/O devices so that the cursor on the screen moves as the mouse moves, and any characters typed by the user appear in the window currently containing a flashing cursor. Most of the time, this system works correctly without need for user modification.

The purpose of this chapter is to describe how to access the low-level controls for the terminal I/O devices. It documents the use of interrupt characters, the keyboard characters that generate interrupts. Then, it describes terminal tables, used to determine the meaning of the different editing characters (character delete, line delete, etc.). Then, the "dribble file" facility that allows terminal I/O to be saved onto a file is presented (see the Dribble Files section below). Finally, the low-level functions that control the mouse and cursor, the keyboard, and the screen are documented.

Interrupt Characters

Errors and breaks can be caused by errors within functions, or by explicitly breaking a function. The user can also indicate his desire to go into a break while a program is running by typing certain control characters known as "interrupt characters". The following interrupt characters are currently enabled in Interlisp-D:

Note: In Interlisp-D with multiple processes, it is not sufficient to say that "the computation" is broken, aborted, etc; it is necessary to specify which process is being acted upon. Usually, the user wants interrupts to occur in the TTY process, which is the one currently receiving keyboard input. However, sometimes the user wants to interrupt the mouse process, if it is currently busy executing a menu command or waiting for the user to specify a region on the screen. Most of the interrupt characters below take place in the mouse process if it is busy, otherwise the TTY process. Control-G can be used to break arbitrary processes. For more information, see Chapter 23.

- Control-B** Causes a break within the mouse process (if busy) or the TTY process. Use Control-G to break a particular process.
- Control-D** Aborts the mouse process (if busy) or the TTY process, and unwinds its stack to the top level. Calls RESET (see Chapter 14).
- Control-E** Aborts the mouse process (if busy) or the TTY process, and unwinds its stack to the last ERRORSET. Calls ERROR! (see Chapter 14).
- Control-G** Pops up a menu listing all of the currently-running processes. Selecting one of the processes will cause a break to take place in that process.
- Control-P** This interrupt is no longer supported in Medley.

INTERLISP-D REFERENCE MANUAL

Control-T Flashes the TTY process' window and prints status information for the TTY process. First it prints "IO wait," "Waiting", or "Running," depending on whether the TTY process is currently in waiting for characters to be typed, waiting for some other reason, or running. Next, it prints the names of the top three frames on the stack, to show what is running. Then, it prints a line describing the percentage of time (since the last control-T) that has been spent running a program, swapping, garbage collecting, doing local disk I/O, etc. For example:

Running in TTYWAITFORINPUT in TTYBIN in TTYIN1

95% Util, 0% Swap, 4% GC

DELETE Clears typeahead in all processes.

The user can disable and/or redefine Interlisp interrupt characters, as well as define new interrupt characters. Interlisp-D is initialized with the following interrupt channels: RESET (**Control-D**), ERROR (**Control-E**), BREAK (**Control-B**), HELP (**Control-G**), PRINTLEVEL (**Control-P**), RUBOUT (**DELETE**), and RAID. Each of these channels independently can be disabled, or have a new interrupt character assigned to it via the function INTERRUPTCHAR described below. In addition, the user can enable new interrupt channels, and associate with each channel an interrupt character and an expression to be evaluated when that character is typed.

(**INTERRUPTCHAR** CHAR TYP/FORM HARDFLG —)

[Function]

Defines CHAR as an interrupt character. If CHAR was previously defined as an interrupt character, that interpretation is disabled.

CHAR is either a character or a character code (see Chapter 2). Note that full sixteen-bit NS characters can be specified as interrupt characters (see Chapter 2). CHAR can also be a value returned from INTERRUPTCHAR, as described below.

If TYP/FORM = NIL, CHAR is disabled.

If TYP/FORM = T, the current state of CHAR is returned without changing or disabling it.

If TYP/FORM is one of the literal atoms RESET, ERROR, BREAK, HELP, PRINTLEVEL, RUBOUT, or RAID, then INTERRUPTCHAR assigns CHAR to the indicated Interlisp interrupt channel, (reenabling the channel if previously disabled).

If the argument TYP/FORM is a symbol designating a predefined system interrupt (RESET, ERROR, BREAK, etc), and HARDFLG is omitted or NIL, then the hardness defaults to the standard hardness of the system interrupt (e.g., MOUSE for the ERROR interrupt).

If TYP/FORM is any other literal atom, CHAR is enabled as an interrupt character that when typed causes the atom TYP/FORM to be immediately set to T.

TERMINAL INPUT/OUTPUT

If *TYP/FORM* is a list, *CHAR* is enabled as a user interrupt character, and *TYP/FORM* is the form that is evaluated when *CHAR* is typed. The interrupt will be hard if *HARDFLG* = *T*, otherwise soft.

(*INTERRUPTCHAR* *T*) restores all Interlisp channels to their original state, and disables all user interrupts.

HARDFLG determines what process the interrupt should run in. If *HARDFLG* is *NIL*, the interrupt will run in the *TTY* process, which is the process currently receiving keyboard input. If *HARDFLG* is *T*, the interrupt will occur in whichever process happens to be running. If *HARDFLG* is *MOUSE*, the interrupt will happen in the mouse process, if the mouse is busy, otherwise in the *TTY* process.

INTERRUPTCHAR returns a value which, when given as the *CHAR* argument to *INTERRUPTCHAR*, will restore things as they were before the call to *INTERRUPTCHAR*. Therefore, *INTERRUPTCHAR* can be used in conjunction with *RESETFORM* or *RESETLST* (see Chapter 14).

INTERRUPTCHAR is undoable.

(**RESET.INTERRUPTS** *PERMITTEDINTERRUPTS* *SAVECURRENT?*) [Function]

PERMITTEDINTERRUPTS is a list of interrupt character settings to be performed, each of the form (*CHAR* *TYP/FORM* *HARDFLG*). The effect of *RESET.INTERRUPTS* is as if (*INTERRUPTCHAR* *CHAR* *TYP/FORM* *HARDFLG*) were performed for each item on *PERMITTEDINTERRUPTS*, and (*INTERRUPTCHAR* *OTHERCHAR* *NIL*) were performed on every other existing interrupt character.

If *SAVECURRENT?* is non-*NIL*, then *RESET.INTERRUPTS* returns the current state of the interrupts in a form that could be passed to *RESET.INTERRUPTS*, otherwise it returns *NIL*. This can be used with a *RESET.INTERRUPTS* that appears in a *RESETFORM*, so that the list is built at "entry", but not upon "exit".

(**LISPINTERRUPTS**) [Function]

Returns the initial default interrupt character settings for Interlisp-D, as a list that *RESET.INTERRUPTS* would accept.

(**INTERRUPTABLE** *FLAG*) [Function]

if *FLAG* = *NIL*, turns interrupts off. If *FLAG* = *T*, turns interrupts on. Value is previous setting. *INTERRUPTABLE* compiles open.

Any interrupt character typed while interrupts are off is treated the same as any other character, i.e., placed in the input buffer, and will not cause an interrupt when interrupts are turned back on.

Terminal Tables

A read table (see Chapter 25) contains input/output information that is media-independent. For example, the action of parentheses is the same regardless of the device from which the input is being performed. A terminal table is an object that contains information that pertains to terminal input/output operations only, such as the character to type to delete the last character or to delete the last line. In addition, terminal tables contain such information as how line-buffering is to be performed, how control characters are to be echoed/printed, whether lowercase input is to be converted to upper case, etc.

Using the functions below, the user may change, reset, or copy terminal tables, or create a new terminal table and install it as the primary terminal table via `SETTERMTABLE`. However, unlike read tables, terminal tables cannot be passed as arguments to input/output functions.

(**GETTERMTABLE** *TTBL*) [Function]

If *TTBL* = `NIL`, returns the primary (i.e., current) terminal table. If *TTBL* is a terminal table, return *TTBL*. Otherwise, generates an **ILLEGAL TERMINAL TABLE** error.

(**COPYTERMTABLE** *TTBL*) [Function]

Returns a copy of *TTBL*. *TTBL* can be a real terminal table, `NIL` (copies the primary terminal table), or `ORIG` (returns a copy of the original system terminal table). Note that `COPYTERMTABLE` is the only function that creates a terminal table.

(**SETTERMTABLE** *TTBL*) [Function]

Sets the primary terminal table to be *TTBL*. Returns the previous primary terminal table. Generates an **ILLEGAL TERMINAL TABLE** error if *TTBL* is not a real terminal table.

(**RESETTERMTABLE** *TTBL FROM*) [Function]

Copies (smashes) *FROM* into *TTBL*. *FROM* and *TTBL* can be `NIL` or a real terminal table. In addition, *FROM* can be `ORIG`, meaning to use the system's original terminal table.

(**TERMTABLEP** *TTBL*) [Function]

Returns *TTBL*, if *TTBL* is a real terminal table, `NIL` otherwise.

Terminal Syntax Classes

A terminal table associates with each character a single "terminal syntax class", one of `CHARDELETE`, `LINEDELETE`, `WORDDELETE`, `RETYPE`, `CTRLV`, `EOL`, and `NONE`. Unlike read table classes, only one character in a particular terminal table can belong to each of the classes (except for the default class `NONE`). When a new character is assigned one of these syntax classes by `SETSYNTAX` (see Chapter 25), the previous character is disabled (i.e., reassigned the syntax class `NONE`), and the value of `SETSYNTAX` is the code for the previous character of that class, if any, otherwise `NIL`.

TERMINAL INPUT/OUTPUT

The terminal syntax classes are interpreted as follows:

CHARDELETE	(Initially BackSpace and Control-A in Interlisp-D) Typing this character deletes the previous character typed. Repeated use of this character deletes successive characters back to the beginning of the line.
LINEDELETE	(Initially Control-Q in Interlisp-D) Typing this character deletes the whole line; it cannot be used repeatedly.
WORDDELETE	(Initially Control-W in Interlisp-D) Typing this character deletes the previous "word", i.e., sequence of non-separator characters.
RETYPE	(Initially Control-R) Causes the line to be retyped as Interlisp sees it (useful when repeated deletions make it difficult to see what remains).
CTRLV	
CNTRLV	(Initially Control-V) When followed by A, B, ... Z, inputs the corresponding control character control-A, control-B, ... control-Z. This allows interrupt characters to be input without causing an interrupt.
EOL	On input from a terminal, the EOL character signals to the line buffering routine to pass the input back to the calling function. It also is used to terminate inputs to READLINE (see Chapter 13). In general, whenever the phrase carriage-return linefeed is used, what is meant is the character with terminal syntax class EOL.
NONE	The terminal syntax class of all other characters.

GETSYNTAX, SETSYNTAX, and SYNTAXP all work on terminal tables as well as read tables (see page X.XX). As with read tables, full sixteen-bit NS characters can be specified in terminal tables (see Chapter 2). When given NIL as a TABLE argument, GETSYNTAX and SYNTAXP use the primary read table or primary terminal table depending on which table contains the indicated CLASS argument. For example, (SETSYNTAX CH 'BREAK) refers to the primary read table, and (SETSYNTAX CH 'CHARDELETE) refers to the primary terminal table. In the absence of such information, all three functions default to the primary read table; e.g., (SETSYNTAX '{ '%[]) refers to the primary read table. If given incompatible CLASS and table arguments, all three functions generate errors. For example, (SETSYNTAX CH 'BREAK TTBL), where TTBL is a terminal table, generates an **ILLEGAL READTABLE** error, and (GETSYNTAX 'CHARDELETE RDTBL) generates an **ILLEGAL TERMINAL TABLE** error.

Terminal Control Functions

(ECHOCHAR CHARCODE MODE TTBL)

[Function]

INTERLISP-D REFERENCE MANUAL

ECHOCHAR sets the "echo mode" of the character *CHARCODE* to *MODE* in the terminal table *TTBL*. The "echo mode" determines how the character is to be echoed or printed. Note that although the name of this function suggests echoing only, it affects all output of the character, both echoing of input and printing of output.

CHARCODE should be a character code. *CHARCODE* can also be a list of characters, in which case *ECHOCHAR* is applied to each of them with arguments *MODE* and *TTBL*. Note that echo modes can be specified for full sixteen-bit NS characters (see Chapter 2).

MODE should be one of the litatoms *IGNORE*, *REAL*, *SIMULATE*, or *INDICATE* which specify how the character should be echoed or printed:

<i>IGNORE</i>	<i>CHARCODE</i> is never printed.
<i>REAL</i>	<i>CHARCODE</i> itself is printed. Some terminals may respond to certain control and meta characters in interesting ways.
<i>SIMULATE</i>	Output of <i>CHARCODE</i> is simulated. For example, control-I (tab) may be simulated by printing spaces. The simulation is machine-specific and beyond the control of the user.
<i>INDICATE</i>	For control or meta characters, <i>CHARCODE</i> is printed as # and/or ↑ followed by the corresponding alphabetic character. For example, Control-A would echo as ↑A, and meta-Control-W would echo as #↑W.

The value of *ECHOCHAR* is the previous echo mode for *CHARCODE*. If *MODE* = *NIL*, *ECHOCHAR* returns the current echo mode without changing it.

Warning: In some fonts, control and meta characters may be used for printable characters. If the echomode is set to *INDICATE* for these characters, they will not print out correctly.

(**ECHOCONTROL** *CHAR MODE TTBL*) [Function]

ECHOCONTROL is an old, limited version of *ECHOCHAR*, that can only specify the echo mode of control characters. *CHAR* is a character or character code. If *CHAR* is an alphabetic character (or code), it refers to the corresponding control character, e.g., (*ECHOCONTROL* 'Z' *INDICATE*) if equivalent to (*ECHOCHAR* (*CHARCODE* ↑Z) 'INDICATE').

(**ECHOMODE** *FLG TTBL*) [Function]

If *FLG* = *T*, turns echoing for terminal table *TTBL* on. If *FLG* = *NIL*, turns echoing off. Returns the previous setting.

Note: Unlike *ECHOCHAR*, this only affects echoing of typed-in characters, not printing of characters.

(**GETECHOMODE** *TTBL*) [Function]

Returns the current echo mode for *TTBL*.

TERMINAL INPUT/OUTPUT

The following functions manipulate the "raise mode," which determines whether lower case characters are converted to upper case when input from the terminal. There is no "raise mode" for input from files.

(**RAISE** *FLG TTBL*) [Function]

Sets the RAISE mode for terminal table *TTBL*. If *FLG* = NIL, all characters are passed as typed. If *FLG* = T, input is echoed as typed, but lowercase letters are converted to upper case. If *FLG* = 0, input is converted to uppercase before it is echoed. Returns the previous setting.

(**GETRAISE** *TTBL*) [Function]

Returns the current RAISE mode for *TTBL*.

(**DELETEDCONTROL** *TYPE MESSAGE TTBL*) [Function]

Specifies the output protocol when a CHARDELETE or LINEDELETE is typed, by specifying character strings to print when characters are deleted.

Interlisp-10 (designed for use on hardcopy terminals) echos the characters being deleted, preceding the first by a \ and following the last by a \, so that it is easy to see exactly what was deleted. Interlisp-D is initially set up to physically erase the deleted characters from the display, so the DELETEDCONTROL strings are initialized to the null string.

The various values of *TYPE* specify different phases of the deletion, as follows:

1STCHDEL	<i>MESSAGE</i> is the message printed the first time CHARDELETE is typed. Initially "\ " in Interlisp-10.
NTHCHDEL	<i>MESSAGE</i> is the message printed when the second and subsequent CHARDELETE characters are typed (without intervening characters). Initially "" in Interlisp-10.
POSTCHDEL	<i>MESSAGE</i> is the message printed when input is resumed following a sequence of one or more CHARDELETE characters. Initially "\ " in Interlisp-10.
EMPTYCHDEL	<i>MESSAGE</i> is the message printed when a CHARDELETE is typed and there are no characters in the buffer. Initially "## CR" in Interlisp-10.
ECHO	If <i>TYPE</i> = ECHO, the characters deleted by CHARDELETE are echoed. <i>MESSAGE</i> is ignored.
NOECHO	If <i>TYPE</i> = NOECHO, the characters deleted by CHARDELETE are not echoed. <i>MESSAGE</i> is ignored.
LINEDELETE	<i>MESSAGE</i> is the message printed when the LINEDELETE character is typed. Initially "## CR".

INTERLISP-D REFERENCE MANUAL

Note: In Interlisp-10, the `LINEDELETE`, `1STCHDEL`, `NTHCHDEL`, `POSTCHDEL`, and `EMPTYCHDEL` messages must be 4 characters or fewer in length.

`DELETECONTROL` returns the previous message as a string. If `MESSAGE` = `NIL`, the value returned is the previous message without changing it. For `TYPE` = `ECHO` and `NOECHO`, the value of `DELETECONTROL` is the previous echo mode, i.e., `ECHO` or `NOECHO`.

(**GETDELETECONTROL** *TYPE TTBL*) [Function]

Returns the current `DELETECONTROL` mode for *TYPE* in *TTBL*.

Line-Buffering

Characters typed at the terminal are stored in two buffers before they are passed to an input function. All characters typed in are put into the low-level "system buffer", which allows type-ahead. When an input function is entered, characters are transferred to the "line buffer" until a character with terminal syntax class `EOL` appears (or, for calls from `READ`, when the count of unbalanced open parentheses reaches 0). Note that `PEEKc` is an exception; it returns the character immediately when its second argument is `NIL`. Until this time, the user can delete characters one at a time from the line buffer by typing the current `CHARDELETE` character, or delete the entire line buffer back to the last carriage-return by typing the current `LINEDELETE`.

This line editing is not performed by `READ` or `RATOM`, but by Interlisp, i.e., it does not matter (nor is it necessarily known) which function will ultimately process the characters, only that they are still in the Interlisp line buffer. However, the function that is requesting input at the time the buffering starts does determine whether parentheses counting is observed. For example, if a program performs `(PROGN (RATOM) (READ))` and the user types in "A (B C D)", the user must type in the carriage-return following the right parenthesis before any action is taken, because the line buffering is happening under `RATOM`. If the program had performed `(PROGN (READ) (READ))`, the line-buffering would be under `READ`, so that the right parenthesis would terminate line buffering, and no terminating carriage-return would be required.

Once a carriage-return has been typed, the entire line is "available" even if not all of it is processed by the function initiating the request for input. If any characters are "left over", they are returned immediately on the next request for input. For example, `(LIST (RATOM) (READc) (RATOM))` when the input is "A Bcr" returns the three-element list (A % B) and leaves the carriage-return in the buffer.

If a carriage-return is typed when the input under `READ` is not "complete" (the parentheses are not balanced or a string is in progress), line buffering continues, but the lines completed so far are not available for editing with `CHARDELETE` or `LINEDELETE`.

The function `CONTROL` is available to defeat line-buffering:

(**CONTROL** *MODE TTBL*) [Function]

TERMINAL INPUT/OUTPUT

If *MODE* = T, eliminates Interlisp's normal line-buffering for the terminal table *TTBL*. If *MODE* = NIL, restores line-buffering (normal). When operating with a terminal table in which (CONTROL T) has been performed, characters are returned to the calling function without line-buffering as described below.

CONTROL returns its previous setting.

(GETCONTROL *TTBL*)

[Function]

Returns the current control mode for *TTBL*.

The function that initiates the request for input determines how the line is treated when (CONTROL T) is in effect:

READ If the expression being typed is a list, the effect is the same as though done with (CONTROL NIL), i.e., line-buffering continues until a carriage-return or matching parentheses. If the expression being typed is not a list, it is returned as soon as a break or separator character is encountered, e.g., (READ) when the input is "ABC<space>" immediately returns ABC. CHARDELETE and LINEDELETE are available on those characters still in the buffer. Thus, if a program is performing several reads under (CONTROL T), and the user types "NOW IS THE TIME" followed by Control-Q, only TIME is deleted, since the rest of the line has already been transmitted to READ and processed.

An exception to the above occurs when the break or separator character is an opening parenthesis, bracket or double-quote, since returning at this point would leave the line buffer in a "funny" state. Thus if the input to (READ) is "ABC(", the ABC is not read until a carriage-return or matching parentheses is encountered. In this case the user could LINEDELETE the entire line, since all of the characters are still in the buffer.

RATOM Characters are returned as soon as a break or separator character is encountered. Until then, LINEDELETE and CHARDELETE may be used as with READ. For example, (RATOM) followed by "ABC<control-A><space>" returns AB. (RATOM) followed by "<control-A>" returns (and types ## indicating that control-A was attempted with nothing in the buffer, since the (is a break character and would therefore already have been read.

READC

PEEKC The character is returned immediately; no line editing is possible. In particular, (READC) is perfectly happy to return

INTERLISP-D REFERENCE MANUAL

the CHARDELETE or LINEDELETE characters, or the ESCAPE character (%).

The system buffer and line buffer can be directly manipulated using the following functions.

(**CLEARBUF** *FILE* *FLG*) [Function]

Clears the input buffer for *FILE*. If *FILE* is T and *FLG* is T, the contents of Interlisp's system buffer and line buffer are saved (and can be obtained via SYSBUF and LINBUF described below).

When you type Control-D or Control-E, or any of the interrupt characters that require terminal interaction (Control-G, or Control-P), Interlisp automatically performs (CLEARBUF T T). For Control-P and, when the break is exited normally, control-H, Interlisp restores the buffer after the interaction.

The action of (CLEARBUF T), i.e., clearing of typeahead, is also available as the RUBOUT interrupt character, initially assigned to the delete key in Interlisp-D. Note that this interrupt clears both buffers at the time it is typed, whereas the action of the CHARDELETE and LINEDELETE character occur at the time they are read.

(**SYSBUF** *FLG*) [Function]

If *FLG* = T, returns the contents of the system buffer (as a string) that was saved at the last (CLEARBUF T T). If *FLG* = NIL, clears this internal buffer.

(**LINBUF** *FLG*) [Function]

Same as SYSBUF for the line buffer.

If both the system buffer and Interlisp's line buffer are empty, the internal buffers associated with LINBUF and SYSBUF are not changed by a (CLEARBUF T T).

(**BKSYSBUF** *X* *FLG* *RDTBL*) [Function]

BKSYSBUF appends the PRIN1-name of *X* to the system buffer. The effect is the same as though the user typed *X*. Some implementations have a limit on the length of *X*, in which case characters in *X* beyond the limit are ignored. Returns *X*.

If *FLG* is T, then the PRIN2-name of *X* is used, computed with respect to the readable *RDTBL*. If *RDTBL* is NIL or omitted, the current readable of the TTY process (which is to receive the characters) is used. Use this for copy selection functions that want their output to be a readable expression in an Exec.

Note that if you are typing at the same time as the BKSYSBUF is being performed, the relative order of the typein and the characters of *X* is unpredictable.

(**BKLINBUF** *STR*) [Function]

TERMINAL INPUT/OUTPUT

STR is a string. `BKLINBUF` sets Interlisp's line buffer to *STR*. Some implementations have a limit on the length of *STR*, in which case characters in *STR* beyond the limit are ignored. Returns *STR*.

(BKSYSCHARCODE CODE)

[Function]

This function appends the character code *CODE* to the system input buffer. The function `BKSYSBUF` is implemented by repeated calls to `BKSYSCHARCODE`.

`BKLINBUF`, `BKSYSBUF`, `LINBUF`, and `SYSBUF` provide a way of "undoing" a `CLEARBUF`. Thus to "peek" at various characters in the buffer, one could perform `(CLEARBUF T T)`, examine the buffers via `LINBUF` and `SYSBUF`, and then put them back.

The more common use of these functions is in saving and restoring typeahead when a program requires some unanticipated (from the user's standpoint) input. The function `RESETBUFS` provides a convenient way of simply clearing the input buffer, performing an interaction with the user, and then restoring the input buffer.

(RESETBUFS FORM , FORM , ... FORMN

[NLambda NoSpread Function]

Clears any typeahead (ringing the terminal's bell if there was, indeed, typeahead), evaluates *FORM* , *FORM* , ... *FORM* , then restores the typeahead. Returns the value of *FORM* . Compiles open.

Dribble Files

A dribble file is a "transcript" of all of the input and output on a terminal. In Interlisp-D, `DRIBBLE` opens a dribble file for the current process, recording the terminal input and output for that process. Multiple processes can have separate dribble files open at the same time.

(DRIBBLE FILE APPENDFLG THAWEDFLG)

[Function]

Opens *FILE* and begins recording the typescript. Returns the old dribble file if any, otherwise `NIL`. If *APPENDFLG* = `T`, the typescript will be appended to the end of *FILE*. If *THAWEDFLG* = `T`, the file will be opened in "thawed" mode, for those implementations that support it. `(DRIBBLE)` closes the dribble file for the current process. Only one dribble file can be active for each process at any one time, so `(DRIBBLE FILE1)` followed by `(DRIBBLE FILE2)` will cause *FILE1* to be closed.

(DRIBBLEFILE)

[Function]

Returns the name of the current dribble file for the current process, if any, otherwise `NIL`.

INTERLISP-D REFERENCE MANUAL

Terminal input is echoed to the dribble file a line buffer at a time. Thus, the typescript produced is somewhat neater than that appearing on the user's terminal, because it does not show characters that were erased via Control-A or Control-Q. Note that the typescript file is not included in the list of files returned by (OPENP), nor will it be closed by a call to CLOSEALL or CLOSEF. Only (DRIBBLE) closes the typescript file.

Cursor and Mouse

A mouse is a small box connected to the computer keyboard by a long wire. On the top of the mouse are two or three buttons. On the bottom is a rolling ball or a set of photoreceptors, to detect when the mouse is moved. As the mouse is moved on a surface, a small image on the screen, called the cursor, moves to follow the movement of the mouse. By moving the mouse, the user can cause the cursor to point to any part of the display screen.

The mouse and cursor are an important part of the Interlisp-D user interface. The Interlisp-D window system allows the user to create, move, and reshape windows, and to select items from displayed menus, all by moving the mouse and clicking the mouse buttons. This section describes the low-level functions used to control the mouse and cursor.

Changing the Cursor Image

Interlisp-D maintains the image of the cursor on the screen, moving it as the mouse is moved. The bitmap that becomes visible as the cursor can be accessed by the following function:

(CURSORBITMAP) [Function]

Returns the cursor bitmap.

CURSORWIDTH [Variable]

CURSORHEIGHT [Variable]

Value is the width and height of the cursor bitmap, respectively.

The cursor bitmap can be changed like any other bitmap by BITBLTing into it or pointing a display stream at it and printing or drawing curves. The CURSOR datatype has the following field names CUBITSPERPIXEL CUIIMAGE, CUMASK, CUHOTSPOTX, CUHOTSPOTY, CUDATA

CURSOR objects can be saved on a file using the file package command CURSORS, or the UGLYVARS file package command.

(CURSORCREATE BITMAP X Y) [Function]

Returns a cursor object which has BITMAP as its image and the location (X, Y) as the hot spot. If X is a POSITION, it is used as the hot spot. If BITMAP has dimensions different from CURSORWIDTH by CURSORHEIGHT, the lesser of the widths and the lesser of the

TERMINAL INPUT/OUTPUT

heights are used to determine the bits that actually get copied into the lower left corner of the cursor. If *X* is NIL, 0 is used. If *Y* is NIL, *CURSORHEIGHT*-1 is used. The default cursor is an uparrow with its tip in the upper left corner and its hot spot at (0, *CURSORHEIGHT*-1).

(**CURSOR** *NEWCURSOR* -) [Function]

Returns a **CURSOR** record instance that contains (a copy of) the current cursor specification. If *NEWCURSOR* is a **CURSOR** record instance, the cursor will be set to the values in *NEWCURSOR*. If *NEWCURSOR* is T, the cursor will be set to the default cursor

DEFAULTCURSOR, an upward left pointing arrow: 

(**SETCURSOR** *NEWCURSOR* -) [Function]

If *NEWCURSOR* is a **CURSOR** record instance, the cursor will be set to the values in *NEWCURSOR*. This does not return the old cursor, and therefore, provides a way of changing the cursor without using storage.

(**FLIPCURSOR**) [Function]

Inverts the cursor.

The following list describes the cursors used by the Interlisp-D system. Most of them are stored as the values of various variables.



In variable **DEFAULTCURSOR**. This is the default cursor.



In variable **WAITINGCURSOR**. Represents an hourglass. Used during long computations.



In variable **MOUSECONFIRMCURSOR**. Indicates that the system is waiting for the user to confirm an action by pressing the left mouse button, or aborting the action by pressing any other button. Used by the function **MOUSECONFIRM** (see Chapter 28).



In variable **SYSOUTCURSOR**. Indicates that the system is saving the virtual memory in a sysout file. See **SYSOUT**, Chapter 12.



In variable **SAVINGCURSOR**. Indicates that **SAVEVM** has been called automatically to save the virtual memory state after the system is idle for long enough. See **SAVEVMWAIT**, Chapter 12.



In variable **CROSSHAIRS**. Used by **GETPOSITION** (see Chapter 28) to indicate a position.

INTERLISP-D REFERENCE MANUAL



In variable `BOXCURSOR`. Used by `GETBOXPOSITION` (see Chapter 28) to indicate where to place the corner of a box.



In variable `FORCEPS`. Used by `GETREGION` (see Chapter 28) when the user switches corners.



In variable `EXPANDINGBOX`. Used by `GETREGION` (see Chapter 28) when a box is first displayed.



In variable `UpperRightCursor`.



In variable `LowerRightCursor`.



In variable `UpperLeftCursor`.



In variable `LowerLeftCursor`.

The previous four cursors are used by `GETREGION` (see Chapter 28) to indicate the four corners of a region.



In variable `VertThumbCursor`. Used during scrolling to indicate thumbing in a vertical scroll bar.



In variable `VertScrollCursor`.



In variable `ScrollUpCursor`.



In variable `ScrollDownCursor`.

The previous four cursors are used by `SCROLL.HANDLER` (see Chapter 28) during vertical scrolling.



In variable `HorizThumbCursor`. Used during scrolling to indicate thumbing in a horizontal scroll bar.



In variable `HorizScrollCursor`.



In variable `ScrollLeftCursor`.



In variable `ScrollRightCursor`.

The previous four cursors are used by `SCROLL.HANDLER` (see Chapter 28) during horizontal scrolling.

TERMINAL INPUT/OUTPUT



These cursors are used by the Teleraid low-level debugger. These cursors are not accessible as standard Interlisp-D cursors.

Flashing Bars on the Cursor

The low-level Interlisp-D system uses the cursor to display certain system status information, such as garbage collection or swapping. This is done because changing the cursor image is very quick, and does not require interacting with the window system. Interlisp inverts horizontal bars on the cursor when the system is swapping pages, or doing certain stack operations. Normally, these bars are only inverted for a very short time, so they look like they are flashing. These cursor changes are interpreted as follows:

Inverted cursor:  Whatever image is being displayed as the cursor, whenever Interlisp does a garbage collection, the whole cursor is inverted.

Top bar:  Swap read. On when Interlisp is swapping in a page from the virtual memory file into the real memory. It is also on when Interlisp allocates a new virtual memory page, even though that doesn't involve a disk read. If this is flashing a lot, the system is doing a lot of swapping. This is an indication that the virtual memory working set is fragmented (see Chapter 22). Performance may be improved by reloading a clean Interlisp system.

Upper middle bar:  Stack operations. If this is flashing a lot, it suggests that some process is neglecting to release stack pointers in a timely fashion (see Chapter 11).

Lower middle bar:  Stack operations. On when Interlisp is moving frames on the stack. If the system is slow, and this is flashing a lot, `HARDRESET` (see Chapter 23) sometimes helps.

Bottom bar:  Swap write. On when Interlisp writes a dirty virtual memory page from the real memory back into the virtual memory file.

Cursor Position

The position at which the cursor bitmap is being displayed can be read or set using the following functions:

`(CURSORPOSITION NEWPOSITION DISPLAYSTREAM OLDPOSITION)` [Function]

Returns the location of the cursor in the coordinate system of `DISPLAYSTREAM` (or the current display stream, if `DISPLAYSTREAM` is `NIL`). If `NEWPOSITION` is non-`NIL`, it

INTERLISP-D REFERENCE MANUAL

should be a position and the cursor will be positioned at *NEWPOSITION*. If *NEWPOSITION* is NIL, the current position is simply returned.

The current position of the cursor is the position of the "hot spot" of the cursor, not the position of the cursor bitmap.

If *OLDPOSITION* is a POSITION object, this object will be changed to point to the location of the cursor and returned, rather than allocating a new POSITION. This can improve performance if CURSORPOSITION is called repeatedly to track the cursor.

To get the location of the cursor in absolute screen coordinates, use the variables LASTMOUSEX and LASTMOUSEY.

(**ADJUSTCURSORPOSITION** *DELTAX* *DELTAY*) [Function]

Moves the cursor *DELTAX* points in the X direction and *DELTAY* points in the Y direction. *DELTAX* and *DELTAY* default to 0.

Mouse Button Testing

There are two or three keys on the mouse. These keys (also called buttons) are referred to by their location: LEFT, MIDDLE, or RIGHT. The following macros are provided to test the state of the mouse buttons:

(**MOUSESTATE** *BUTTONFORM*) [Macro]

Reads the state of the mouse buttons, and returns T if that state is described by *BUTTONFORM*. *BUTTONFORM* can be one of the key indicators LEFT, MIDDLE, or RIGHT; the atom UP (indicating all keys are up); the form (ONLY *KEY*); or a form of AND, OR, or NOT applied to any valid button form.

For example: (MOUSESTATE LEFT) will be true if the left mouse button is down. (MOUSESTATE (ONLY LEFT)) will be true if the left mouse button is the only one down. (MOUSESTATE (OR (NOT LEFT) MIDDLE)) will be true if either the left mouse button is up or the middle mouse button is down.

(**LASTMOUSESTATE** *BUTTONFORM*) [Macro]

Similar to MOUSESTATE, but tests the value of LASTMOUSEBUTTONS (below) rather than getting the current state. This is useful for determining which keys caused MOUSESTATE to be true.

(**UNTILMOUSESTATE** *BUTTONFORM* *INTERVAL*) [Macro]

BUTTONFORM is as described in MOUSESTATE. Waits until *BUTTONFORM* is true or until *INTERVAL* milliseconds have elapsed. The value of UNTILMOUSESTATE is T if *BUTTONFORM* was satisfied before it timed out, otherwise NIL. If *INTERVAL* is NIL, it waits indefinitely. This compiles into an open loop that calls the TTY wait background

TERMINAL INPUT/OUTPUT

function. This form should not be used inside the TTY wait background function. UNTILMOUSESTATE does not use any storage during its wait loop.

Low Level Mouse Functions

This section describes the functions and variables that provide low level access to the mouse and cursor.

(**LASTMOUSEX** *DISPLAYSTREAM*) [Function]

Returns the value of the cursor's X position in the coordinates of *DISPLAYSTREAM* (as of the last call to GETMOUSESTATE, below).

(**LASTMOUSEY** *DISPLAYSTREAM*) [Function]

Returns the value of the cursor's Y position in the coordinates of *DISPLAYSTREAM* (as of the last call to GETMOUSESTATE, below).

LASTMOUSEX [Variable]

Value is the X position of the cursor in absolute screen coordinates (as of the last call to GETMOUSESTATE, below).

LASTMOUSEY [Variable]

Value is the Y position of the cursor in absolute screen coordinates (as of the last call to GETMOUSESTATE, below).

LASTMOUSEBUTTONS [Variable]

Value is an integer that has bits on corresponding to the mouse buttons that are down (as of the last call to GETMOUSESTATE, below). Bit 4Q is the left mouse button, 2Q is the right button, 1Q is the middle button.

LASTKEYBOARD [Variable]

Value is an integer encoding the state of certain keys on the keyboard (as of the last call to GETMOUSESTATE, below). Bit 200Q = lock, 100Q = left shift, 40Q = ctrl, 10Q = right shift, 4Q = blank Bottom, 2Q = blank Middle, 1Q = blank Top. If the key is down, the corresponding bit is on.

(**GETMOUSESTATE**) [Function]

Reads the current state of the mouse and sets the variables LASTMOUSEX, LASTMOUSEY, and LASTMOUSEBUTTONS. In polling mode, the program must remember the previous state and look for changes, such as a key going up or down, or the cursor moving outside a region of interest.

(**DECODEBUTTONS** *BUTTONSTATE*) [Function]

INTERLISP-D REFERENCE MANUAL

Returns a list of the mouse buttons that are down in the state *BUTTONSTATE*. If *BUTTONSTATE* is not a small integer, the value of *LASTMOUSEBUTTONS* (above) is used. The button names that can be returned are: *LEFT*, *MIDDLE*, *RIGHT* (the three mouse keys).

Keyboard Interpretation

For each key on the keyboard and mouse there is a corresponding bit in memory that the hardware turns on and off as the key moves up and down. System-level routines decode the meaning of key transitions according to a table of "key actions", which may be to put particular character codes in the sysbuffer, cause interrupts, change the internal shift / control status, or create events to be placed in the mouse buffer.

(**KEYDOWNP** *KEYNAME*) [Function]

Used to read the instantaneous state of any key, independent of any buffering or pre-assigned key action. Returns T if the key named *KEYNAME* is down at the moment the function is executed.

Most keys are named by the characters on the key-top. Therefore, (*KEYDOWNP* 'a) or (*KEYDOWNP* 'A) returns T if the "A" key is down.

There are a number of keys that do not have standard names printed on them. These can be accessed by special names as follows:

Space	SPACE
Carriage return	CR
Line-feed	LF
Backspace	BS
Tab	TAB
Blank keys on 1132	The 1132 keyboard has three unmarked keys on the right of the normal keyboard. These can be accessed by <i>BLANK-BOTTOM</i> , <i>BLANK-MIDDLE</i> , and <i>BLANK-TOP</i> .
Escape	ESCAPE
Shift keys	<i>LSHIFT</i> for the left shift key, <i>RSHIFT</i> for the right shift key.
Shift lock key	LOCK
Control key	CTRL
Mouse buttons	The state of the mouse buttons can be accessed using <i>LEFT</i> , <i>MIDDLE</i> , and <i>RIGHT</i> .

TERMINAL INPUT/OUTPUT

If *KEYNAME* is a small integer, it is taken to be the internal key number. Otherwise, it is taken to be the name of the key. This means, for example, that the name of the "6" key is not the number 6. Instead, spelled-out names for all the digit keys have been assigned. The "6" key is named *SIX*. It happens that the key number of the "6" key is 2. Therefore, the following two forms are equivalent:

(*KEYDOWNP* ' *SIX*)

(*KEYDOWNP* 2)

(*SHIFTDOWNP* *SHIFT*)

[Function]

Returns T if the internal "shift" flag specified by *SHIFT* is on; NIL otherwise.

If *SHIFT* = 1*SHIFT*, 2*SHIFT*, *LOCK*, *META*, or *CTRL*, *SHIFTDOWNP* returns the state of the left shift, right shift, shift lock, control, and meta flags, respectively.

If *SHIFT* = *SHIFT*, *SHIFTDOWNP* returns T if either the left or right shift flag is on.

If *SHIFT* = *USERMODE1*, *USERMODE2*, or *USERMODE3*, *SHIFTDOWNP* returns the state of one of three user-settable flags that have no other effect on key interpretation. These flags can be set or cleared on character transitions by using *KEYACTION* (below).

(*KEYACTION* *KEYNAME* *ACTIONS* —)

[Function]

Changes the internal tables that define the action to be taken when a key transition is detected by the system keyboard handler. *KEYNAME* is specified as for *KEYDOWNP*. *ACTIONS* is a dotted pair of the form (DOWN-ACTION . UP-ACTION), where the acceptable transition actions and their interpretations are:

NIL

IGNORE Take no action on this transition (the default for up-transitions on all ordinary characters).

(*CHAR* *SHIFTEDCHAR* *LOCKFLAG*)

If a transition action is a three-element list, *CHAR* and *SHIFTEDCHAR* are either character codes or (non-numeric) single-character litatoms standing for their character codes. Note that *CHAR* and *SHIFTEDCHAR* can be full sixteen-bit NS characters (see page X.XX). When the transition occurs, *CHAR* or *SHIFTEDCHAR* is transmitted to the system buffer, depending on whether either of the two shift keys are down.

LOCKFLAG is optional, and may be *LOCKSHIFT* or *NOLOCKSHIFT*. If *LOCKFLAG* is *LOCKSHIFT*, then *SHIFTEDCHAR* will also be transmitted when the *LOCK* shift is down (the alphabetic keys initially specify *LOCKSHIFT*, but the digit keys specify *NOLOCKSHIFT*). For

INTERLISP-D REFERENCE MANUAL

example, (a A LOCKSHIFT) and (61Q ! NOLOCKSHIFT) are the initial settings for the down transitions of the "a" and "1" keys respectively.

1SHIFTUP, 1SHIFTDOWN

2SHIFTUP, 2SHIFTDOWN

CTRLUP, CTRLDOWN

METAUP, METADOWN Change the status of the internal "shift" flags for the left shift, right shift, control, and meta keys, respectively. These shifts affect the interpretation of ordinary key actions. If either of the shifts is down, then SHIFTEDCHARs are transmitted. If the control flag is on, then the the seventh bit of the character code is cleared as characters are transmitted. If the meta flag is on, the the eighth bit of the character code is set (normally cleared) as characters are transmitted. For example, the initial keyactions for the left shift key is (1SHIFTDOWN . 1SHIFTUP) .

LOCKUP, LOCKDOWN, LOCKTOGGLE

Change the status of the internal "shift" flags for the shift lock key. If the lock flag is down, then SHIFTEDCHARs are transmitted if the key action specified LOCKSHIFT. LOCKUP and LOCKDOWN clear and set the shift lock flag, respectively. LOCKTOGGLE complements the flag (turning it off if the flag is on; on if the flag is off).

USERMODE1UP, USERMODE1DOWN, USERMODE1TOGGLE

USERMODE2UP, USERMODE2DOWN, USERMODE2TOGGLE

USERMODE3UP, USERMODE3DOWN, USERMODE3TOGGLE

Change the status of the three user flags USERMODE1, USERMODE2, and USERMODE3, whose status can be determined by calling SHIFTDOWNP (above). These flags have no other effect on key interpretation.

EVENT An encoding of the current state of the mouse and selected keys is placed in the mouse-event buffer when this transition is detected.

KEYACTION returns the previous setting for KEYNAME. If ACTIONS is NIL, returns the previous setting without changing the tables.

(MODIFY.KEYACTIONS *KEYACTIONS* *SAVECURRENT?*)

[Function]

KEYACTIONS is a list of key actions to be set, each of the form (KEYNAME . ACTIONS). The effect of MODIFY.KEYACTIONS is as if (KEYACTION KEYNAME ACTIONS) were performed for each item on *KEYACTIONS*.

TERMINAL INPUT/OUTPUT

If *SAVECURRENT?* is non-NIL, then *MODIFY.KEYACTIONS* returns a list of all the results from *KEYACTION*, otherwise it returns NIL. This can be used with a *MODIFY.KEYACTIONS* that appears in a *RESETFORM*, so that the list is built at "entry", but not upon "exit".

(**METASHIFT** *FLG*)

[NoSpread Function]

If *FLG* is T, changes the keyboard handler (via *KEYACTION*) so as to interpret the "stop" key on the 1108 as a metashift: if a key is struck while the meta is down, it is read with the 200Q bit set. For CHAT users this is a way of getting an "Edit" key on your simulated Datamedia.

If *FLG* is other than NIL or T, it is passed as the *ACTIONS* argument to *KEYACTION*. The reason for this is that if someone has set the "STOP" key to some random behavior, then (*RESETFORM* (*METASHIFT* T) --) will correctly restore that random behavior.

Display Screen

Medley supports a high-resolution bitmap display screen. All printing and drawing operations to the screen are actually performed on a bitmap in memory, which is read by the computer hardware to become visible as the screen. This section describes the functions used to control the appearance of the display screen.

(**SCREENBITMAP**)

[Function]

Returns the screen bitmap.

SCREENWIDTH
SCREENHEIGHT

[Variable]

[Variable]

Value is the width and height of the screen bitmap, respectively.

WHOLEDISPLAY

[Variable]

Value is a region that is the size of the screen bitmap.

The background shade of the display window can be changed using the following function:

(**CHANGEBACKGROUND** *SHADE* —)

[Function]

Changes the background shade of the window system. *SHADE* determines the pattern of the background. If *SHADE* is a texture, then the background is simply painted with it. If *SHADE* is a *BITMAP*, the background is tessellated (tiled) with it to cover the screen. If *SHADE* is T, it changes to the original shade, the value of *WINDOWBACKGROUNDSHADE*. It returns the previous value of the background.

WINDOWBACKGROUNDSHADE

[Variable]

INTERLISP-D REFERENCE MANUAL

Value is the default background shade for the display.

(**VIDEOCOLOR** *BLACKFLG*) [NoSpread Function]

Sets the interpretation of the bits in the screen bitmap. If *BLACKFLG* is *NIL*, a 0 bit will be displayed as white, otherwise a 0 bit will be displayed as black. **VIDEOCOLOR** returns the previous setting. If *BLACKFLG* is not given, **VIDEOCOLOR** will return the current setting without changing anything.

Note: This function only works on the Xerox 1100 and Xerox 1108.

(**VIDEORATE** *TYPE*) [Function]

Sets the rate at which the screen is refreshed. *TYPE* is one of *NORMAL* or *TAPE*. If *TYPE* is *TAPE*, the screen will be refreshed at the same rate as TV (60 cycles per second). This makes the picture look better when video taping the screen. Note: Changing the rate may change the dimensions of the display on the picture tube.

Maintaining the video image on the screen uses cpu cycles, so turning off the display can improve the speed of compute-bound tasks. When the display is off, the screen will be white but any printing or displaying that the program does will be visible when the display is turned back on.

Note: Breaks and *PAGEFULLFN* waiting (see Chapter 28) turn the display on, but users should be aware that it is possible to have the system waiting for a response to a question printed or a menu displayed on a non-visible part of the screen. The functions below are provided to turn the display off.

Note: These functions have no effect on the Xerox 1108 display.

(**SETDISPLAYHEIGHT** *NSCANLINES*) [Function]

Sets the display to only show the top *NSCANLINES* of the screen. If *NSCANLINES* is *T*, resets the display to show the full screen. Returns the previous setting.

(**DISPLAYDOWN** *FORM* *NSCANLINES*) [Function]

Evaluates *FORM* (with the display set to only show the top *NSCANLINES* of the screen), and returns the value of *FORM*. It restores the screen to its previous setting. If *NSCANLINES* is not given, it defaults to 0.

Miscellaneous Terminal I/O

(**RINGBELLS** *N*) [Function]

Flashes (reverse-videos) the screen *N* times (default 1). On the Xerox 1108, this also beeps through the keyboard speaker.

(**PLAYTUNE** *Frequency/Duration.pairlist*) [Function]

TERMINAL INPUT/OUTPUT

On the Xerox 1108, PLAYTUNE plays a sequence of notes through the keyboard speaker. *Frequency/Duration.pairlist* should be a list of dotted pairs (*FREQUENCY* . *DURATION*). PLAYTUNE maps down its argument, beeping the 1108 keyboard buzzer at each frequency for the specified amount of time. Specifying NIL for a frequency means to turn the beeper off the specified amount of time. The units of time are TICKS (Chapter 12), which last about 28.78 microseconds on the Xerox 1108. PLAYTUNE makes no sound on a Xerox 1132. The default "simulate" entry for Control-G (ASCII BEL) on the 1108 uses PLAYTUNE to make a short beep.

PLAYTUNE is implemented using BEEPON and BEEPOFF:

(**BEEPON** *FREQ*) [Function]

On the Xerox 1108, turns on the keyboard speaker playing a note with frequency *FREQ*, measured in Hertz (see Chapter 12). The speaker will continue to play the note until BEEPOFF is called.

(**BEEPOFF**) [Function]

Turns off the keyboard speaker on the Xerox 1108.

(**SETMAINTPANEL** *N*) [Function]

On the Xerox 1108, this sets the four-digit "maintanance panel" display on the front of the computer to display the number *N*.



Venue

Medley for the Novice

Address comments to:
Venue
User Documentation
1549 Industrial Road
San Carlos, CA 94070
415-508-9672

Medley for the Novice

Release 2.0

February 1992

Copyright © 1992 by Venue.

All rights reserved.

Medley is a trademark of Venue.

Xerox® is a registered trademark and InterPress is a trademark of Xerox Corporation.

UNIX® is a registered trademark of UNIX System Laboratories.

PostScript is a registered trademark of Adobe Systems Inc.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

The information in this document is subject to change without notice and should not be construed as a commitment by Venue. While every effort has been made to ensure the accuracy of this document, Venue assumes no responsibility for any errors that may appear.

Text was written and produced with Venue text formatting tools; Xerox printers were used to produce text masters. The typeface is Classic.

1. BRIEF GLOSSARY

The following definitions will acquaint you with general terms used throughout this primer. You will probably want to read through them now, and use this chapter as a reference while you read through the rest of the primer.

advising	A Medley facility for specifying function modifications without necessarily knowing how a particular function works or even what it does. Even system functions can be changed with advising.
argument	A piece of information given to a Lisp function so that it can execute successfully. When a function is explained in the primer, the arguments that it requires will also be given. Arguments are also called Parameters.
atom	The smallest structure in Lisp; like a variable in other programming languages, but can also have a property list and a function definition.
Background Menu	The menu that appears when the mouse is not in any window and the right mouse button is pressed.
binding	The value of a variable. It could be either a local or a global variable. See unbound.
bitmap	A rectangular array of "pixels," each of which is on or off representing one point in the bitmap image.
BREAK	An Lisp function that causes a function to stop executing, open a Break window, and allows you to find out what is happening while the function is halted.
Break Window	A window that opens when an error is encountered while running your program (i.e., when your program has broken). There are tools to help you debug your program from this window. This is explained further in Chapter 14.
browse	To examine a data structure by use of a display that allows you to "move" around within the data structure.
button	(1) (n.) A key on a mouse. (2) (v.t.) To press one of the mouse keys when making a selection.
CAR	A function that returns the head or first element of a list. See CDR.
caret	The small blinking arrowhead that marks where text will appear when it is typed in from the keyboard.
CDR	A function that returns the tail (that is, everything but the first element) of a list. See CAR.

CLISP	A mechanism for augmenting the standard Lisp syntax. One such augmentation included in Interlisp is the iterative statement. See Chapter 9.
cr	Press your Return key.
datatype	(1) The kind of a datum. In Interlisp, there are many system-defined datatypes, e.g., Floating-Point, Integer, Atom, etc. (2) A datatype can also be user-defined. In this case, it is like a record made up from system types and other user-defined datatypes.
DWIM	"Do-what-I-mean." Many errors made by Medley users could be corrected without any information about the purpose of the program or expression in question (e.g., misspellings, certain kinds of parenthesis errors). The DWIM facility is called automatically whenever an error occurs in the evaluation of an Interlisp expression. If DWIM is able to make a correction, the computation continues as though no error had occurred; otherwise, the standard error mechanism is invoked.
error	Occasionally, while a program is running, an error may occur which will stop the computation. Interlisp provides extensive facilities for detecting and handling error conditions, to enable the testing, debugging, and revising of imperfect programs.
evaluate or EVAL	To find the value of a form. For example, if the variable X is bound to 5, we get 5 by evaluating X. Evaluation of a Lisp function involves evaluating the arguments and then applying the function.
Executive Window	This is your main window, where you will run functions and develop your programs. This is the window that the caret is in when you turn on your machine and load Medley.
file package	A set of functions and conventions that facilitate the bookkeeping involved with working in a large system consisting of many source code files and their compiled counterparts. Essentially, the file package keeps track of where things are and what things have changed. It also keeps track of which files have been modified and need to be updated and recompiled.
form	Another way of saying s-expression. A Lisp expression that can be evaluated.
function	A piece of Lisp code that executes and returns a value.
history	The programmer's assistant is built around a memory structure called the history list. The history functions (e.g. FIX, UNDO, REDO) are part of this assistant. These operations allow you to conveniently rework previously specified operations.
History List	As you type on the screen, you will notice a number followed by a slash, followed by another number. The first number is the exec number, the second is the event number. Each number, and the information on that line, is stored sequentially as the History List Using the History List, you

	can easily reexecute lines typed earlier in a work session. See Chapter 2.
icon	A pictorial representation, usually of a shrunken window.
inspector	An interactive display program for examining and changing the parts of a data structure. Medley has inspectors for lists and other data types.
iterative statement	(also called i.s.) A statement in Interlisp that repetitively executes a body of code. For example, <code>(for x from 1 to 5 do (PRINT x))</code> is an i.s.
iterative variable	(also called i.v.) Usually, an iterative statement is controlled by the value that the i.v. takes on. In the iterative statement example above, x is the iterative variable because its value is being changed by each cycle through the loop. All iterative variables are local to the iterative statement where they are defined.
Lisp	Family of languages invented for "list processing." These languages have in common a set of basic primitives for creating and manipulating symbol structures. Interlisp-D is an implementation of the Lisp language together with an environment (set of tools) for programming, and a set of packages that extend the functionality of the system.
list	A collection of atoms and lists; a list is denoted by surrounding its contents with a pair of parentheses.
Masterscope	A program analysis tool. When told to analyze a program, Masterscope creates a database of information about the program. In particular, Masterscope knows which functions call other functions and which functions use which variables. Masterscope can then answer questions about the program and display the information with a browser.
menu	A way of graphically presenting you with a set of options. There are two kinds of menus: pop-up menus are created when needed and disappear after an item has been selected; permanent menus remain on the screen after use until deliberately closed.
mouse	The mouse is the box attached to your keyboard. It controls the movement of the cursor on your screen. As you become familiar with the mouse, you will find it much quicker to use the mouse than the keyboard.
Mouse Cursor	The small arrow on the screen that points to the northwest.
Mouse Cursor Icons	Four types of mouse cursor icons are shown below.
	Wait. The processor is busy.
	The Mouse Confirm Cursor. It appears when you have to confirm that the choice you just made was correct. If it was, press the left button. If the choice was not correct, press the right button to abort.



This means "sweep out" the shape of the window. To do this, move the mouse to a position where you want a corner. Press the left mouse button, and hold it down. Move the mouse diagonally to sketch a rectangle. When the rectangle is the desired size and shape, release the left button.



This is the "move window" prompt. Move the mouse so that the large "ghost" rectangle is in the position where you want the window. When you click the left mouse button, the window will appear at this new location.

NIL

NIL is the Lisp symbol for the empty list. It can also be represented by a left parenthesis followed by a right parenthesis (). It is the only expression in Lisp that is both an atom and a list.

pixel

Pixel stands for "picture element." The computer monitor screen is made up of a rectangular array of pixels. Each pixel corresponds to one bit. When a bit is turned on (i.e., set to 1), the pixel on the screen represented by this bit is black.

pretty printing

Pretty printing refers to the way Lisp functions are printed with special indentation, to make them easier to read. Functions are pretty printed in the structure editor, SEdit (see Chapter 7). You can pretty print uncompiled functions by calling the function PP with the function you would like to see as an argument, i.e. (PP function-name). For an example of this, see Figure 1.5.

```
Exec (INTERLISP)
105+ PP PP
FNS definition for PP:
(DEFINEQ
  (PP
    [NLAMBDA X (* --)
      (DECLARE (LOCALVARS . T))
      (MAPC (NLAMBDA (ARGS X)
        (FUNCTION (LAMBDA (NAME)
          (for TYPE in (TYPESOF NAME NIL '(FIELDS)
            'CURRENT)
            do (CL:FORMAT *TERMINAL-IO* "~A definition for
              ~S:~%" TYPE NAME)
            (SHOWDEF NAME TYPE))
          )
        )
      )
    ]
  )
)
107+
```

Figure 1.5. Example of Pretty Printing Function PP

Programmer's
Assistant

The programmer's assistant accesses the History List to allow you to FIX, UNDO, and/or REDO your previous expressions typed to the executive window (see Chapter 2).

Prompt Window

The narrow black window at the top of the screen. It displays system prompts, or prompts you have developed (see Figure 1.6).



Figure 1.6. Prompt Window

property list	A list of the form (<property-name1> <property-value1> <property-name2> <property-value2>) associated with an atom. It accessed by the functions GETPROP and PUTPROP.
record	A record is a data structure that consists of named "fields". Accessing elements of a record can be separated from the details of how the data structure is actually stored. This eliminates many programming details. A record definition establishes a record template, describing the form of a record. A record instance is an actual record storing data according to a particular record template. (See datatype, second definition.)
Right Button Default Window Menu	This is the menu that appears when the mouse is in a window, and the right mouse button is pressed. It looks like the menu in Figure 1.7. If this menu does not appear when you press the right button of the mouse and the mouse is in the window, move the mouse so that it is pointing to the title bar of the window, and press the right button.

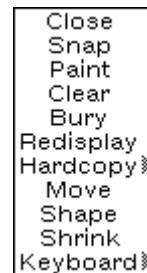


Figure 1.7. Right Button Default Window Menu

s-expression	Short for "symbolic expression". In Lisp, this refers to any well-formed collection of left parentheses, atoms, and right parentheses.
stack	A pushdown list. Whenever a function is entered, information about that specific function call is pushed onto (i.e., added to the front of) the stack. This information includes the variable names and their values associated with the function call. When the function is exited, that data is popped off the stack.
sysout	A file containing a whole Lisp environment: namely, everything you defined or loaded into the environment, the windows that appeared on the screen, the amount of memory used, and so on. Everything is stored in the sysout file exactly as it was when the function SYSOUT was called.

TRACE	A function that creates a trace of the execution of another function. Each time the traced function is called, it prints out the values of the arguments it was called with, and prints out the value it returns upon completion.
unbound	Without value; an atom is unbound if a value has never been assigned to it.
window	A rectangular area of the screen that acts as the main display area for some Lisp process,

PREFACE

It was dawn and the local told him it was down the road a piece, left at the first fishing bridge in the country, right at the appletree stump, and onto the dirt road just before the hill. At midnight he knew he was lost. -Anonymous

Welcome to the Medley Lisp Development Environment, a collection of powerful tools for assisting you in programming in Lisp, developing sophisticated user interfaces, and creating prototypes of your ideas in a quick and easy manner. Unfortunately, along with the power comes mind-numbing complexity. The Medley documentation set describes all the tools in detail, but it would be unreasonable for us to expect a new user to wade through all of it, so this primer is intended as an introduction, to give you a taste of some of the features.

We developed this primer to provide a starting point for new Medley users, to enhance your excitement and challenge you with the potential before you. We're going to make some assumptions about you. For starters, we're going to assume that you're sitting at a workstation that can run Medley. All of the examples in the book figure that you're going to want to try things out. We're also going to assume that you've had some exposure to Lisp, hopefully Common Lisp.

Medley actually consists of two complete Lisp implementations, Common Lisp and InterLisp. All the screen I/O and some of the system functions are in InterLisp. However, thanks to the package system, you can call back and forth between the two languages by simply including a package delimiter in front of a symbol name. This sounds complicated, but it will become clearer once we do some examples.

Throughout we make reference to the *Interlisp-D Reference Manual* by section and page number. The material in the primer is just an introduction. When you need more depth, use the detailed treatment provided in the manual.

While only you can plot your ultimate destination, you will find this primer indispensable for clearly defining and guiding you to the first landmarks on your way.

Acknowledgements

The early inspiration and model for this primer came from the Intelligent Tutoring Systems group and the Learning Research and Development Center at the University of Pittsburgh. We gratefully acknowledge their pioneering contribution to more effective artificial intelligence.

This primer was originally developed by Computer Possibilities, a company committed to making AI technology available. Primary development and writing was done by Cynthia Cosic, with technical writing support provided by Sam Zordich. It has been redone by Venue staff to reflect changes in the environment since the original publication.

At Xerox Artificial Intelligence Systems, John Vittal managed and directed the project. Substantial assistance was provided by many members of the AIS staff who provided both editorial and systems support.

[This page intentionally left blank]

2. TYPING SHORTCUTS

Once you have logged in to Medley, you are in Lisp. The functions you type into the Executive Window will now execute, that is, perform the designated task. Lisp is case-sensitive; it often matters whether text is typed in upper- or lowercase letters. Use the Shift-Lock key on your keyboard to ensure that everything typed is in capital letters.

You must type all Lisp functions in parentheses. The Lisp interpreter will read from the left parenthesis to the closing right parenthesis to determine both the function you want to execute and the arguments to that function. Executing this function is called "evaluation." When the function is evaluated, it returns a value, which is then printed in the Executive Window. This entire process is called the read-eval-print loop, and is how most Lisp interpreters, including the one for Lisp, run.

The prompt in is a number followed by a left-pointing arrow (see Figure 2.3). This number is the function's position on the History List—a list that stores your interactions with the Lisp interpreter. Type the function `(PLUS 3 4)`, and notice the History List assigns to the function (the number immediately to the left of the arrow). Lisp reads in the function and its arguments, evaluates the function, and then prints the number 7.

Programmer's Assistant

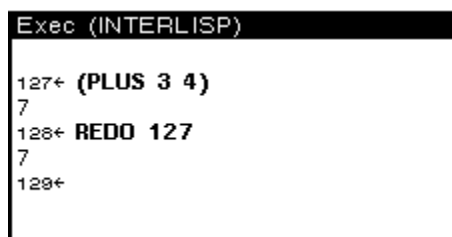
In addition to this read-eval-print loop, there is also a "programmer's assistant." It is the programmer's assistant that prints the number as part of the prompt in the executive window, and uses these numbers to reference the function calls typed after them.

When you issue commands to the programmer's assistant, you will not use parentheses as you do with ordinary function calls. You simply type the command, and some specification that indicates which item on the history list the command refers to. Some programmer's assistant commands are `FIX`, `REDO`, and `UNDO`. They are explained in detail below.

Programmer's assistant commands are useful only at the Lisp top level, that is, when you are typing into the Executive Window. They do not work in user-defined functions.

As an example use of the programmer's assistant, use `REDO` to redo your function call `(PLUS 3 4)`. Type `REDO` at the prompt (programmer's assistant commands can be typed in either upper- or lowercase), then specify the previous expression in one of the following ways:

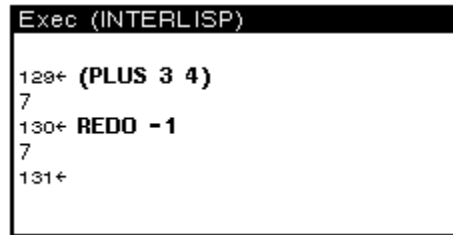
- When you originally typed in the function you now want to refer to, there was a History List number to the left of the arrow in the prompt. Type this number after the programmer's assistant command. This is the method illustrated in Figure 2-1.



```
Exec (INTERLISP)
127← (PLUS 3 4)
7
128← REDO 127
7
129←
```

Figure 2-1. Using a Programmer's Assistant Command to REDO a Function

- A negative number will specify the function call typed in that number of prompts dago. In this example, you would type in -1, the position immediately before the current position. This is shown in Figure 2-2.

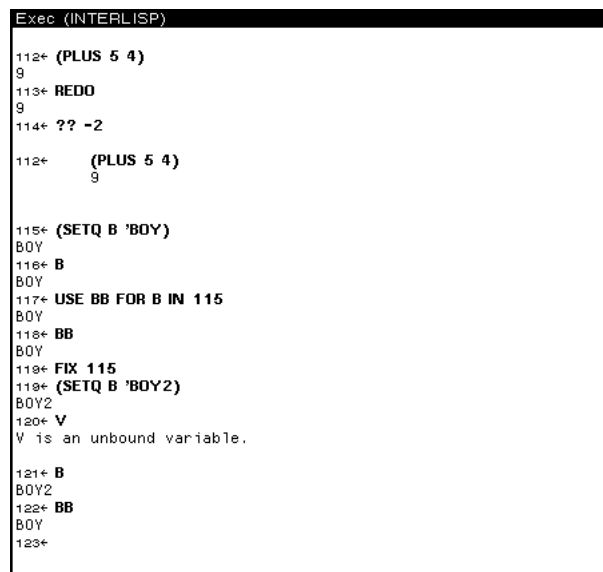


```
Exec (INTERLISP)
129← (PLUS 3 4)
7
130← REDO -1
7
131←
```

Figure 2-2. Using a Negative Number after the Programmer's Assistant Command

- You can also specify the function for the programmer's assistant with one of the items that was in that function call. The programmer's assistant will search backwards in the History List, and use the first function it finds that includes that item. For example, type **REDO PLUS** to have the functiion (PLUS 3 4) reevaluated.
- If you type a programmer's assistant command without specifying a function (i.e., simply typing the command, following by a Return), the programmer's assistant executes the command using the function entered at the previous prompt.

Figure 2-3 shows a few more examples of how to use the programmer's assistant.



```
Exec (INTERLISP)
112← (PLUS 5 4)
9
113← REDO
9
114← ?? -2
9
112← (PLUS 5 4)
9
115← (SETQ B 'BOY)
BOY
116← B
BOY
117← USE BB FOR B IN 115
BOY
118← BB
BOY
119← FIX 115
BOY2
119← (SETQ B 'BOY2)
BOY2
120← V
V is an unbound variable.
121← B
BOY2
122← BB
BOY
123←
```

Figure 2-3. Some Applications of the Programmer's Assistant

If You Make a Mistake

Editing in the Executive Window is explained in detail in Chapter 7. In the following section, only a few of the most useful commands are repeated.

To move the caret to a new place in the command being typed, point the mouse cursor at the appropriate position. Then press the left mouse button.

To move the caret back to the end of the command being typed, press Control-X (hold the Control key down, and type **X**).

To delete:

Character behind the caret	Press the Backspace key
Word behind the caret	Press Control-W (hold the Control key down and type W)
Any part of the command	Move the caret to the appropriate place in the command. Hold the right mouse button down and move the the mouse cursor over the text. All of the blackened text between the caret and mouse cursor is deleted when you release the right mouse button.
Entire command	Press Control-U (hold the Control key down and type U)

Deletions can be undone. Just press the **UNDO** key.

To add more text to the line, move the caret to the appropriate position and start to type. Whatever you type will appear at the caret.

[This page intentionally left blank]

TABLE of CONTENTS

Preface	vii
1. Brief Glossary	1-1
2. Typing Shortcuts	
Programmer's Assistant.....	2-1
If You Make a Mistake	2-2
3. Using Menus	
Making a Selection from a Menu	3-1
Explanations of Menu Items	3-2
Submenus	3-2
Summary	3-3
4. How to Use Files	
Types of Files	4-1
Directories	4-1
Directory Options	4-2
Subdirectories	4-2
To See What Files Are Loaded	4-3
Simple Commands for Manipulating Files	4-3
Connecting to a Directory	4-3
File Version Numbers	4-4
5. FileBrowser	
Calling the FileBrowser	5-1
FileBrowser Commands	5-3
6. Those Wondertul Windows!	
Windows Provided by Medley.....	6-1
Creating a Window	6-2
Right Button Default Window Menu	6-2
Explanation of Each Menu Item	6-3
Scrollable Windows	6-4
Other Window Functions	6-5
PROMTPRINT	6-5
WHICHW	6-6

7. Editing and Saving

Defining Functions	7-1
Simple Editing in the Executive Window	7-2
Using the List Structure Editor	7-3
Commenting Functions	7-4
File Functions and Variables: How to See and Save Them	7-5
File Variables	7-5
Saving Interlisp-D on Files	7-5

8. Your Init File

Using the USERGREETFILES Variable	8-1
Making an Init File	8-1

9. Medley Forgiveness: DWIM 9-1**10. Break Package**

Break Windows	10-1
Break Package Examples	10-1
Ways to Stop Execution from the Keyboard (Breaking Lisp)	10-3
Break Menu	10-3
Returning to Top Level	10-4

11. WhatTo Do If ... 11-1**12. Window and Regions**

Windows 12-1	
CREATEW	12-1
WINDOWPROP	12-2
Getting Windows to Do Things	12-3
BUTTONEVENTFN	12-5
Looking at a Window's Properties	12-5
Regions	12-5

13. What Are Menus?

Displaying Menus	13-1
Getting Menus to Do Stuff	13-2
WHENHELDFN and WHENSELECTEDFN Fields of a Menu	13-3
Looking at a Menu's Fields	13-5

14. Bitmaps 14-1

15. Displaystreams

Drawing on a Displaystream	15-1
DRAWUNE	15-1
DRAWTO	15-2
DRAWCIRCLE	15-3
FILLCIRCLE.....	15-1
Locating and Changing Your Position in a Displaystream	15-4
DSPXPOSITION	15-5
DSPYPOSITION	15-5
MOVETO	15-5

16. Fonts

What Makes Up a Font	16-1
Fontdescriptors and FONTCREATE	16-2
Display Fonts	16-3
InterPress Fonts	16-3
Functions for Using Fonts.....	16-4
FONTPROP - Looking at Font Properties.....	16-4
STRINGWIDTH	16-5
DSPFONT- Changing the Font in One Window	16-5
Personalizing Your Font Profile	16-6

17. The Inspector

Calling the Inspector	17-1
Using the Inspector	17-2
Inspector Example	17-2

18. Masterscope

SHOW DATA Command and GRAPHER	18-2

19. Where Does All the Time Go? SPY

How to Use Spy with the SPY Window	19-1
How to Use SPY from the Lisp Top Level	19-2
Interpreting SPY's Results	19-2

20. Free Menus

Free Menu Example	20-1
Parts of a Free Menu Item	20-2
Types of Free Menu Items	20-3

21. The Grapher

Say it with Graphs	21-1
Add a Node	21-2
Add a Link	21-2
Delete a Link	21-2
Delete a Node	21-2
Move a Node	21-2
Making a Graph from a List	21-2
Incorporating Grapher into Your Program	21-2
More of Grapher	21-2

22. Resource Management

Naming Variables and Records	22-1
Some Space and Time Considerations	22-2
Global Variables	22-3
Circular Lists	22-3
When You Run Out of Space	22-4

23. Simple Interactions with the Cursor, a Bitmap, and a Window

GETMOUSESTATE Example Function	23-1
Advising GETMOUSESTATE	23-2
Changing the Cursor	23-2
Functions for Tracing the Cursor	23-3
Running the Functions	23-6

24. Glossary of Global System Variables

Directories	24-1
Flags	24-2
History Lists	24-3
System Menus	24-3
Windows	24-4
Miscellaneous	24-4

25. Other Useful References 25.1**Index** INDEX-1

[This page intentionally left blank]

3. USING MENUS

The purpose of this chapter is to show you how to use menus. Many things can be done more easily using menus, and there are many different menus provided in the Medley environment. Some are "pop-up" menus that are only available until a selection is made, then disappear until they are needed again. An example of one of these is the Background Menu that appears when the mouse is not in any window and the right mouse button is pressed. A background menu is shown in Figure 3-1. Your background menu may have different items on it.



Figure 3-1. Background Menu

Another common pop-up menu is the right button default window menu. This menu is explained more in Chapter 6.

Other menus are more permanent, such as the menu that is always available for use with the Filebrowser. This menu is shown in Figure 3-2., and the specifics of its use with the filebrowser are explained in Chapter 5.

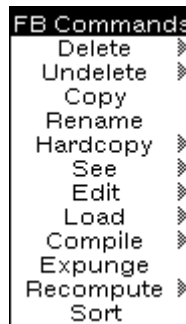


Figure 3-2. Filebrowser Menu

Making a Selection from a Menu

To make a selection from a menu, point with the mouse to the item you would like to select. If one of the mouse buttons is already pressed, the menu item should be highlighted in reverse video. If it is a permanent menu, you must press the left mouse button to highlight the item. When you release the button, the item will be selected. Figure 3-3 shows a menu with the item "Undo" chosen.



Figure 3-3. Menu with the Item "Undo" Chosen

Explanation of Menu Items

Many menu items have explanations associated with the. If you are not sure what the consequences of choosing a particular menu item will be, highlight the menu item but do not release the left mouse button. If the menu item has an explanation associated with it, the explanation will be printed in the prompt window. Figure 3-4 shows the explanation associated with the item "Snap" from the background menu.

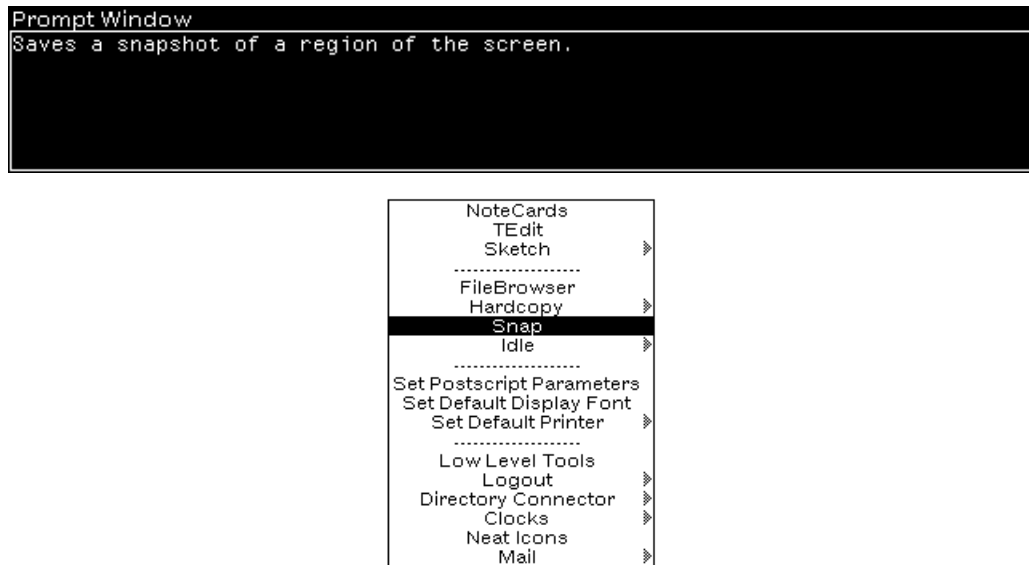


Figure 3-4. Explanation Associated with Selected Menu Item

Submenus

Some menu items have submenus associated with them. This means that, for these items, you can make even more precise choices if you would like to.

A submenu can also be found as described below.

As shown in Figure 3-5, a submenu can be indicated by a gray arrow to the right of the menu item. To see the submenu, highlight the menu item and move the mouse to the right to follow the arrow. Choosing an item from a submenu is done the same way you make a choice from the menu. Any submenus that might be associated with the items in the submenu are indicated in the same way as the submenus associated with the items in the menu.

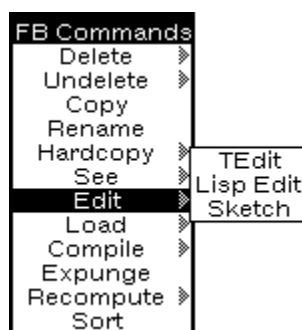


Figure 3-5. Edit Submenu Displayed with Right Arrow

Summary

In summary, here are a few rules of thumb to remember about the interactions of the mouse and system menus:

- Press the left mouse button to select a menu item
- Press the middle mouse button to get more options on a submenu
- Press the right mouse button to see the default right button window menu, and the background menu

[This page intentionally left blank]

4. HOW TO USE FILES

Types of Files

A program file, or Lisp file, contains a series of expressions that can be read and evaluated by the Lisp interpreter. These expressions can include function or macro definitions, variables and their values, properties of variables, and so on. How to save Interlisp-D expressions on these files is explained in Chapter 7. Loading a file is explained in the Simple Commands for Manipulating Files section below.

Not all files, however, have Lisp expressions stored on them. For example, TEdit files store text; sketches are stored on files made with the package Sketch, or can be incorporated into TEdit files. These files are not loaded directly into the environment, but are accessed with the package used to create them, such as TEdit or Sketch.

When you name a file, there are conventions that you should follow. These conventions allow you to tell the type of file by the extension to its name.

If a file contains:	Then:
Lisp expressions	It should not have an extension or have the extension <code>.LISP</code> . For example, a file called MYCODE should contain Lisp expressions.
Compiled Code	It should have the extension <code>.LCOM</code> or <code>.DFASL</code> . For example, a file called MYCODE.DFASL should contain compiled code.
A Sketch	Its extension should be <code>.SKETCH</code> . For example, a file called MOUNTAINS.SKETCH should contain a Sketch.
Text	It should have the extension <code>.TEDIT</code> . For example, a file called REPORT.TEDIT should contain text that can be edited with the editor TEDIT.

Directories

This section focuses on how you can find files, and how you can easily manipulate files. To see all the files listed on a device, use the function `DIR`. For example, to see what files are stored in your current directory, type:

```
(DIR *.*)
```

Partial directory listings can be gotten by specifying a file name, rather than just a device name. The wildcard character `*` can be used to match any number of unknown characters. For example, the command `(DIR T*)` will list the names of all files that begin with the letter T. An example using the wildcard is shown in Figure 4-1.

```
Exec (INTERLISP)

126+ (DIR {DSK}/USERS/PORTER/TMP/D*)

      {DSK}<users>porter>tmp>
DRAFT.TEDIT;1
DRAFT2.TEDIT;1
127+
```

Figure 4-1. Using DIR with a Wildcard

Directory Options

Various words can appear as extra arguments to the DIR command. these words give you extra information about the files.

SIZE displays the size of each file in the directory. For example, type:

```
(DIR {DSK} SIZE)
```

DATE displays the creation date of each file in the directory. An example of this is shown in Figure 4-2.

```
Exec (INTERLISP)

127+ (DIR {DSK}/USERS/PORTER/TMP/D* DATE)

                                CREATIONDATE

      {DSK}<users>porter>tmp>
DRAFT.TEDIT;1      28-Jan-92 11:26:21
DRAFT2.TEDIT;1     28-Jan-92 11:26:22
128+
```

Figure 4-2. Example Using DATE

DEL deletes all the files found by the directory command.

Subdirectories

Subdirectories are very helpful for organizing files. A set of files that have a single purpose (for example, all the external documentation files for a system) can be grouped together into a subdirectory.

To associate a subdirectory with a filename, simply include the desired subdirectory as part of the name of the file. Subdirectories are specified after the device name and before the simple filename. The first subdirectory should be between less-than and greater-than signs (angle brackets) < >, with nested subdirectory names only followed by a greater than sign >. For example:

```
{DSK}<Directory>SubDirectory>SubSubDirectory>...>filename
```

or use the UNIX convention:

```
{DSK}/Directory/Subdirectory/Subsubdirectory/filename
```


To See What Files Are Loaded

If you type `FILELST<CR>`, the names of all the files you loaded will be displayed.

Type `SYSFILES<CR>` to see what files are loaded to create the sysout.

Simple Commands for Manipulating Files

When using these functions, always be sure to specify the full filename, including subfile directories if appropriate.

To have the contents of a file displayed in a window:

`(SEE 'filename)`

To copy a file (see Figure 4-3):

`(COPYFILE 'oldfilename 'newfilename)`



```
Exec (INTERLISP)
136+ (COPYFILE 'TAGREFS.TEDIT 'PRIMERREFS.TEDIT)
{DSK}<users>sybalsky>PRIMERREFS.TEDIT;1
137+
```

Figure 4-3. Example Use of COPYFILE

To delete a file (see Figure 4-4):

`(DELFILE 'filename)`



```
Exec (INTERLISP)
137+ (DELFILE 'TAGREFS.TEDIT)
{DSK}<users>sybalsky>tagrefs.tedit;1
138+
```

Figure 4-4. Example Use of DELFILE

To rename a file:

`(RENAMEFILE 'oldfilename 'newfilename)`

Files that contain Lisp expressions can be loaded into the environment. That means that the information on them is read, evaluated, and incorporated into the Medley environment. To load a file, type:

`(LOAD 'filename)`

Connecting to a Directory

Often, each person or project has a subdirectory where files are stored. If this is your situation, you will want any files you create to be put into this directory automatically. This means you should "connect" to the directory.

CONN is the Medley command that connects you to a directory. For example, CONN in Figure 4-5 connects you to the subsubdirectory IM, in the subdirectory PRIMER, the directory LISPFILES, on the device DSK. This information—the device and the directory names down to the subdirectory to which you want to be connected—is called the "path" to that subdirectory. CONN expects the path to a directory as an argument.

```
Exec (INTERLISP)

139← CONN {dsk}/users/porter/
{DSK}<users>porter>
140←
```

Figure 4-5. CONNecting to Subdirectory Primer Subsubdirectory IM

Once you are connected to a directory, the command DIR will assume you want to see the files in that directory, or any of its subdirectories.

Other commands that require a filename as an argument (e.g., SEE, above) will assume that the file is in the connecteds directory if there is no path specified with the filename. This will often save you typing.

File Version Numbers

When stored, each filename is followed by a semicolon and a number, as shown in this example:

```
MYFILE.TEDIT;1
```

The number is the version number of the file. This is the system's way of protecting your files from being overwritten. Each time the file is written, a new file is created with a version number one greater than the last. This new file will have everything from your previous file, plus all of your changes.

In most cases, you can exclude the version number when referencing the file. When the version is not specified, and there is more than one version of the file on that particular directory, the system generally uses your most recent version. An exception is the function DELFILE, which deletes the oldest version (the one with the lowest version number) if none is specified.

[This page intentionally left blank]

5. FILEBROWSER

The FileBrowser is a Lisp Library Package that works with files stored on disk and floppy devices, and can be used as a file directory editor. If it is not loaded into your sysout, you need to load it first by typing:

```
(LOAD 'FILEBROWSER.LCOM)
```

Calling the FileBrowser

Calling the FileBrowser with the directory calls up the files stored in that directory:

```
(FB '<usr>local>lde>)
```

Another way to call a FileBrowser is to choose "FileBrowser" from the background menu. You will be prompted for a description of the files to be included (see Figure 5-1). Type an asterisk (*), then press Return to see all the files in the connected directory.

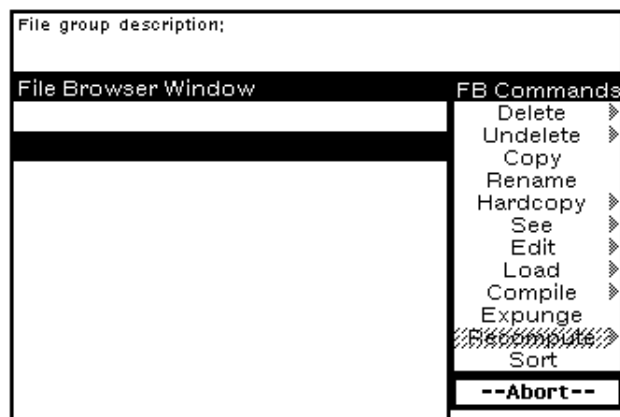


Figure 5-1. Prompt for Files to Include in FileBrowser

These show a directory of the device in a window you can leave on the screen at all times. The parts of the FileBrowser window are shown below.

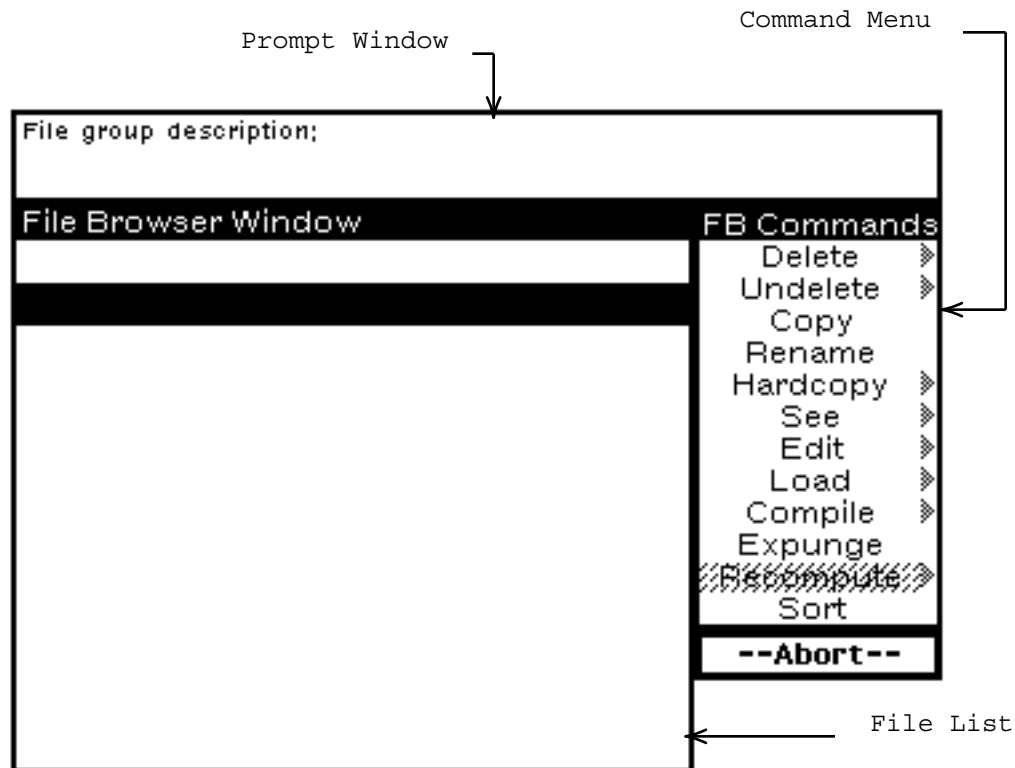


Figure 5-2. Parts of a FileBrowser Window

Now you do not need to continually type the directory command.

To use the FileBrowser, choose a file by pointing to the file with the mouse and pressing the left or middle mouse button. A small dark arrow appears to the left of the file name. Choose a command from the menu at the right. In Figure 5-3, the files OCH1.TEDIT;1, OCH10.TEDIT;1, and OCH11.TEDIT;1 have been selected.

The left mouse button only allows you to choose one file at a time. Even if you choose other files, only the last file you selected with the left mouse button will remain marked as chosen. When you use the middle mouse button to select a file, the file is added to those already chosen.

To unpick an already chosen file, hold the Control key down while pressing the middle mouse button.

{DSK}<users>porter>primer>*.*,* at 1			FB Commands	
Tot: 15 / 29 pgs Del: 0 / 0 pgs			Delete	▶
			Undelete	▶
			Copy	▶
			Rename	▶
			Hardcopy	▶
			See	▶
			Edit	▶
			Load	▶
			Compile	▶
			Expunge	▶
			Recompute	▶
			Sort	▶
			--Abort--	
Name	Pages	Cre		
▶ OCH1.TEDIT;1	1	6-		
▶ OCH10.TEDIT;1	2	6-		
▶ OCH11.TEDIT;1	2	6-		
OCH12.TEDIT;1	2	6-		
OCH13.TEDIT;1	2	6-		
OCH14.TEDIT;1	2	6-		
OCH2.TEDIT;1	2	6-		
OCH3.TEDIT;1	2	6-		
OCH4.TEDIT;1	2	6-		
OCH5.TEDIT;2	2	6-		
OCH5.TEDIT;1	2	6-		

Figure 5-3. Files Chosen

The next section contains a summary of the FileBrowser commands.

FileBrowser Commands

Delete In the FileBrowser, this command marks a file, or files, for deletion (see Figure 5-4). These files are marked by a black line crossing through them. You may select and mark any number of files for deletion. Delete does not actually remove these files from the device. The Expunge command actually wipes out the files previously marked for deletion.

The screenshot shows a FileBrowser window with a title bar that reads "{DSK}<users>porter>primer>*.*,* at 1 FB Commands". Below the title bar, it says "Tot: 15 / 29 pgs Del: 3 / 5 pgs". The main area contains a table with three columns: "Name", "Pages", and "Cre". The table lists several files, some of which are marked for deletion with a black line through the name. To the right of the table is a vertical menu with the following options: Delete, Undelete, Copy, Rename, Hardcopy, See, Edit, Load, Compile, Expunge, Recompute, and Sort. Each option has a right-pointing arrow next to it.

Name	Pages	Cre
▶OCH1.TEDIT;1	1	6-
▶OCH10.TEDIT;1	2	6-
▶OCH11.TEDIT;1	2	6-
OCH12.TEDIT;1	2	6-
OCH13.TEDIT;1	2	6-
OCH14.TEDIT;1	2	6-
OCH2.TEDIT;1	2	6-
OCH3.TEDIT;1	2	6-
OCH4.TEDIT;1	2	6-
OCH5.TEDIT;2	2	6-
OCH5.TEDIT;1	2	6-
-----	-	-

Figure 5-4. Files Marked for Deletion

Undelete Undoes the delete command for one or more files. Undelete erases the black line through a file marked for deletion.

Copy This command copies the chosen file. The destination filename should be typed at a prompt that appears in the window above the FileBrowser. Wildcards do not work for this prompt. You must type the whole unquoted filename. If more than one file is chosen to be copied, you will be prompted for a directory name. The files will be copied into the directory you give, but with the same filenames as the ones they have in their original location.

Rename This command works much like the Copy command, but does not leave the original file. The chosen file will be renamed to the destination filename. You will be prompted, in the prompt window, for the destination filename. Give the complete unquoted filename. If more than one file is chose to be renamed, you will be prompted for a directory name. The files will be moved into the directory you give.

Hardcopy If you do not have a hardcopy device, using this command causes an error. Otherwise, it gives a hardcopy of the file.

See Shows you a file in a window. To use this command, choose a single filename, then the See command. You are prompted for a window. Each time the See command is chosen, a new window is opened to display the file.

Edit Calls the editor with the file as input. If the file is an executable one (i.e., Lisp code as opposed to a documentation file), only the FILECOMS list is edited. The FILECOMS list is the list of variables, lists, and

functions that are contained on that file. FileBrowser loads it and then allows you to edit the FILECOMS.

Load	Choose a file with the left mouse button, or a group of files with the middle mouse button. Once the filenames have been blackened, choose the Load command to load them all into Medley.
Compile	This command calls the file compiler with the chosen filename(s) as arguments. The compiler compiles a file found on a storage device ({DSK}), not the functions defined in the Medley image. If any functions on a loaded file have been changed, run the function (MAKEFILE 'filename) to write the current version before compiling it. Files do not have to be loaded to use the Compile command.
Expunge	This command completely deletes all the marked files from the directory. This allows you to remove unwanted files from your storage device.
Recompute	Choose this command when you know that the directory has been changed and should be reread (e.g., after creating new versions of a file).

6. THOSE WONDERFUL WINDOWS!

A window is a designated area on the screen. Every rectangular box on the screen is a window. While Medley supplies many of the windows (such as the Executive Window), you may also create your own. Among other things, you will type, draw pictures, and save portions of your screen with windows.

Windows Provided by Medley

Two important windows are available as soon as you enter the Medley environment. One is the Executive Window, the main window where you will run your functions. It is the window that the caret is in when you turn on your machine, and load Medley. It is shown in Figure 6-1.



Figure 6-1. Medley Executive Window

The other window that is open when you enter Medley is the "Prompt Window". It is the long thin black window at the top of the screen. It displays system prompts, or prompts you have associated with your programs. (See Figure 6-2.)



Figure 6-2. Prompt Window

Other programs, such as the editors, also use windows. These windows appear when the program starts to run, and close (no longer appear on the screen) when the program is done running.

Creating a Window

To create a new window, type: (**CREATEW**). The mouse cursor will change, and have a small square attached to it. (See Figure 6-3.)



Figure 6-3. Mouse Cursor Asking You to Sweep Out Window

There may be a prompt in the prompt window to create a window. Press and hold the left mouse button. Move the mouse around, and notice that it sweeps out a rectangle. When the rectangle is the size that you'd like your window to be, release the left mouse button. More specific information about the creation of windows, such as giving them titles and specifying their size and position on the screen when they are created, is given in the WINDOWPROP section of Chapter 12.

Right Button Default Window Menu

Position the cursor inside the window you just created, and press and hold the right mouse button. A menu of commands should appear (do not release the right button!), like the one in Figure 6-4. To execute one of the commands on this menu, choose the item. Making a choice from a menu is explained in Chapter 3.

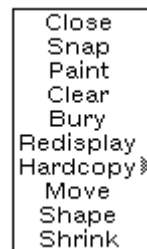


Figure 6-4 Right Button Default Window Menu

As an example, select "Move" from this menu. The mouse cursor will become a ghost window (just an outline of a window, the same size as the one you are moving), with a square attached to one corner, like the one shown in Figure 6-5.



Figure 6-5 Mouse Cursor for Moving a Window

Move the mouse around. The ghost window will follow. Click the left mouse button to place the window in a new location.

Choose "Shape", and notice that you are prompted to sweep out another window. Your original window will have the shape of the window you sketch out.

Explanation of Each Menu Item

The meaning of each right button default window menu item is explained below:

Close	Removes the window from the screen
Snap	Copies a portion of the screen into a new window
Paint	Allows drawing in a window
Clear	Clears the window by erasing everything within the window boundaries
Bury	Puts the window beneath all other windows that overlap it
Redisplay	Redisplays the window contents
Hardcopy	Sends the contents of the window to a printer or to a file
Move	Allows the window to be moved to a new spot on the screen
Shape	Repositions and/or reshapes the window
Shrink	Reduces the window to a small black rectangle called an icon, or, if appropriate, to the shape for that window type (see Figure 6-6).



Figure 6-6 Example Icon

Expand	Changes an icon back to its original window. Position the mouse cursor on the icon, depress the right button, and select Expand. Or, just button the icon with the middle mouse button.

These right-button default window menu selections are available in most windows, including the Executive Window. When the right button has other functions in a window (as in an editor window), the right button default window menu should be accessible by pressing the Right button in the black border at the top of the window.

Scrollable Windows

Some windows in Medley are "scrollable". This means that you can move the contents of the window up and down, or side to side, to see anything that doesn't fit in the window.

Point the mouse cursor to the left or bottom border of a window. If the window is scrollable, a "scroll bar" will appear. The mouse cursor will change to a double headed arrow. (See Figure 6-7.)

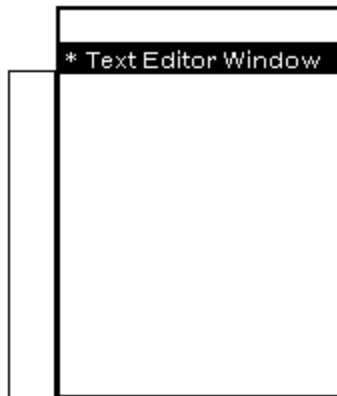


Figure 6-7. Scroll Bar of Scrollable Window

The scroll bar represents the full contents of the window. The example scroll bar is completely white because the window has nothing in it. When a part of the scroll bar is shaded, the amount shaded represents the amount of the window's contents currently shown. If everything is showing, the scroll bar will be fully shaded. (See Figure 6-8.) The position of the shading is also important. It represents the relationship of the section currently displayed to the full contents of the window. For example, if the shaded section is at the bottom of the scroll bar, you are looking at the end of the file.

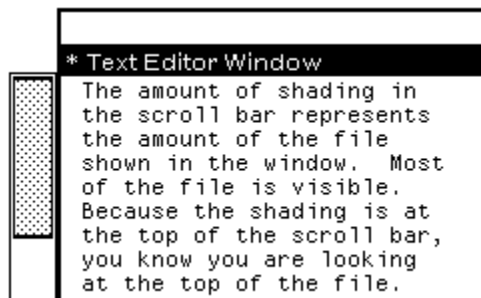


Figure 6-8 Top of File When Shading at Top of Scroll Bar

When the scroll bar is visible, you can control the section of the window's contents displayed:

- To move the contents higher in the window (scroll the contents up in the window), press the left button of the mouse, the mouse cursor changes to look like this:



Figure 6-9. Upward Scrolling Cursor

The contents of the window will scroll up, making the line that the cursor is beside the topmost line in the window.

- To move the contents lower in the window (scroll the contents "down" in the window), press the right button of the mouse, and the mouse cursor changes to look like this:



Figure 6-10. Downward Scrolling Cursor

The contents of the window scroll down, moving the line that is the topmost line in the window to beside the cursor.

- To show a specific section of the window's contents, remember that the scroll bar represents the full contents of the window. Move the mouse cursor to the relative position of the section you want to see (e.g., to the top of the scroll bar if you want to see the top of the window's contents). Press the middle button of the mouse. The mouse cursor will look like this:



Figure 6-11 Proportional Scrolling Cursor

When you release the middle mouse button, the window's contents at that relative position will be displayed.

The position of the mouse in the scroll bar defines how much of the window will be scrolled. If it is near the top, then only a little will be scrolled. If it is near the bottom, most of the window will be scrolled.

Other Window Functions

PROMPTPRINT

Prints an expression to the black prompt window.

For example, type

```
(PROMPTPRINT "THIS WILL BE PRINTED IN THE PROMPT WINDOW")
```

The message will appear in the prompt window. (See Figure 6-12.)

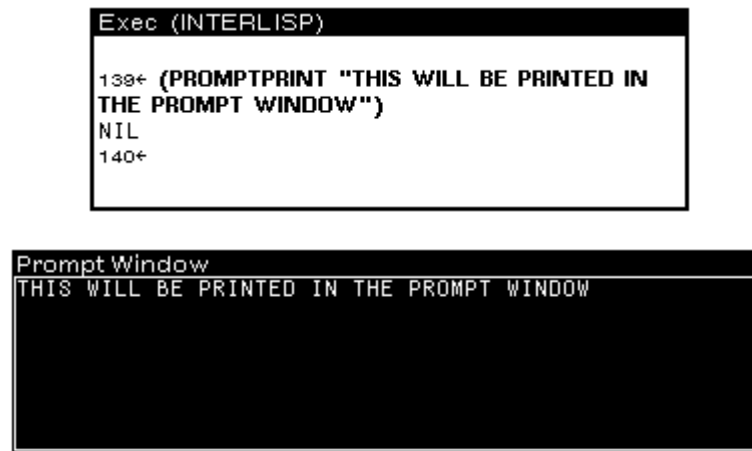


Figure 6-12 PROMPTPRINTing

WHICHW

Returns as a value the name of the window that the mouse cursor IS in.

(WHICHW) can be used as an argument to any function expecting a window, or to reclaim a window that has no name (that is not attached to some particular part of the program.).

7. EDITING AND SAVING

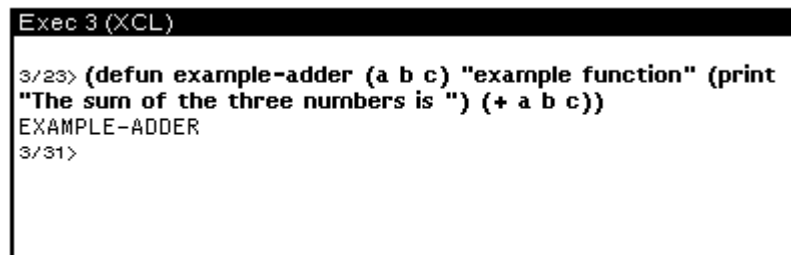
This chapter explains how to define functions, how to edit them, and how to save your work.

Defining Functions

DEFUN can be used to define new functions. The syntax for it is:

```
(DEFUN (<functionname> (<parameter-list><body-of-function>))
```

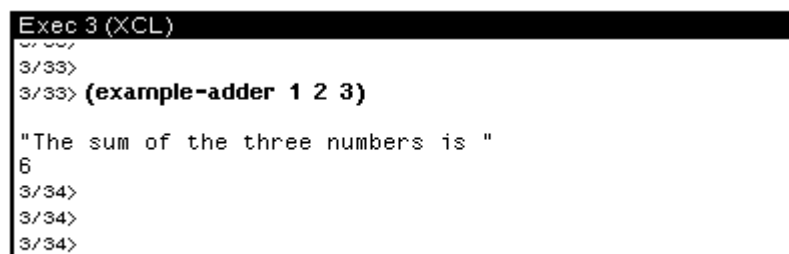
New functions can be created with DEFUN by typing directly into the Executive Window. Once defined, a function is a part of the Medley environment. For example, the function EXAMPLE-ADDER is defined in Figure 7-1.



```
Exec 3 (XCL)
3/23> (defun example-adder (a b c) "example function" (print
"The sum of the three numbers is ") (+ a b c))
EXAMPLE-ADDER
3/31>
```

Figure 7-1. Defining the Function EXAMPLE-ADDER

Now that the function is defined, it can be called from the Executive Window:



```
Exec 3 (XCL)
3/33>
3/33> (example-adder 1 2 3)

"The sum of the three numbers is "
6
3/34>
3/34>
3/34>
```

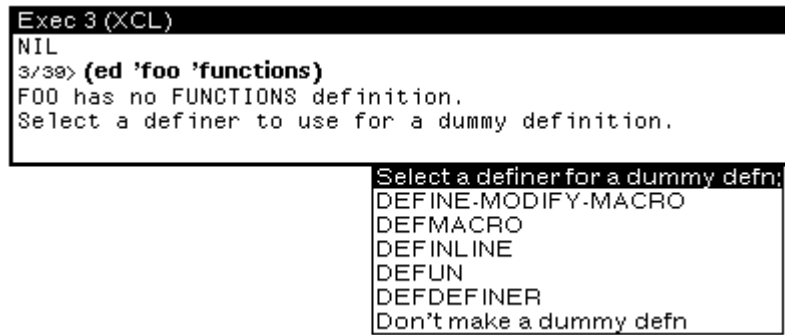
Figure 7-2.. After EXAMPLE-ADDER is defined, it can he executed

The function returns 6, after printing out the message.

Functions can also be defined using the editor DEdit described above. To do this, simply type

```
(ED function-name 'FUNCTIONS)
```

You will be told that no definition exists for the function, and a menu will pop up asking you what type of function you would like to create:



```
Exec 3 (XCL)
NIL
3/38> (ed 'foo 'functions)
FOO has no FUNCTIONS definition.
Select a definer to use for a dummy definition.

Select a definer for a dummy defn:
DEFINE-MODIFY-MACRO
DEFMACRO
DEFINLINE
DEFUN
DEFDEFINER
Don't make a dummy defn
```

Figure 7-3 Selecting a Function Template

Selecting the appropriate type will pop up an editor window with a function template. The use of the editor is explained in the Using the List Structure Editor section below.

Simple Editing in the Executive Window

First, type in an example function to edit:

```
3/41> (defun your-first-function (a b)
      (if (> a b)
          '(the first is greater)
          '(the second is greater)))
```

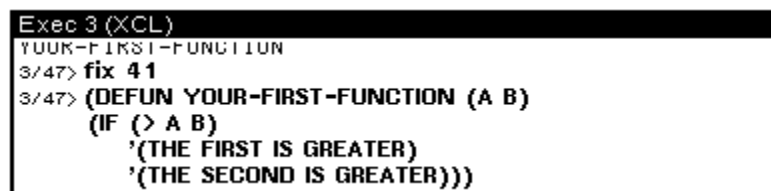
To run the function, type:

```
3/42> (YOUR-FIRST-FUNCTION 3 5)
(THE SECOND IS GREATER)
```

Now, let's alter this. Type:

```
3/43> FIX 41
```

Note that your original function is redisplayed, and ready to edit. (See Figure 7-4.)



```
Exec 3 (XCL)
YOUR-FIRST-FUNCTION
3/47> fix 41
3/47> (DEFUN YOUR-FIRST-FUNCTION (A B)
      (IF (> A B)
          '(THE FIRST IS GREATER)
          '(THE SECOND IS GREATER)))
```

Figure 7-4. Using FIX to Edit a Fundion

Move the text cursor to the appropriate place in the function by positioning the mouse cursor and pressing the left mouse button.

Delete text by moving the caret to the beginning of the section to be deleted. Hold the right mouse button down and move the mouse cursor over the text. All of the highlighted text between the caret and mouse cursor is deleted when you release the right mouse button.

If you make a mistake, deletions can be undone. Press the UNDO key on the keypad to the left of the keyboard.

Now change GREATER to BIGGER:

1. Position the mouse cursor on the G of GREATER, and click the left mouse button. The text cursor is now where the mouse cursor is.
2. Next, press the right mouse button and hold it down. Notice that if you move the mouse cursor around, it will blacken the characters from the text cursor to the mouse cursor. Move the mouse so that the word "GREATER" is highlighted.
3. Release the right mouse button and GREATER is deleted.
4. Without moving the cursor, type in BIGGER.
5. There are two ways to end the editing session and run the function. One is to type Control-X. (Hold the Control key down, and type X.) Another is to move the text cursor to the end of the line and crø In both cases, the function has been edited!

Try the new version of the function by typing:

```
3/48> (YOUR-FIRST-FUNCTION 8 9)
      (THE SECOND IS BIGGER)
```

and get the new result, or you can type:

```
3/49> REDO 42
      (THE SECOND IS BIGGER)
```

Using the List Structure Editor

If the function you want to edit is not readily available (i.e. the function is not in the Executive Window, and you can't remember the history list number, or you simply have a lot of editing), use the List Structure Editor, often called SEdit. This editor is evoked with a call to ED:

```
81←(ED 'YOUR-FIRST-FUNCTION 'FUNCTIONS)
```

Your function will be displayed in an edit window, as in Figure 7-5.

If there is no edit window on the screen, you will be prompted to create a window. As before, hold the left mouse button down, move the mouse until it forms a rectangle of an acceptable size and shape, then release the button. Your function definition will automatically appear in this edit window.

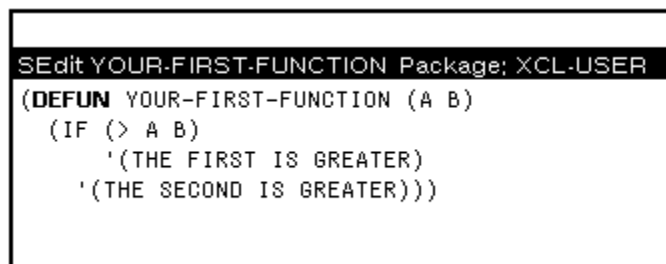
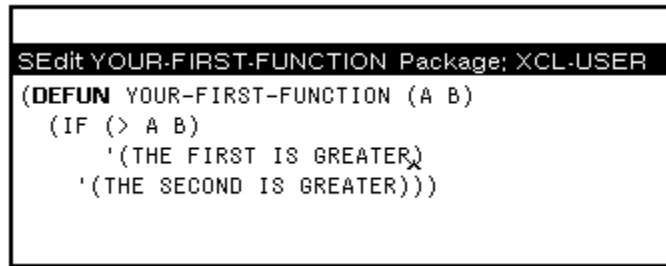


Figure 7-5. An Edit Window

Many changes are easily done with the structure editor. Notice that by pressing the left mouse button you can place the caret in position, and by pressing the middle mouse button you can select atoms or s-expressions. Repeated pressing of the middle button selects bigger pieces of text.

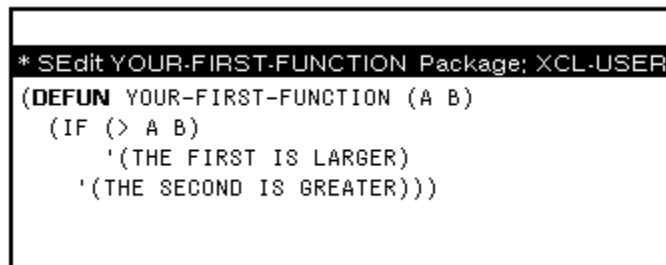
To add an expression that does not appear in the edit window (i.e., it cannot simply be underlined), place the caret at the insertion point and type it in.. For example, to replace the first GREATER with LARGER, place the caret to the left of GREATER, as shown in Figure 7-6.



```
SEdit YOUR-FIRST-FUNCTION Package: XCL-USER
(DEFUN YOUR-FIRST-FUNCTION (A B)
  (IF (> A B)
    '(THE FIRST IS GREATER)
    '(THE SECOND IS GREATER)))
```

Figure 7-6. Caret Placement Prior to Changing GREATER with LARGER

Now press the DELETE key seven times, and type in LARGER. The window now looks like this:



```
* SEdit YOUR-FIRST-FUNCTION Package: XCL-USER
(DEFUN YOUR-FIRST-FUNCTION (A B)
  (IF (> A B)
    '(THE FIRST IS LARGER)
    '(THE SECOND IS GREATER)))
```

Figure 7-7. GREATER Changed to LARGER

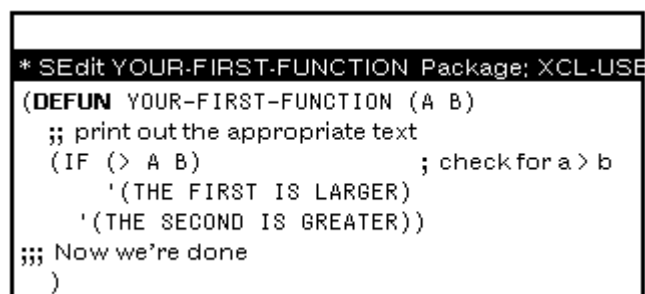
Notice the asterisk in the left edge of the title bar of the window. This designates that the function has been changed. Now exit the edit session by typing Control-X, and the function will be redefined.

Commenting Functions

Text can be marked as a comment by typing a semi-colon before the text of the comment.

```
; This is the form of a comment
```

Inside an editor window, the comment will be printed in a different font and may be moved to the far right of the code. SEdit is familiar with the Common Lisp convention of single comments being on the far right, double comments being justified with the function level, and triple comments being on the far left, as is shown in Figure 7-8.



```
* SEdit YOUR-FIRST-FUNCTION Package: XCL-USER
(DEFUN YOUR-FIRST-FUNCTION (A B)
  ;; print out the appropriate text
  (IF (> A B) ; check for a > b
    '(THE FIRST IS LARGER)
    '(THE SECOND IS GREATER))
  ;; Now we're done
)
```

Figure 7-8. Placement of Comments

There are other editor commands which can be very useful. To learn about them, read Appendix B of the *Release Notes*.

File Functions and Variables: How to See and Save Them

With Medley, all work is done inside the Lisp environment. There is no operating system or command level other than the Executive Window. All functions and data structures are defined and edited using normal Lisp commands. This section describes tools in the Medley environment that will keep track of any changes that you make in the environment that you have not yet saved on files, such as defining new functions, changing the values of variables, or adding new variables. And it then has you save the changes in a file you specify. All of these functions are in the INTERLISP (IL:) package.

File Variables

Certain system-defined global variables are used by the file package to keep track of the environment as it stands. You can get system information by checking the values of these variables. Two important variables follow.

- `FILELST` evaluates to a list, all files that you have loaded into the Medley environment.
- `filenameCOMS` (Each file loaded into the Lisp environment has associated with it a global variable, whose name is formed by appending `COMS` to the end of the filename.) This variable evaluates to a list of all the functions, variables, bitmaps, windows, and soon, that are stored on that particular file.

For example, if you type:

```
MYFILECOMS
```

the system will respond with something like:

```
((FNS YOUR-FIRST-FUNCTION )
  VARS))
```

Saving Interlisp-D on Files

The functions `(FILES?)` and `(MAKEFILE 'filename)` are useful when it is time to save function, variables, windows, bitmaps, records and whatever else to files.

`(FILES?)` displays a list of variables that have values and are not already a part of any file, and then the functions that are not already part of any file.

Type:

```
(FILES?)
```

the system will respond with something like:

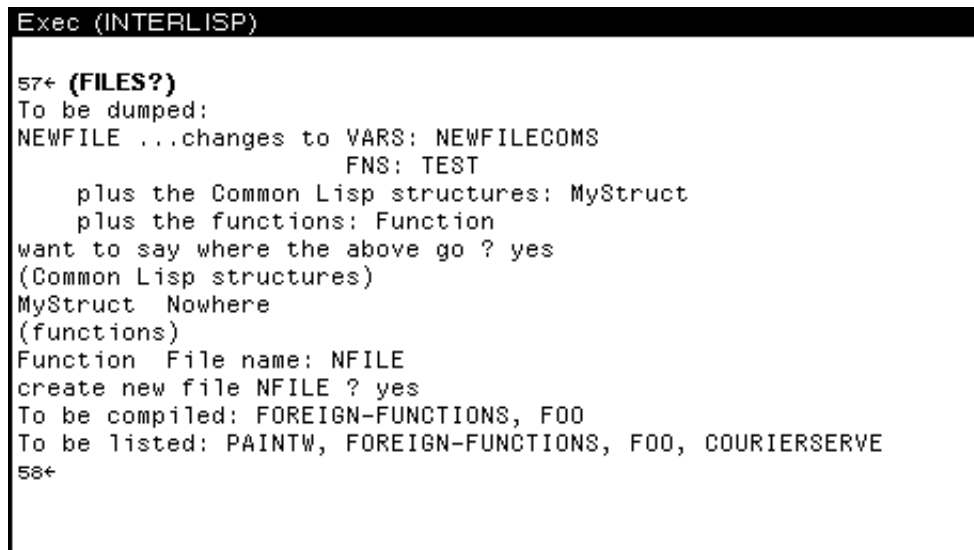
```
the variables: MY.VARIABLE CURRENT.TURTLE...to be
dumped
```

the functions: RIGHT LEFT FORWARD BACKWARD
CLEAR-SCREEN...to be dumped
want to say where the above go?

If you type Y, the system will prompt with each item. There are three options:

1. To save the item, type the filename (unquoted) of the file where the item should be placed. (This can be a brand new file or an existing file.)
2. To skip the item, without removing it from consideration the next time (FILES?) is called, type crø This will allow you to postpone the decision about where to save the item.
3. If the item should not be saved at all, type J. Nowhere will appear after the item.

Part of an example interaction is shown in the following figure:

A screenshot of a terminal window titled "Exec (INTERLISP)". It shows the output of the (FILES?) function. The text is as follows:

```
57+ (FILES?)  
To be dumped:  
NEWFILE ...changes to VARS: NEWFILECOMS  
FNS: TEST  
plus the Common Lisp structures: MyStruct  
plus the functions: Function  
want to say where the above go ? yes  
(Common Lisp structures)  
MyStruct Nowhere  
(functions)  
Function File name: NFILE  
create new file NFILE ? yes  
To be compiled: FOREIGN-FUNCTIONS, FOO  
To be listed: PAINTW, FOREIGN-FUNCTIONS, FOO, COURIERSSERVE  
58+
```

Figure 7-9. Part of an interaction using the function FILES?

(FILES?) assembles the items by adding them to the appropriate file's COMS variable (see the File Variables section above). (FILES?) does NOT write the file to secondary storage (disks or floppies). It only updates the global variables discussed in the File Variables section above.

(MAKEFILE 'filename)

actually writes the file to secondary storage.

Type:

(MAKEFILE 'MY.FILE.NAME)

and the system will create the file. The function returns the full name of the file created. (i.e. {DSK}MY.FILE.NAME.; 1).

Files written to (DSK) are permanent files. They can be removed only by the user deleting them or by reformatting the disk.

Other file manipulation functions can be found in Chapter 4.

8. YOUR INIT FILE

Lisp has a number of global variables that control the environment. Global variables make it easy to customize the environment to fit your needs. One way to do this is to develop an INIT file. This is a file that is loaded when you start an image. You can use it to set variables, load files, define functions, and any other things that you want to do to make the Medley environment suit you.

Using the USERGREETFILES Variable

As described in File Variables section of Chapter 11, each program file has a global variable associated with it, whose name is formed by appending COMS to the end of the root filename. For any of the standard INIT file names, the variable INITCOMS is used. Your INIT file could be called INIT, INIT.LISP, INIT.USER, or whatever the convention is at your site. There is no default name preferred by the system, it just looks for the files listed in the variable USERGREETFILES (see below). Check to see what the preference is at your site. Put this file in your directory. Your directory name should be the same as your login name. The INIT file is loaded by the function GREET. GREET is normally run when Medley is started. If this is not the case at your site, or you want to use the machine and Medley has already been started, you can run the function GREET yourself. If your user name was, for example, TURING, then you would type:

```
(GREET 'TURING)
```

This does a number of things, including undoing any previous greeting operation, loading the site init file, and loading your init file. Where GREET looks for your INIT file depends on the value of the variable USERGREETFILES. The value of this variable is set when the system's SYSOUT file is made, so check its value at your site! For example, its value could be:

```
80+ USERGREETFILES
({{DSK}INIT %. COM)
({DSK}INIT- USER %. COM)
({DSK}INIT- USER)
({DSK}INIT))
```

Figure 8-1. Possible Value of USERGREETFILES

In each place you see >USER>, the argument passed to GREET is substituted into the path. This is your login name if you are just starting Medley. For example, the first value in the list would have the system check to see whether there was a {DSK}<LISPFILES>TURING>INIT.LISP file. No error is generated if you do not have an INIT file, and none of the files in USERGREETFILES are found.

Making an Init File

As described in File Variables section of Chapter 11, each program file has a global variable associated with it, whose name is formed by appending COMS to the end of the root filename. For any of the standard INIT file names, the variable INITCOMS is used. To set up an init file, begin by editing this variable. Type:

```
(DV INITCOMS)
```

An SEdit window will appear. This window is the same as the one called with the function `DF`, and described in the *Using the List Structure Editor* section in Chapter 7. This chapter assumes that you know how to use the SEdit structure editor .

The `COMS` variable is a list of lists. The first atom in each internal list specifies for the file package what types of items are in the list, and what it is to do with them. This section will deal with three types of lists: `VARS`, `FILES`, and `P`. Please read about others in Chapter 17 of the *IRM*.

Notice that inside the `vars` list, there is yet another list. The first item in the list is the name of the variable. It is bound to the value of the second item. There are many other variables that you can set by adding them to the `VARS` list. Some of these variables are described in Chapter 24, and many others can be found in the *IRM*.

If you want to automatically load files, that can be done in your init file also. For example, if you always want to load the Library file `SPY.LCOM`, you can load it by editing the `INITCOMS` variable to list the appropriate file in the list starting with `FILES`:

```
.  
.   
.   
(FILES SPY)  
.   
.   
. 
```

Figure 8-2. `INITCOMS` Changed to Load `SPY.LCOM` File

Other files can also be added by simply adding their names to this `FILES` list.

Another list that can appear in a `COMS` list begins with `P`. This list contains Lisp expressions that are evaluated when the file is loaded. Do not put `DEFINEQ` expressions in this list. Define the function in the environment, and then save it on the file in the usual way (see Chapter 7).

One type of expression you might want to see here, however, is a `FONTCREATE` function (see Chapter 16). For example, if you want to use a Helvetica 12 BOLD font, and there is not a fontdescriptor for it normally in your environment, the appropriate call to `FONTCREATE` should be in the "P" list. The `INITCOMS` would look like this:

```
.  
.   
.   
(FILES SPY)  
(P (FONTCREATE 'HELVETICA 12 'BOLD))  
.   
.   
. 
```

Figure 8-3. `INITCOMS` Edited to Include a call to `FONTCREATE`

To quit, exit from SEdit in the usual way. When you run the function `MAKEFILES` (see Chapter 7), be sure that you are connected to the directory (see Chapter 4) where the `INIT` file should appear. Now when `GREET` is run, your Init file will be loaded.

9. MEDLEY FORGIVENESS: DWIM

DWIM (Do What I Mean) is an Interlisp utility that makes life easier.

DWIM tries to match unrecognized variable and function names to known ones. This allows Lisp to interpret minor typing errors or misspellings in a function, without causing a break. Line 152 of Figure 9-1 illustrates how the misspelled BANNANNA was replaced by BANANA before the expression was evaluated.

```
Exec (INTERLISP)

151← (DEFINEQ (PEEL (BANANA) (CDR BANNANNA)))
      (PEEL)
152← (PEEL '(A B D))
      BANNANNA {in PEEL} -> BANANA ? Yes
      (B D)
153←
```

Figure 9-1. Examples of DWIM Features

Sometimes DWIM may alter an expression you didn't want it to. This may occur if, for example, a hyphenated function name (e.g., (MY-FUNCTION)) is misused. If the system does not recognize the function name, it may think you are trying to subtract "FUNCTION" from "MY". DWIM also takes the liberty of updating the function, so it will have to be fixed. However, this is as much a blessing as a curse, since it points out the misused expression!

10. BREAKPACKAGE

The Break Package is a part of Interlisp that makes debugging your programs much easier.

Break Windows

A break is a function either called by the programmer or by the system when an error has occurred. A separate window opens for each break. This window works much like the Executive Window, except for extra menus unique to a break window. Inside a break window, you can examine variables, look at the call stack at the time of the break, or call the editor. Each successive break opens a new window, where you can execute functions without disturbing the original system stack. These windows disappear when you resolve the break and return to a higher level.

Break Package Example

This example illustrates the basic break package functions. A more complete explanation of the breaking functions, and the break package will follow.

The correct definition of FACTORIAL is:

```
(defun factorial (x)
  (if (zerop x)
      1
      (* x (factorial (1- x)))))
```

To demonstrate the break package, we have edited in an error: DUMMY in the IF statement is an unbound atom, it lacks a value.

```
((defun factorial (x)
  (if (zerop x)
      dummy
      (* x (factorial (1- x)))))
```

The evaluated function

```
(FACTORIAL 4)
```

should return 24, but the above function has an error. DUMMY is an unbound atom, an atom without an assigned value, so Lisp will "break". A break window appears (Figure 10-1), that has all the functionality of the typing lisp expressions into the Executive Window (The top level), in addition to the break menu functions. Each consecutive break will move to another level "down".

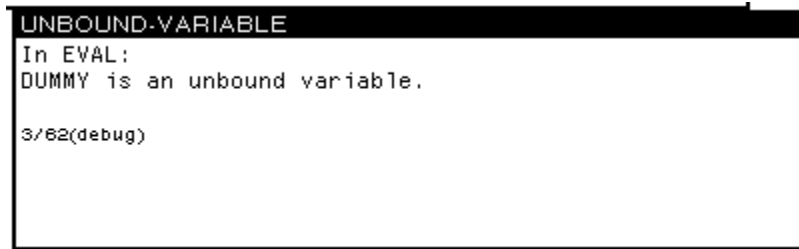


Figure 10-1. Break Window

Move the mouse cursor into the break window and hold down the middle mouse button. The Break Menu will appear. Choose BT. Another menu, called the stack menu, will appear beside the break window. Choosing stack items from this menu will display another window. This window displays the function's local variable bindings, or values (see Figure 10-2). This new window, titled FACTORIAL Frame, is an inspector window (see inspector Chapter 17).

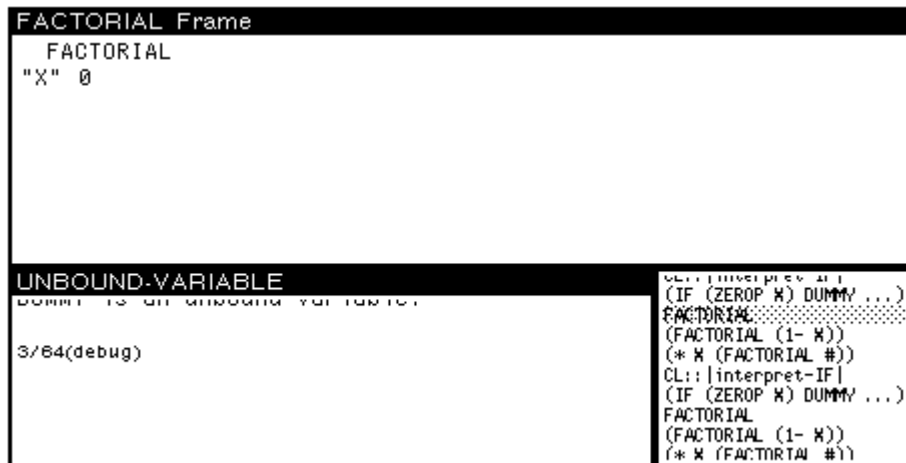


Figure 10-2. Back Trace of the System Stack

From the break window, you can call the editor for the function FACTORIAL by middle-buttonning on the word FACTORIAL and selecting DisplayEdit from the menu that pops up.

Replace the unbound atom DUMMY with 1. Exit the editor .

The function is fixed, and you can restart it from the last call on the stack. (It does not have to be started again from the Top Level.) To begin again from the last call on the stack, choose the last (top) FACTORIAL call in the BT menu. Select REVERT from the middle button break window, or type it into the window. The break window will close, and a new one will appear with the message: **Breakpoint at FACTORIAL**

To start execution with this last call to FACTORIAL, choose OK from the middle button break menu. The break window will disappear, and the correct answer, 24, will be returned to the top level.

Ways to Stop Execution from the Keyboard (Breaking Lisp)

There are ways you can stop execution from the keyboard. They differ in terms of how much of the current operating state is saved:

Control-G	Provides you with a menu of processes to interrupt. Your process will usually be "EXEC". Choose it to break your process. A break window will then appear.
Control-B	Causes your function to break, saves the stack, then displays a break window with all the usual break functions. For information on other interrupt characters, see Chapter 30 in the <i>IRM</i> .

Break Menu

Move the mouse cursor into the break window. Hold the middle button down, and a new menu will pop up, like the one in Figure 10-3.

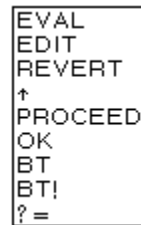


Figure 10-3. Middle Button Menu in Break window

Five of the selections are particularly important when just starting to use Medley:

BT	Back Trace displays the stack in a menu beside the break window. Back Trace is a very powerful debugging tool. Each function call is placed on the stack and removed when the execution of that function is complete. Choosing an item on the stack will open another window displaying that item's local variables and their bindings. This is an inspector window that offers all the power of the inspector. (For details, see the section on the Inspector, Chapter 17.)
? =	Before you use this menu option, display the stack by choosing BT from this menu, and choose a function from it. Now, choose ?=. It will display the current values of the arguments to the function that has been chosen from the stack.
↑	Move back to the previous break window, or if there is no other break window, back to the top level, the Executive Window.
REVERT	Move the point of execution back to a specified function call before the error. The function to revert back to is, by default, the last function call before the break. If, however, a different function call is chosen on the BT menu, revert will go back to the start of this function and open a new break window. The items on the stack above the new starting place will no longer exist. This is used in the tutorial example (see the Break Package Example section above).

OK Continue execution from the point of the break. This is useful if you have a simple error, i.e., an unbound variable or a nonnumeric argument to an arithmetic function. Reset the variable in the break window, then select OK. (see the Break Package Example section above).

In addition to being available on the middle button menu of the break window, all of these functions can be typed directly into the window. Only BT behaves differently when typed. It types the stack into the trace window instead of opening a new window.)

Returning to Top Level

Typing Control-D will immediately take you to the top level from any break window. The functions called before the break will stop, but any side effects of the function that occurred before the break remain. For example, if a function set a global variable before it broke, the variable will still be set after typing Control-D.

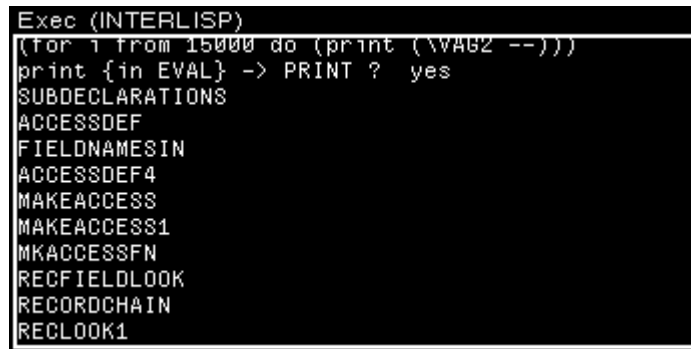
11. WHAT TO DO IF ...

The purpose of this chapter is to explain what to do in some of the problems commonly experienced by Medley users.

Executive Window turns black

An example is shown in Figure 11-1.

Press any key to unfreeze the window and continue. This pause happens when the command you just typed causes enough information to be printed to fill the window. It gives you a chance to read that one window of text before moving on.



```
Exec (INTERLISP)
(for 1 from 15000 do (print (\VAB2 --)))
print {in EVAL} -> PRINT ? yes
SUBDECLARATIONS
ACCESSDEF
FIELDNAMESIN
ACCESSDEF4
MAKEACCESS
MAKEACCESS1
MKACCESSFN
RECFIELDLOOK
RECORDCHAIN
RECLOOK1
```

Figure 11-1. Blackened Executive Window

You closed the Executive Window

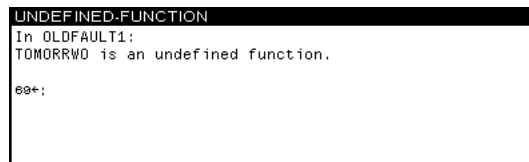
Open another from the Background Menu.

Mouse disappears

Type (**CURSOR T**) in the Executive Window. The cursor will reappear.

Second window appears

This probably happens because you made a typing mistake, as in Figure 11-2.



```
UNDEFINED-FUNCTION
In OLDFAULT1:
TOMORRWO is an undefined function.
gg*:
```

Figure 11-2. Second Window Appears (Break Window) after Typing Error Made

Type a Control-D by simultaneously pressing the Control key and the "D". This aborts the error condition, returning control to the Executive Window.

You keep getting beeped at

Usually the beeping means that Medley want input from you. Look for the flashing caret. It will usually be preceded by some kind of prompt, indicating what you should type.

You cannot delete the first letter

of the filename you are typing to (FILES?). Type Control-E (error) You will get a linefeed and ←←← printed to the window. Now type the correct filename.

Your function is just sitting there

It is not returning a value, and you think that your program may be in an infinite loop or is having some other major problem. You can see what process is currently running by typing Control-T, or you could interrupt the process by typing Control-E.

A Break Window appears

If the Break Window look something like that shown in Figure 11-3, you are trying to save a file, but there is not enough space on the hard disk.

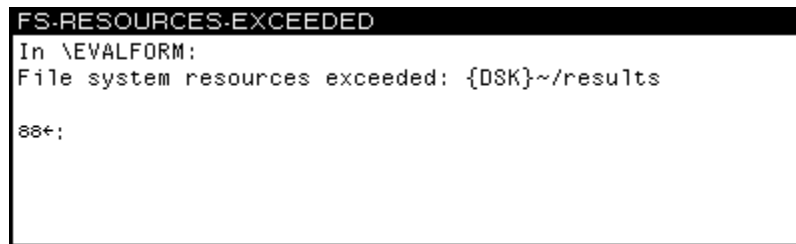


Figure 11-3. Break Window Caused by Insufficient Space in Save File

Exit from the Break Window by typing an up arrow ↑ followed by a Return. Delete old versions of files, and any other files you do not need. Then try again to save the file

You have run out of space

Generally, a Break Window has appeared. The GAINSPACE function allows you to delete non-essential data structures. To use it, type:

(GAINSPACE)

into the Executive Window. Answer N to all questions except the following:

- Delete edit history
- Delete history list
- Delete values of old variables
- Delete your MASTERSCOPE database
- Delete information for undoing your greeting.

Save your work and reload Lisp as soon as possible.

A redefined message appears

The message (*Some.Crucial.Function.Or.Variable* redefined) appears in the Executive Window (see Figure 11-4). The function, variable, or other property has been "smashed" (i.e., its original definition has been changed). If this is not what you wanted, type **UNDO** immediately!

```

Exec (INTERLISP)

124+ (DEFINEQ (CAR (A) (SomeOtherFn A)))
New fns definition for CAR.
(CAR)
125+ UNDO
DEFINEQ undone.
126+

```

Figure 11-4. CAR redefined!

UNBOUND ATOM

If this occurs, you probably just typed something wrong, or you passed an argument that should have been quoted to a function.

UNDEFINED CAR OF FORM

First, look at what caused the error. If the CAR of the form is a list, then you typed something wrong. If it is an atom, then perhaps that atom does not have a function associated with it. If it is a CLISP word like `if` or `for`, then DWIM may have been turned off (see Chapter 9). Type `(DWIM 'C)` to reenale DWIM.

You have traced APPLY

and your screen is spewing out information about everything going on in the environment. Type Control E, and type `(UNBREAK 'APPLY)` before rereturning to the Executive.

[This page intentionally left blank]

12. WINDOWS AND REGIONS

Windows

Windows have two basic parts: an area on the screen containing a collection of pixels, and a property list. The window properties determine how the window looks, the menus that can be accessed from it, what should happen when the mouse is inside the window and a mouse button is pressed, and soon.

CREATEW

Some of the window's properties can be specified when a window is created with the function `CREATEW`. In particular, it is easy to specify the size and position of the window; its title; and the width of its borders.

(CREATEW region title borderwidth)

Region is a record (named `REGION`, with the fields `left`, `bottom`, `width`, and `height`) or a list. A region describes a rectangular area on the screen, the window's dimensions and position. The fields `left` and `bottom` refer to the position of the bottom left corner of the region on the screen. `Width` and `height` refer to the width and height of the region. The usable space inside the window will be smaller than the width and height, because some of the window's region is consumed by the title bar, and some is taken by the borders.

Title is a string that will be placed in the title bar of the window.

Borderwidth is the width of the border around the exterior of the window, in number of pixels.

For example, typing:

```
(SETQ MY.WINDOW (CREATEW
  (CREATEREGION 100 150 300 200)
  "THIS IS MY OWN WINDOW"))
```

or

```
(SETQ MY.WINDOW (CREATEW
  (CREATEW ' (100 150 300 200)
  "THIS IS MY OWN WINDOW"))
```

produces a window with a default borderwidth. Note that you did not need to specify all the window's properties (see Figure 12-1).

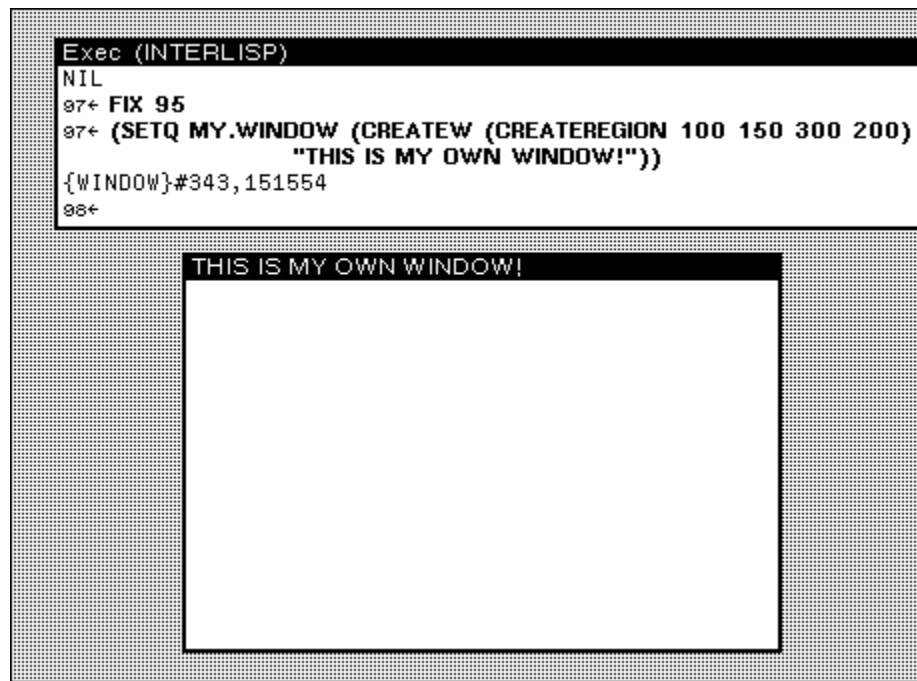


Figure 12-1. Creating a Window

In fact, if (CREATEW) is called without specifying a region, you will be prompted to sweep out a region for the window (see Chapter 10)

WINDOWPROP

The function to access or add to any property of a window's property list is WINDOWPROP.

```
(WINDOWPROP window property <value>)
```

When you use WINDOWPROP with only two arguments—window and property—it returns the value of the window's property. When you use WINDOWPROP with all three arguments—window, property and value—it sets the value the window's property to the value you inserted for the third argument.

For example, consider the window, MY WINDOW, created using (CREATEW). TITLE and REGION are both properties. Type

```
(WINDOWPROP MY.WINDOW 'TITLE)
```

and the value of MY.WINDOW's TITLE property is returned, "THIS IS MY OWN WINDOW". To change the title, use the WINDOWPROP function, and give it the window, the property title, and the new title of the window.

```
(WINDOWPROP MY.WINDOW 'TITLE "MY FIRST WINDOW")
```

automatically changes the title and automatically updates the window. Now the window looks like Figure 12-2.

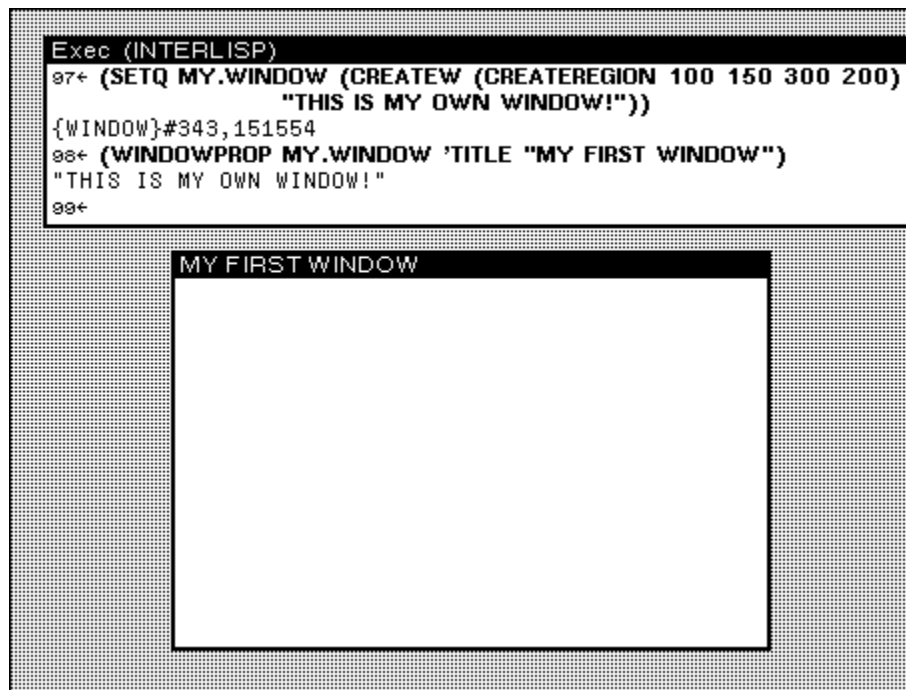


Figure 12-2. TITLE is a Window Property

Altering the region of the window, `MY.WINDOW`, is also be done with `WINDOWPROP`, in the same way you changed the title. (Changing either of the first two numbers of a region changes the position of the window on the screen. Changing either of the last two numbers changes the dimensions of the window itself.)

Getting Windows to Do Things

Four basic window properties will be discussed here: `CURSOR1NFN`, `CURSOROUTFN`, `CURSORMOVEDFN`, and `BUTTONEVENTFN`.

A function can be stored as the value of the `CURSOR1NFN` property of a window. It is called when the mouse cursor is moved into that window.

Look at the following example:

1. First, create a window called `MY.WINDOW`. Type:

```

(SETQ MY.WINDOW
  (CREATEW
    (CREATEREGION 200 200 200 200)
    "THIS WINDOW WILL SCREAM!"))
  
```

This creates a window.

2. Now define the function `SCREAMER`. It will be stored on the property `CURSOR1NFN`. (Notice that this function has one argument, `WINDOWNAME`. All functions called from the property `CURSOR1NFN` are passed the window it was called from. So the value of `MY.WINDOW` is bound to `WINDOWNAME`. When it is called, `SCREAMER` simply rings bells.

```
(DEFINEQ (SCREAMER (WINDOWNAME)
  (RINGBELLS)
  (PROMPTPRINT "YAY - IT WORKS!")
  (RINGBELLS)))
```

3. Now, alter that window's `CURSORINFN` property, so that the system calls the function `SCREAMER` at the appropriate time. Type:

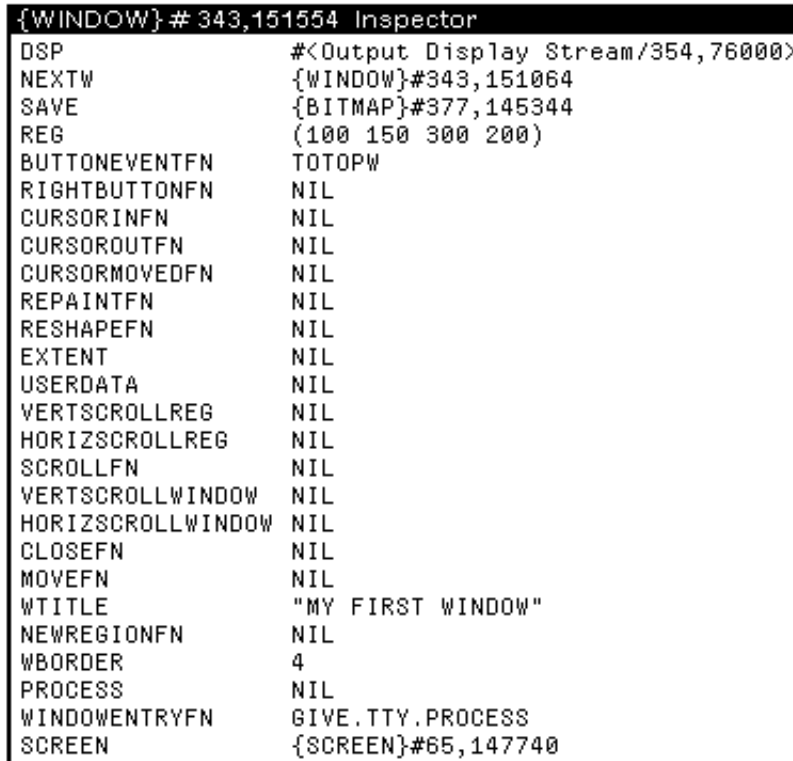
```
(WINDOWPROP MY.WINDOW 'CURSORINFN
  (FUNCTION SCREAMER))
```

4. After this, when you move the mouse cursor into `MY.WINDOW`, the `CURSORINFN` property's function is called, and it rings bells twice.

`CURSORINFN` is one of the many window properties that come with each window - just as `REGION` and `TITLE` did. Other properties include:

<code>CURSOROUTFN</code>	The function that is the value of this property is executed when the cursor is moved out of a window.
<code>CURSORMOVEDFN</code>	The function that is the value of this property is executed when the cursor is moved while it is inside the window.
<code>BUTTONEVENTFN</code>	The function that is the value of this property is executed when either the left or middle mouse buttons are pressed (or released).

Figure 12-3 shows `MY.WINDOW`'s properties. Notice that the `CURSORINFN` has the function `SCREAMER` stored in it. The properties were shown in this window using the function `INSPECT`. `INSPECT` is covered in Chapter 17.



```
{WINDOW} # 343,151554 Inspector
DSP                #<Output Display Stream/354,76000>
NEXTW              {WINDOW}#343,151064
SAVE               {BITMAP}#377,145344
REG                (100 150 300 200)
BUTTONEVENTFN      TOTOPW
RIGHTBUTTONFN      NIL
CURSORINFN         NIL
CURSOROUTFN        NIL
CURSORMOVEDFN      NIL
REPAINTFN          NIL
RESHAPEFN          NIL
EXTENT             NIL
USERDATA           NIL
VERTSCROLLREG      NIL
HORIZSCROLLREG     NIL
SCROLLFN           NIL
VERTSCROLLWINDOW   NIL
HORIZSCROLLWINDOW  NIL
CLOSEFN            NIL
MOVEFN             NIL
WTITLE             "MY FIRST WINDOW"
NEWREGIONFN        NIL
WBORDER            4
PROCESS            NIL
WINDOWENTRYFN      GIVE.TTY.PROCESS
SCREEN             {SCREEN}#65,147740
```

Figure 12-3. Inspecting `MY.WINDOW` for Mouse-Related Window Properties

You can define functions for the values of the properties `CURSORMOVEDFN` and `CURSORMOVEDFN` in much the same way as you did for `CURSORMOVEDFN`. The function that is the value of the property `BUTTONEVENTFN`, however, can be specialized to respond in different ways, depending on which mouse button is pressed. This is explained in the next section.

BUTTONEVENTFN

`BUTTONEVENTFN` is another property of a window. The function that is stored as the value of this property is called when the mouse is inside the window, and a mouse button is pressed. As an example of how to use it, type:

```
(WINDOWPROP MY.WINDOW 'BUTTONEVENTFN
  (FUNCTION SCREAMER))
```

When the mouse cursor is moved into the window, bells will ring because of the `CURSORMOVEDFN`, but it will also ring bells when either the left or middle mouse button is pressed. Notice that the right mouse button functions as it usually does, with the window manipulation menu. If only the left button should evoke the function `SCREAMER`, then the function can be written to do just this, using the function `MOUSESTATE`, and a form that only `MOUSESTATE` understands, `ONLY`. For example:

```
(DEFINEQ
  (SCREAMER2 (WINDOWNAME)
    (if (MOUSESTATE (ONLY LEFT))
      then (RINGBELLS))))
```

In addition to `(ONLY LEFT)`, `MOUSESTATE` can also be passed `(ONLY MIDDLE)`, `(ONLY RIGHT)` or combinations of these (e.g. `(OR (ONLY LEFT) (ONLY MIDDLE))`). You do not need to use `ONLY` with `MOUSESTATE` for every application. `ONLY` means that that button is pressed and no other.

If you do write a function using `(ONLY RIGHT)`, be sure that your function also checks position of the mouse cursor. Even if you want your function to be executed when the mouse cursor is inside the window and the right button is pressed, there is a convention that the function `DOWINDOWCOM` should be executed when the mouse cursor is in the title bar or the border of the window and the right mouse button is pressed. Please program your windows using this tradition! For more information, please see Chapter 28 in the *IRM*.

Looking at a Window's Properties

`INSPECT` is a function that displays a list of the properties of a window, and their values. Figure 12.3 shows the `INSPECT` function run with `MY.WINDOW`. Note the properties introduced in `CREATEW`: `WBORDER` is the window's border, `REG` is the region, and `WTITLE` is the window's title.

Regions

A region is a record, with the fields `LEFT`, `BOTTOM`, `WIDTH`, and `HEIGHT`. `LEFT` and `BOTTOM` refer to where the bottom left hand corner of the region is positioned on the screen. `WIDTH` and `HEIGHT` refer to the width and height of the region.

CREATEREGION creates an instance of a record of type REGION. Type:

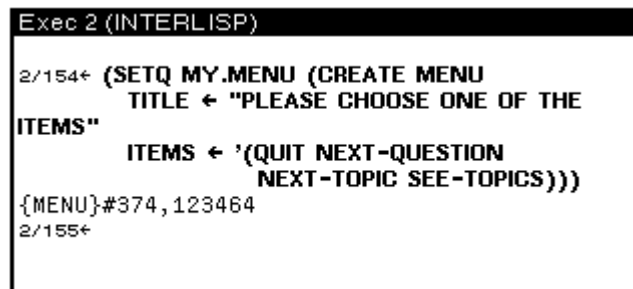
```
(SETQ MY.REGION (CREATEREGION 15 100 200 450))
```

to create a record of type REGION that denotes a rectangle 200 pixels high, and 450 pixels wide, whose bottom left corner is at position (15, 100). This record instance can be passed to any function that requires a region as an argument, such as CREATEW, above.

13. WHAT ARE MENUS?

While Medley provides a number of menus of its own (see Chapter 3), this section addresses the menus you wish to create. You will learn how to create a menu, display a menu, and define functions that make your menu useful. Menus are instances of records (see Chapter 24). There are 27 fields that determine the composition of every menu. Because Medley provides default values for most of these descriptive fields, you need to familiarize yourself with only a few that we describe in this section.

Two of these fields, the `TITLE` of your menu, and the `ITEMS` you wish it to contain, can be typed into the executive window as shown below:



The screenshot shows a terminal window titled "Exec 2 (INTERLISP)". Inside, the following code is entered and executed:

```
2/154← (SETQ MY.MENU (CREATE MENU  
TITLE ← "PLEASE CHOOSE ONE OF THE  
ITEMS"  
ITEMS ← '(QUIT NEXT-QUESTION  
NEXT-TOPIC SEE-TOPICS)))  
{MENU}#374,123464  
2/155←
```

Figure 13-1. Creating a menu

Note that creating a menu does not display it. `MY.MENU` is set to an instance of a menu record that specifies how the menu will look, but the menu is not displayed.

Displaying Menus

Typing either the `MENU` or `ADDMENU` functions will display your menu on the screen. `MENU` implements pop-up menus, like the Background Menu or the Window Menu. `ADDMENU` puts menus into a semi-permanent window on the screen, and lets you select items from it.

`(MENU MENU POSITION)` pops up a menu at a particular position on the screen.

Type:

```
(MENU MY.MENU NIL)
```

to position the menu at the end of the mouse cursor. Note that the `POSITION` argument is `NIL`. In order to go on, you must either choose an item, or move outside the menu window and press a mouse button. When you do either, the menu will disappear. If you choose an item, then want to choose another, the menu must be redisplayed.

`(ADDMENU menu window position)` positions a permanent menu on the screen, or in an existing window.

Type:

```
(ADDMENU MY.MENU)
```

to display the menu as shown in Figure 13-2. This menu will remain active, (will stay on the screen) without stopping all the other processes. Because ADDMENU can display a menu without stopping all other processes, it is very popular in users programs.

If window is specified, the menu is displayed in that window. If window is not specified, a window the correct size for the menu is created, and the menu is displayed in that window.

If position is not specified, the menu appears at the current position of the mouse cursor.

```
PLEASE CHOOSE ONE OF THE ITEMS
QUIT
NEXT-QUESTION
NEXT-TOPIC
SEE-TOPICS
```

Figure 13-2. Simple MenuDisplayed with ADDMENU

Getting Menus to Do Stuff

One way to make a menu do things is to specify more about the menu items. Instead of items simply being the strings or atoms that will appear in the menu, items can be lists, each list with three elements (see Figure 13-3). The first element of each list is what will appear in the menu; the second expression is what is evaluated, and the results of the evaluation returned, when the item is selected; and the third expression is the expression that should be printed in the Prompt window when a mouse button is held down while the mouse is pointing to that menu item. This third item should be thought of as help text for the user. If the third element of the list is NIL, the system responds with **Will select this item when you release the button.**

```
Exec (INTERLISP)

100← (SETQ MY.MENU2 (CREATE MENU
  TITLE ← "PLEASE CHOOSE ONE OF THE ITEMS"
  ITEMS ← '((QUIT (PRINT "STOPPED") "CHOOSE THIS TO STOP")
    (NEXT-QUESTION
      (PRINT "HERE IS THE NEXT QUESTION...")
      "CHOOSE THIS TO BE ASKED THE NEZT QUESTION")
    (NEXT-TOPIC
      (PRINT "HERE IS THE NEXT TOPIC...")
      "CHOOSE THISE TO MOVE ON TO THE NEXT TOPIC")
    (SEE-TOPICS
      (PRINT "THE FOLLOWING HAVE NOT BEEN
LEARNED")
      "CHOOSE THIS TO SEE TOPICS NOT YET
COVERED"))))
{MENU}#356,10334
101← (ADDMENU MY.MENU2)
{WINDOW}#343,132150
102←

PLEASE CHOOSE ONE OF THE ITEMS
QUIT
NEXT-QUESTION
NEXT-TOPIC
SEE-TOPICS
```

Figure 13-3. Creating a Menu to do Things, then displaying it with the function
ADDMENU

Now when an item is selected from MY.MENU2, something will happen. When a mouse button is held down, the expression typed as the third element in the item's specification will be printed in the Prompt window. (See Figure 13-4.)

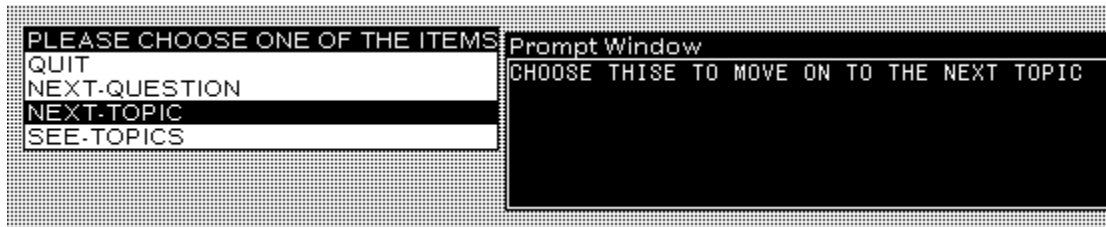


Figure 13-4. Mouse Button Held Down While Mouse Cursor Selects NEXT-QUESTION

When the mouse button is released (i.e., the item is selected) the expression that was typed as the second element of the item's specification will be run. (See Figure 13-5.)

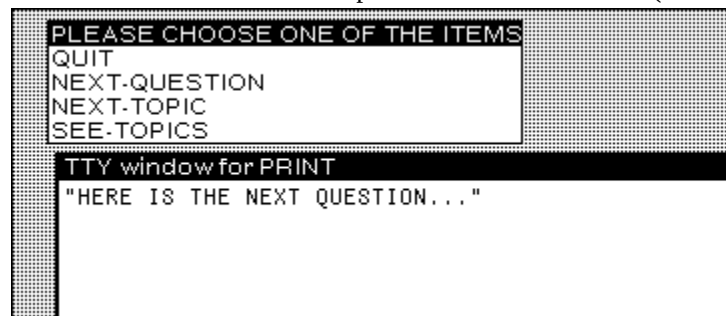


Figure 13-5. NEXT-QUESTION Selected

WHENHELDFN and WHENSELECTEDFN Fields of a Menu

Another way to get a menu to do things is to define functions, and make them the values of the menu's WHENHELDFN and WHENSELECTEDFN fields. As the value of the WHENHELDFN field of a menu, the function you defined will be executed when you press and hold a mouse button inside the menu. As the value of the WHENSELECTEDFN field of a menu, the function you defined will be executed when you choose a menu item. This example has the same functionality as the previous example, where each menu item was entered as a list of three items.

As an example, type in these two functions so that they can be executed when the menu is created and displayed:


```

(DEFINEQ (MY.MENU3.WHENHELD (ITEM.SELECTED MENU.FROM
BUTTON.PRESSED)
(SELECTQ ITEM.SELECTED
  (QUIT (PROMPTPRINT "CHOOSE THIS TO STOP"))
  (NEXT-QUESTION (PROMPTPRINT "CHOOSE THIS TO BE ASKED THE
                        NEXT QUESTION"))
  (NEXT-TOPIC (PROMPTPRINT "CHOOSE THIS TO MOVE ON TO THE
                        NEXT SUBJECT"))
  (SEE-TOPICS (PROMPTPRINT "CHOOSE THIS TO SEE THE TOPICS
                        NOT YET LEARNED"))
  (ERROR (PROMPTPRINT "NO MATCH FOUND"))))

(DEFINEQ (MY.MENU3.WHENSELECTED (ITEM.SELECTED MENU.FROM
BUTTON.PRESSED)
(SELECTQ ITEM.SELECTED
  (QUIT (PRINT "STOPPED"))
  (NEXT-QUESTION (PRINT "HERE IS THE NEXT QUESTION"))
  (NEXT-TOPIC (PRINT "HERE IS THE NEXT SUBJECT"))
  (SEE-TOPICS (PRINT "THE FOLLOWING HAVE NOT BEEN
                        LEARNED . . ."))
  (ERROR (PROMPTPRINT "NO MATCH FOUND"))))

```

Now, to create the menu, type:

```

(SETQ MY.MENU3 (CREATE MENU
  TITLE ← "PLEASE CHOOSE ONE OF THE ITEMS"
  ITEMS ← '(QUIT NEXT-QUESTION NEXT-TOPIC SEE-TOPICS)
  WHENHELDFN ← (FUNCTION MY.MENU3.WHENHELD)
  WHENSELECTEDFN ← (FUNCTION MY.MENU3.WHENSELECTED)))

```

To see your menu work, type

```
(ADDMENU MY.MENU3)
```

Now, due to executing the WHENHELDFN function, holding down any mouse button while pointing to a menu item will display an explanation of the item in the prompt window. The screen will once again look like Figure 13-4 when the mouse button is held when the mouse cursor is pointing to the item NEXT-TOPIC.

Now due to executing the WHENSELECTEDFN function, releasing the mouse button to select an item will cause the proper actions for that item to be taken. The screen will once again look like Figure 13-5 when the item NEXT-TOPIC is selected. The crucial thing to note is that the functions you defined for WHENHELDFN and WHENSELECTEDFN are automatically given the following arguments:

1. The item that was selected, ITEM.SELECTED
2. The menu it was selected from, MENU.FROM
3. The mouse button that was pressed BUTTON.PRESSED

These functions, MY.MENU3.WHENHELD and MY.MENU3.WHENSELECTED, were quoted using FUNCTION instead of QUOTE both for program readability and so that the compiler can produce faster code when the program is compiled. It is good style to quote functions in Lisp by using the function FUNCTION instead of QUOTE.

Looking at a Menu's Fields

INSPECT is a function that displays a list of the fields of a menu, and their values. Figure 13-6 shows the various fields of MY.MENU3 when the function (INSPECT MY.MENU3) was called. Notice the values that were assigned by the examples, and all the defaults.

161← (INSPECT MY.MENU3)	
{WINDOW}#357,73064	
162←	{MENU}# 336,174464 Inspector
	ITEMWIDTH 236
	ITEMHEIGHT 12
	IMAGEWIDTH 238
	IMAGEHEIGHT 62
	MENUREGIONLEFT -1
	MENUREGIONBOTTOM -1
	IMAGE {WINDOW}#376,26000
	SAVEIMAGE NIL
	ITEMS (QUIT NEXT-QUESTION NEXT-TOPIC SEE-T
	MENUROWS 4
	MENUCOLUMNS 1
	MENUGRID (0 0 236 12)
	CENTERFLG NIL
	CHANGEOFFSETFLG NIL
	MENUFONT {FONTDESCRIPTOR}#74,70204
	TITLE "PLEASE CHOOSE ONE OF THE ITEMS"
	MENUOFFSET (0 . 0)
	WHENSELECTEDFN MY.MENU3.WHENSELECTED
	MENUBORDERSIZE 0
	MENUOUTLINESIZE 1
	WHENHELDFN MY.MENU3.WHENHELD
	MENUPOSITION NIL
	WHENUNHELDFN CLRprompt
	MENUUSERDATA NIL
	MENUTITLEFONT NIL
	SUBITEMFN NIL
	MENUFEEEDBACKFLG NIL
	SHADEDITEMS NIL

Figure 13-6. MY.MENU3 Fields

14. BITMAPS

A bitmap is a rectangular array of dots. The dots are called "pixels" (for picture elements). Each dot, or pixel, is represented by a single bit. When a pixel or bit is turned on (i.e. that bit set to 1), a black dot is inserted into a bitmap. If you have a bitmap of a floppy on your screen (Figure 14-1), then all of the bits in the area that make up the floppy are turned on, and the surrounding bits are turned off.

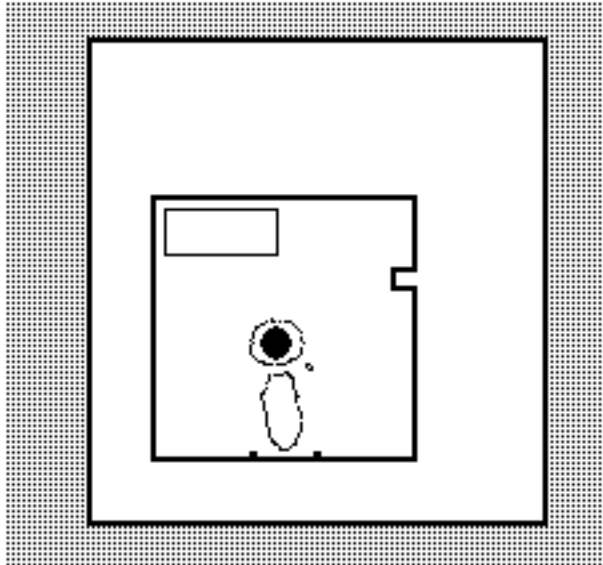


Figure 14-1. Bitmap of a Floppy

`BITMAPCREATE` creates a bitmap, even though it can't be seen.

```
(BITMAPCREATE width height)
```

If the width and height are not supplied, the system will prompt you for them.

`EDITBM` edits the bitmap. The syntax of the function is:

```
(EDITBM bitmapname)
```

Try the following to produce the results in Figure 14-4:

```
(SETQ MY.BITMAP (BITMAPCREATE 60 40))  
EDITBM MY.BITMAP)
```

To draw In the bitmap, move the mouse into the gridded section of the bitmap editor, and press and hold the left mouse button. Move the mouse around to turn on the bits represented by the spaces in the grid. Notice that each space in the grid represents one pixel on the bitmap

To erase Move the mouse into the gridded section of the bitmap editor, and press and hold the center mouse button. Move the mouse around to turn off the bits represented by the spaces in the gridded section of the bitmap editor.

To work on a different section Point with the mouse cursor to the picture of the actual bitmap (the upper left corner of the bitmap editor). Press and hold the left mouse button. A menu with the single item, Move will appear. (See Figure 14-2.) Choose this item.

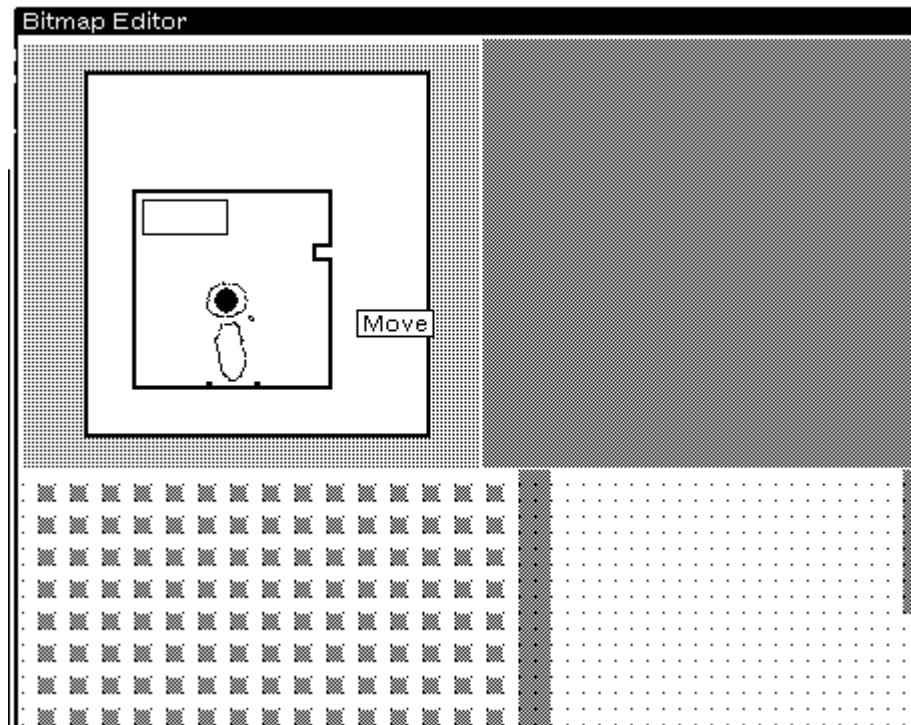


Figure 14-2. Menu with Single Item (Move)

You will be asked to position a ghost window over the bitmap. This ghost window represents the portion of the bitmap that you are currently editing. Place it over the section of the bitmap that you wish to edit and click the left mouse button (see Figure 14-3).

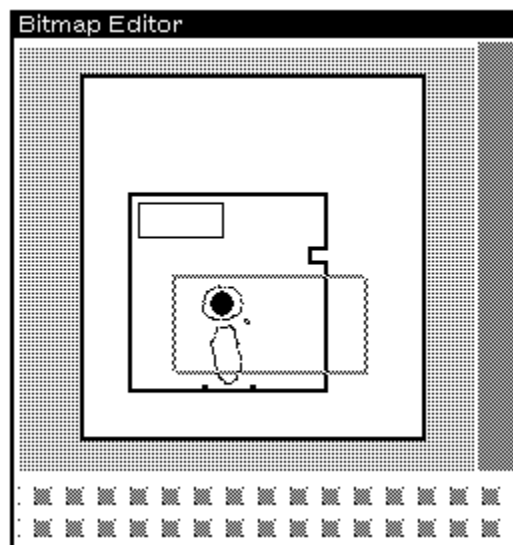


Figure 14-3. Ghost Window Awaiting Positioning

To end the session, bring the mouse cursor into the upper-right portion of the window (the grey area) and press the center button. Select OK from the menu to save your artwork.

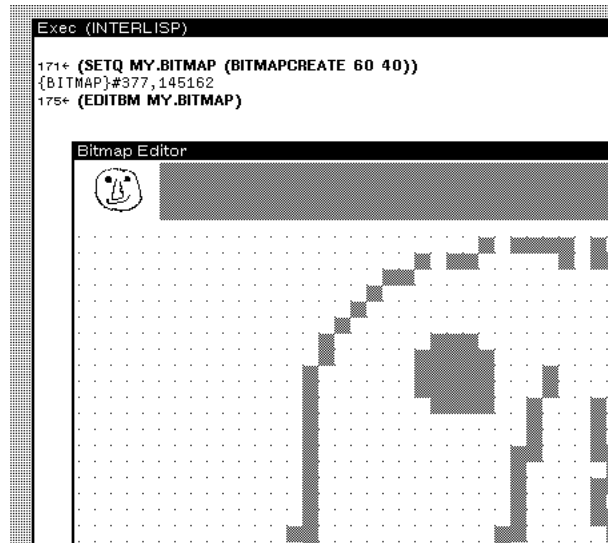


Figure 14-4. Editing a Bitmap

`BITBLT` is the primitive function for moving bits (or pixels) from one bitmap to another. It extracts bits from the source bitmap, and combines them in appropriate ways with those of the destination bitmap. The syntax of the function is:

```
(BITBLT sourcebitmap sourceleft sourcebottom destinationbitmap
destinationleft destinationbottom width height sourcetype
operation texture clippingregion)
```

Here's how it's done—using `MY.BITMAP` as the `sourcebitmap` and `MY.WINDOW` as the `destinationbitmap`.

```
(BITBLT MY.BITMAP NIL NIL
MY.WINDOW NIL NIL NIL NIL 'INPUT 'REPLACE)
```

Note that the destination bitmap can be, and usually is, a window. Actually, it is the bitmap of a window, but the system handles that detail for you. Because of the `NILs` (meaning "use the default"), `MY.BITMAP` will be `BITBLT`'d into the lower right corner of `MY.WINDOW` (see Figure 14-5).



Figure 14-5. BITBLTing a Bitmap onto a Window

Here is what each of the BITBLT arguments to the function mean:

sourcebitmap	The bitmap to be moved into the destinationbitmap
sourceleft	A number, starting at 0 for the left edge of the sourcebitmap, that tells BITBLT where to start moving pixels from the sourcebitmap. For example, if the leftmost 10 pixels of sourcebitmap were not to be moved, sourceleft should be 10. The default value is 0.
sourcebottom	A number, starting at 0 for the bottom edge of the sourcebitmap, that tells BITBLT where to start moving pixels from the sourcebitmap. For example, if the bottom 10 rows of pixels of sourcebitmap were not to be moved, sourcebottom should be 10. The default value is 0.
destinationbitmap	The bitmap that will receive the sourcebitmap. This is often a window (actually the bitmap of a window, but Interlisp-D takes care of that for you).
destinationleft	A number, starting at 0 for the left edge of the destinationbitmap, that tells BITBLT where to start placing pixels from the sourcebitmap. For example, to place the sourcebitmap 10 pixels in from the left, destinationleft should be 10. The default value is 0.

destinationbottom	A number, starting at 0 for the bottom edge of the destinationbitmap, that tells BITBLT where to start placing pixels from the sourcebitmap. For example, to place the sourcebitmap 10 pixels up from the bottom, destinationbottom should be 10. The default value is 0.
width	How many pixels in each row of sourcebitmap should be moved. The same amount of space is used in destinationbitmap to receive the sourcebitmap. If this argument is NIL, it defaults to the number of pixels from sourceleft to the end of the row of sourcebitmap.
height	How many rows of pixels of sourcebitmap should be moved. The same amount of space is used in destinationbitmap to receive the sourcebitmap. If this argument is NIL, it defaults to the number of rows from sourcebottom to the top of the sourcebitmap.
sourcetype	Refers to one of three ways to convert the sourcebitmap for writing. For now, just use ' INPUT.
operation	Refers to how the sourcebitmap gets BITBLT'd on to the destinationbitmap. ' REPLACE will BLT the exact sourcebitmap. Other operations allow you to AND, OR or XOR the bits from the sourcebitmap onto the bits on the destinationbitmap.
texture	Just use NIL for now.
clippingregion	Just use NIL for now.

For more information on these operations, see Chapter 27 in the *IRM*.

15. DISPLAYSTREAMS

A displaystream is a generalized "place to display". They determine exactly what is displayed where. One example of a displaystream is a window. Windows are the only displaystreams that will be used in this chapter. If you want to draw on a bitmap that is not a window, other than with BITBLT, or want to use other types of displaystreams, please refer to Chapter 27 in the *IRM*.

This chapter explains functions for drawing on displaystreams: DRAWLINE, DRAWTO, DRAWCIRCLE., and FILLCIRCLE. In addition, functions for locating and changing your current position in the displaystream are covered: DSPXPOSITION, DSPYPOSITION, and MOVETO.

Drawing on a Displaystream

The examples below show you how the functions for drawing on a display stream work. First, create a window. Windows are displaystreams, and the one you create are used for the examples in this chapter. Type:

```
(SETQ EXAMPLE.WINDOW (CREATEW))
```

DRAWLINE

DRAWLINE draws a line in a displaystream. For example, type:

```
(DRAWLINE 10 15 100 150 5 'INVERT EXAMPLE.WINDOW)
```

The results should look like Figure 15-1:

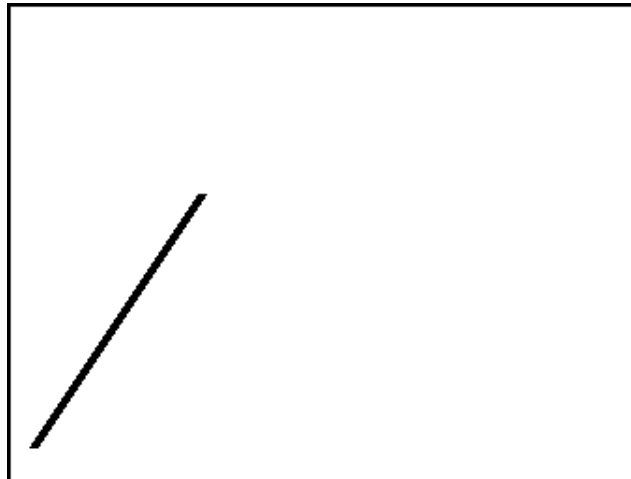


Figure 15-1. Line Drawn onto the EXAMPLE.WINDOW Displaystream

The syntax of DRAWLINE is

```
(DRAWLINE x1 y1 x2 y2 width operation stream color dashing)
```

The coordinates of the left bottom corner of the displaystream are 0 0.

x1 and y1	x and y coordinates of the beginning of the line
x2andy2	ending coordinates of the line
width	width of the line, in pixels
operation	way the line is to be drawn. <code>INVERT</code> causes the line to invert the bits that are already in the displaystream. Drawing a line the second time using <code>INVERT</code> erases the line. For other operations, see Chapter 27 in the <i>IRM</i> .
stream	displaystream. In this case, you used a window.

DRAWTO

`DRAWTO` draws a line that begins at your current position in the displaystream. For example, type:

```
(DRAWTO 120 135 5 'INVERT EXAMPLE.WINDOW)
```

The results should look like Figure 15-2:

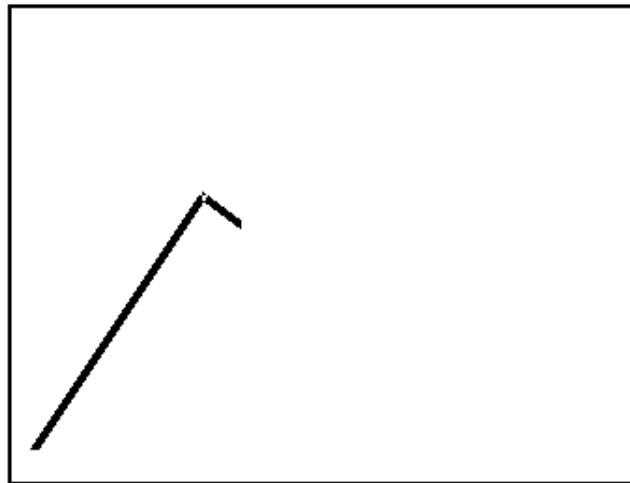


Figure 15-2. Another Line drawn onto the `EXAMPLE.WINDOW` Displaystream

The syntax of `DRAWTO` is

```
(DRAWTO x y width operation stream color dashing)
```

The line begins at the current position in the displaystream.

x	x coordinate of the end of the line
y	y coordinate of the end of the line
width	width of the line
operation	way the lino is to be drawn. <code>INVERT</code> causes the line to invert the bits that are already in the displaystream. Drawing a line the second time using <code>INVERT</code> erases the line. For other operations, see Chapter 27 in the <i>IRM</i>
stream	displaystream. In this case, you used a window.

DRAWCIRCLE

DRAWCIRCLE draws a circle on a displaystream. To use it, type:

```
(DRAWCIRCLE 150 100 30 ' (VERTICAL 5) NIL EXAMPLE.WINDOW)
```

Now your window, EXAMPLE.WINDOW, should look like Figure 15-3:

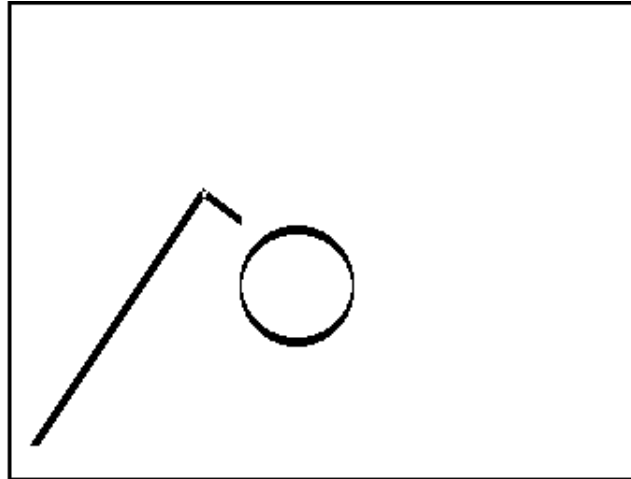


Figure 15-3. Circle Drawn onto the EXAMPLE.WINDOW Displaystream

The syntax of DRAWCIRCLE is

```
(DRAWCIRCLE centerx centery radius brush dashing stream)
```

<i>centerx</i>	x coordinate of the center of the circle
<i>centery</i>	coordinate of the center of the circle
<i>radius</i>	radius of the circle in pixels
<i>brush</i>	list.- The first- item of the list is the shape of the brush. Some of your options include ROUND, SQUARE, and VERTICAL. The second item of that list is the width of the brush in pixels.
<i>dashing</i>	list of positive integers. The brush is "on" for the number of units indicated by the first element of the list, "off" for the number of units indicated by the second element of the list. The third element specifies how long it will be on again, and so forth. The sequence is repeated until the circle has been drawn.
<i>stream</i>	displaystream. In this case, you used a window.

FILLCIRCLE

FILLCIRCLE draws a filled circle on a displaystream. To use it, type:

```
(FILLCIRCLE 200 150 10 GRAYSHADE EXAMPLE.WINDOW)
```

EXAMPLE.WINDOW now looks like Figure 15-4:

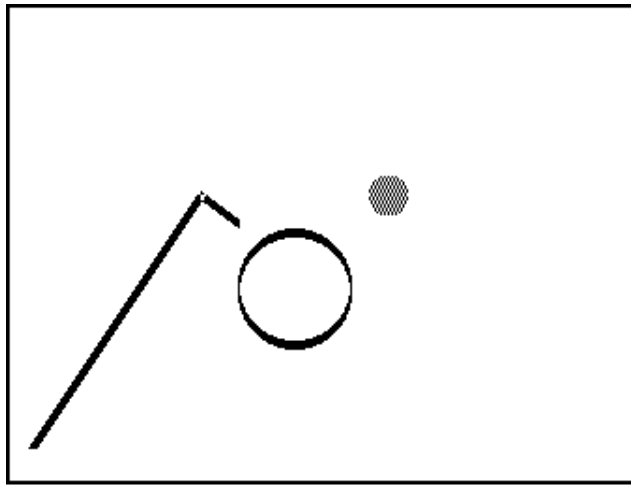


Figure 15-4. A filled circle drawn onto the displaystream

The syntax of `FILLCIRCLE` is:

`(FILLCIRCLE centerx centery radius texture stream)`

<code>centerx</code>	x coordinate of the center of the circle
<code>centery</code>	y coordinate of the center of the circle
<code>radius</code>	radius of the circle in pixels
<code>texture</code>	shade that will be used to fill in the circle. Interlisp-D provides you with three shades: <code>WHITESHADE</code> , <code>BLACKSHADE</code> , and <code>GRAYSHADE</code> . You can also create your own shades. For more information on how to do this, see Chapter 27 in the <i>IRM</i> .
<code>stream</code>	displaystream. In this case, you used a window

There are many other functions for drawing on a displaystream. Please refer to Chapter 27 in the *IRM*.

Text can also be placed into displaystreams. To do this, use printing functions such as `PRIN1` and `PRIN2`, but supply the name of the displaystream as the "file" to print to. To place the text in the proper position in the displaystream, see the section below.

Locating and Changing Your Position in a Displaystream

There are functions provided to locate, and to change your current position in a displaystream. This can help you place text, and other images where you want them in a displaystream. This primer will only discuss three of these. There are others, and they can be found in the Chapter 27 of the *IRM*.

DSPXPOSITION

`DSPXPOSITION` is a function that will either change the current x position in a displaystream, or simply report it. To have the function report the current x position in `EXAMPLE.WINDOW`, type:

```
(DSPXPOSITION NIL EXAMPLE.WINDOW)
```

DSPXPOSITION expects two arguments. The first is the new x position. If this argument is NIL, the current position is not changed, merely reported. The second argument is the displaystream.

DSPYPOSITION

DSPYPOSITION is an analogous function, but It changes or reports the current y position in a displaystream. As with DSPXPOSITION, if the first argument is a number, the current y position will be changed to that position. If it is NIL, the current position is simply reported. To have the function report the current y position in EXAMPLE.WINDOW, type:

```
(DSPYPOSITION NIL EXAMPLE.WINDOW)
```

MOVETO

The function MOVETO always changes your position in the displaystream. It expects three arguments:

```
(MOVETO x y stream)
```

x	new x position in the display stream
y	new y position in the display stream
stream	display stream. The examples so far have used a window

16. FONTS

This chapter explains fonts and fontdescriptors, what they are and how to use them, so that you can use functions requiring fontdescriptors

You have already been exposed to many fonts in Medley. For example, when you use the structure editor, DEdit (see the Using the List Structure Editor section of Chapter 7), you noticed that the comments were printed in a smaller font than the code, and that CLISP words (see the CLISP section of Chapter 9) were printed in a darker font than the other words in the function. These are only some of the fonts that are available in Medley.

In addition to the fonts that appear on your screen, Medley uses fonts for printers that are different than the ones used for the screen. The fonts used to print to the screen are called DISPLAYFONTS. The fonts used for printing are called INTERPRESSFONTS, or PRESSFONTS, depending on the type of printer.

What Makes Up a Font

Fonts are described by family, weight, slope, width, and size. This section discusses each of these, and describes how they affect the font you see on the screen.

Family is one way that fonts can differ. Here are some examples of how "family" affects the look of a font:

CLASSIC	This family makes the word "Able" look like this: Able
MODERN	This family makes the word "Able" look like this: Able
TITAN	This family makes the word "Able" look like this: Able

Weight also determines the look of a font. Once again, "Able" will be used as an example, this time only with the Classic family. A font's weight can be:

BOLD	And look like this: Able
MEDIUM or REGULAR	And look like this: Able

The slope of a font is italic or regular. Using the Classic family font again, in a regular weight, the slope affects the font like this:

ITALIC	Looks like this: <i>Able</i>
REGULAR	Looks like this: Able

The width of a font is called its "expansion". It can be COMPRESSED, REGULAR, or EXPANDED.

Together, the weight, slope, and expansion of a font specifies the font's "face". Specifically, the face of a font is a three element list:

(weight slope expansion)

To make it easier to type, when a function requires a font face as an argument, it can be abbreviated with a three-character atom. The first specifies the weight, the second the

slope, and the third character the expansion. For example, some common font faces are abbreviated:

MRR	This is the usual face, MEDIUM, REGULAR, REGULAR
MIR	Makes an italic font. It stands for: MEDIUM, ITALIC, REGULAR
BRR	Makes a bold font. The abbreviation means: BOLD, REGULAR, REGULAR
BIR	Means that the font should be both bold and italic. BIR stands for BOLD, ITALIC, REGULAR

The above examples are used so often, that there are also more mnemonic abbreviations for them. They can also be used to specify a font face for a function that requires a face as an argument. They are:

STANDARD	This is the usual face: MEDIUM, REGULAR, REGULAR; it was abbreviated above, MRR
ITALIC	This was abbreviated above as MIR, and specifies an italic font
BOLD	Makes a bold font; it was abbreviated above, BRR
BOLDITALIC	Makes a font both bold and italic: BOLD, ITALIC, REGULAR; it was abbreviated above, BIR

A font also has a size. It is a positive integer that specifies the height of the font in printers points. A point is, on an 1108 screen, about 1/72 of an inch. On the screen of an 1186, a point is 1/80 of an inch. The size of the font used in this chapter is 10. For comparison, here is an example of a TITAN, MRR, size 12 font: Able.

Fontdescriptors and FONTCREATE

For Medley to use a font, it must have a fontdescriptor. A fontdescriptor is a data type in Interlisp-D that holds all the information needed in order to use a particular font. When you print out a fontdescriptor, it looks like this:

```
{FONTDESCRIPTOR}#74,45540
```

Fontdescriptors are created by the function FONTCREATE. For example,

```
(FONTCREATE 'HELVETICA 12 'BOLD)
```

creates a fontdescriptor that, when used by other functions, prints in **HELVETICA BOLD** size 12. Interlisp-D functions that work with fonts expect a fontdescriptor produced with the FONTCREATE function.

The syntax of FONTCREATE is:

```
(FONTCREATE family size face)
```

Remember from the previous section, face is either a three element list (weight slope expansion), a three character atom abbreviation, e.g. MRR, or one of the mnemonic abbreviations, e.g. STANDARD.

If FONTCREATE is asked to create a fontdescriptor that already exists, the existing fontdescriptor is simply returned.

Display Fonts

Display fonts require files that contain the bitmaps used to print each character on the screen. All of these files have the extension `.DISPLAYFONT`. The file name itself describes the font style and size that uses its bitmaps. For example:

`MODERN12.DISPLAYFONT`

contains bitmaps for the font family MODERN in size 12 points. Wherever you put your `.DISPLAYFONT` files, you should make this one of the values of the variable `DISPLAYFONTDIRECTORIES`. Its value is a list of directories to search for the bitmap files for display fonts. Usually, it contains the "FONT" directory where you copied the bitmap files, and the current connected directory. The current connected directory is specified by the atom `NIL`. When looking for a `.DISPLAYFONT` file, the system checks the FONT directory on the hard disk, then the current connected directory.

Figure 16-1 shows an example value of `DISPLAYFONTDIRECTORIES`:

```
Exec (INTERLISP)

183← DISPLAYFONTDIRECTORIES
({{dsk}/users/sybalsky/sd/" "{dsk}/usr/local/ide/Li
spcore>XeroxPrivate>Fonts>"

    "{Pallas:mv:envos}<Fonts>display>presentation>"
    "{Pallas:mv:envos}<Fonts>display>publishing>"
    "{Pallas:mv:envos}<Fonts>display>printwheel1>"

    "{Pallas:mv:envos}<Fonts>display>miscellaneous>"
    "{Pallas:mv:envos}<Fonts>display>JIS1>"
    "{Pallas:mv:envos}<Fonts>display>JIS2>"
    "{Pallas:mv:envos}<Fonts>display>CHINESE>")
184←
```

Figure 16-1. Value for the Atom `DISPLAYFONTDIRECTORIES`

InterPress Fonts

InterPress is the format that is used by Xerox laser printers. These printers normally have a resolution that is much higher than that of the screen: 300 points per inch.

To format files appropriately for output on such a printer, Interlisp must know the actual size for each character that is to be printed. This is done through the use of width files that contain font width information for fonts in InterPress format. For InterPress fonts, you should make the location of these files one of the values of the variable `INTERPRESSFONTDIRECTORIES`. Its value is a list of directories to search for the font widths files for InterPress fonts. Figure 16-2 is an example value of `INTERPRESSFONTDIRECTORIES`:

```
Exec (INTERLISP)

184+ INTERPRESSFONTDIRECTORIES
({dsk}/users/sybalsky/sd/" "{dsk}/usr/local/1de/Li
spcore>XeroxPrivate>Fonts)"
  "{Pallas:mv:envos}<Fonts>interpress>presenta
tion)"
  "{Pallas:mv:envos}<Fonts>interpress>publishing)"
  "{Pallas:mv:envos}<Fonts>interpress>printwheel)"
  "{Pallas:mv:envos}<Fonts>interpress>miscella
neous)"
  "{Pallas:mv:envos}<Fonts>interpress>JIS1)"
  "{Pallas:mv:envos}<Fonts>interpress>JIS2)"
  "{Pallas:mv:envos}<Fonts>interpress>CHINESE)"
)
185+
```

Figure 16-2. Value for Atom INTERPRESSFONTDIRECTORIES

Functions for Using Fonts

FONTPROP Looking at Font Properties

It is possible to see the properties of a fontdescriptor. This is done with the function FONTPROP. For the following examples, the fontdescriptor used will be the one returned by the function (DEFAULTFONT 'DISPLAY). In other words, the fontdescriptor examined will be the default display font for the system.

There are many properties of a font that might be useful for you. Some of these are:

FAMILY To see the family of a font descriptor, type:

```
(FONTPROP (DEFAULTFONT 'DISPLAY) 'FAMILY)
```

SIZE As above, this is a positive integer that determines the height of the font in printer's points. As an example, the SIZE of the current default font is:

```
Exec (INTERLISP)

129+ (FONTPROP (DEFAULTFONT 'DISPLAY) 'SIZE)
10
130+
```

Figure 16-3. Value of Font Property SIZE of Default Font

ASCENT The value of this property is a positive integer, the maximum height of any character in the specified font from the baseline (bottom). The top of

the tallest character in the font, then, will be at $(\text{BASELINE} + \text{ASCENT} - 1)$. For example, the ASCENT of the default font is:



```
Exec (INTERLISP)
128← (FONTPROP (DEFAULTFONT 'DISPLAY) 'ASCENT)
9
128←
```

Figure 16-4. Value Font Property ASCENT of Default Font

- DESCENT** The DESCENT is an integer that specifies the maximum number of points that a character in the font descends below the baseline (e.g., letters such as "p" and "g" have tails that descend below the baseline.). The bottom of the lowest character in the font will be at $(\text{BASELINE} - \text{DESCENT})$. To see the DESCENT of the default font, type:
- ```
(FONTPROP (DEFAULTFONT 'DISPLAY) 'DESCENT)
```
- HEIGHT** HEIGHT is equal to  $(\text{DESCENT} - \text{ASCENT})$ .
- FACE** The value of this property is a list of the form *(weight slope expansion)*. These are the weight, slope, and expansion described above. You can see each one separately, also. Use the property that you are interested in, WEIGHT, SLOPE, or EXPANSION, instead of FACE as the second argument to FONTPROP.

For other font properties, see Chapter 27 of the *IRM*.

## STRINGWIDTH

It is often useful to see how much space is required to print an expression in a particular font. The function STRINGWIDTH does this. For example, type:

```
(STRINGWIDTH "Hi there!" (FONTCREATE 'GACHA 10 'STANDARD))
```

The number returned is how many left to right pixels would be needed if the string were printed in this font. (Note that this doesn't just work for pixels on the screen, but for all kinds of streams. For more information about streams, see Chapter 15.) Compare the number returned from the example call with the number returned when you change GACHA to TIMESROMAN.

## DSPFONT - Changing the Font in One Window

The function DSPFONT changes the font in a single window. As an example of its use, first create a window to write in. Type:

```
(SETQ MY.FONT.WINDOW (CREATEW))
```

in the Executive Window. Sweep out the window. To print something in the default font, type:

```
(PRINT 'HELLO MY.FONT.WINDOW)
```

in the Executive Window. Your window, MY.FONT.WINDOW, will look something like Figure 16-5:

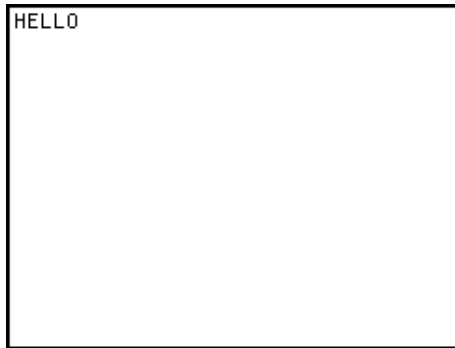
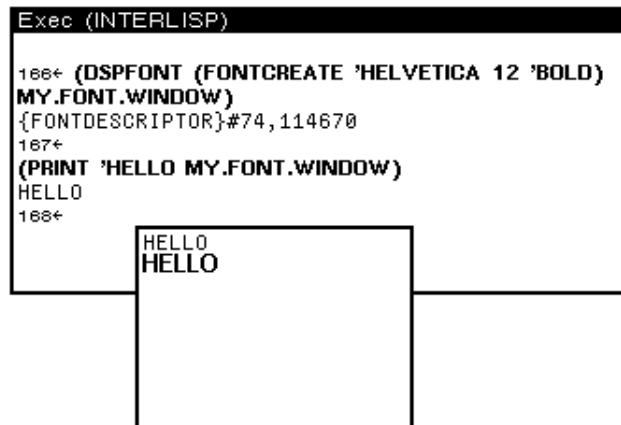


Figure 16-5. HELLO, Printed with the Default Font in MY.FONT.WINDOW

Now change the font in the window. Type:

```
(DSPFONT (FONTCREATE 'HELVETICA 12 'BOLD) MY.FONT.WINDOW)
```

in the Executive Window. The arguments to FONTCREATE can be changed to create any desired font. Now retype the PRINT statement, and your window will look something like Figure 16-6:



Flgure 16-6. Font in MY.FONT.WINDOW Changed

Notice the font has been changed.

## Personalizing Your Font Profile

Medley keeps a list of default font specifications. This list is used to set the font in all windows where the font is not specifically set by the user (see the DSPFONT section above). The value of the atom FONTPROFILE is this list (see Figure 16-7).

A FONTPROFILE is a list of font descriptions that certain system functions access when printing output. It contains specifications for big fonts (used when pretty printing a function to type the function name), small fonts (used for printing comments in the editor), and various other fonts.

```

Exec (INTERLISP)

176+ FONTPROFILE
((|FILE BROWSER PROMPT| 10
 (HELVETICA 8 (MEDIUM REGULAR REGULAR
)))
 (|FILE BROWSER FONT| 9
 (GACHA 10 (MEDIUM REGULAR REGULAR)))
 (PROMPT% WINDOW 8 (GACHA 10
 (MEDIUM REGULAR
 REGULAR)))

 (DEFAULTFONT 1 (GACHA 10)
 (GACHA 8)
 (TERMINAL 8))
 (ITALICFONT 1 (HELVETICA 10 MIR)
 (GACHA 8 MIR)
 (MODERN 8 MIR))
 (BOLDFONT 2 (HELVETICA 10 BRR)
 (HELVETICA 8 BRR)
 (MODERN 8 BRR))
 (LITTLEFONT 3 (HELVETICA 8)
 (HELVETICA 6 MIR)
 (MODERN 8 MIR))
 (TINYFONT 6 (GACHA 8)
 (GACHA 6)
 (TERMINAL 6))
 (BIGFONT 4 (HELVETICA 12 BRR)
 NIL
 (MODERN 10 BRR))
 (MENUFONT 5 (HELVETICA 10))
 (COMMENTFONT 6 (HELVETICA 10)
 (HELVETICA 8)
 (MODERN 8))
 (TEXTFONT 7 (TIMESROMAN 10)
 NIL
 (CLASSIC 10)))

177+

```

Figure 16-7. Value of the Atom FONTPROFILE

The list is in the form of an association list. The font class names (e.g., DEFAULTFONT, or BOLDFONT) are the keywords of the association list. When a number follows the keyword, it is the font number for that font class.

The lists following the font class name or number are the font specifications, in a form that the function FONTCREATE can use. The first font specification list after a keyword is the specification for printing to windows. The list (GACHA 10) in the figure above is an example of the default specification for the printing to windows. The last two font specification lists are for Press and InterPress file printing, respectively. For more information, see Chapter 27 in the *IRM*.

Now, to change your default font settings, change the value of the variable FONTPROFILE. Medley has a list of profiles stored as the value of the atom FONTDEFS. Choose the profile to use, then install it as the default FONTPROFILE.

Evaluate the atom FONTDEFS and notice that each profile list begins with a keyword (see Figure 16-8). This keyword corresponds to the size of the fonts included. BIG, SMALL, and STANDARD are some of the keywords for profiles on this list—SMALL and STANDARD appear in Figure 16-8.

```
Exec (INTERLISP)

187+ FONTDEFS
[[HUGE (FONTPROFILE (DEFAULTFONT 1 (MODERN 24)
 NIL
 (TERMINAL 8))
 (BOLDFONT 2 (MODERN 24 BRR)
 NIL
 (MODERN 8 BRR))
 (LITTLEFONT 3 (MODERN 18 MRR)
 NIL
 (MODERN 8 MIR))
 (BIGFONT 4 (MODERN 36 BRR)
 NIL
 (MODERN 10 BRR))
 (TELEFONT 5 (MODERN 10 BRR))
```

Figure 16-8. Part of Value of the Atom FONTDEFS

To install a new profile from this list, follow the following example, but insert any keyword for BIG.

To use the profile with the keyword BIG instead of the standard one, evaluate the following expression:

```
(FONTSET 'BIG))
```

Now the fonts are permanently replaced. (That is, until another profile is installed.)

## 17. THE INSPECTOR

---

The Inspector is a window-oriented tool designed to examine data structures. Because Medley is such a powerful programming environment, many types of data structures would be difficult to see in any other way.

### Calling the Inspector

---

Take as an example an object defined through a sequence of pointers (i.e., a bitmap on the property list of a window on the property list of an atom in a program.)

To inspect an object named NAME, type:

```
(INSPECT 'NAME)
```

If NAME has many possible interpretations, an option menu will appear. For example, in Interlisp-D, a litatom can refer to both an atom and a function. For example, if NAME was a record, had a function definition, and had properties on its property list, then the menu would appear as in Figure 17-1.



Figure 17-1. Option Window for Inspection of NAME

If NAME were a list, then the option menu shown in Figure 17.2 would appear. The options include:

- Calling the display editor on the list
- Calling the TTY editor (see Chapter 6)
- Seeing the list's elements in a display window. If you choose this option, each element in the list will appear in the right column of the Inspector window. The left column of the Inspector window will be made up of numbers (see Figure 17-3).
- Inspecting the list as a record type (this last option would produce a menu of known record types). If you choose a record type, the items in the list will appear in the right column of the Inspector window. The left column of the Inspector window will be made up of the field names of the record.



Figure 17-2. Option Window for Inspection of List

## Using the Inspector

---

If you choose to display your data structure in an edit window, simply edit the structure and exit in the normal manner when done. If you choose to display the data structure in an inspector window, then follow these instructions:

- To select an item, point the mouse cursor at it and press the left mouse button.
- Items in the right column of an Inspector window can themselves be inspected. To do this, choose the item, and press the center mouse button.
- Items in the right column of an Inspector window can be changed. To do this, choose the corresponding item in the left column, and press the center mouse button. You will be prompted for the new value, and the item will be changed. The sequence of steps is shown in Figure 17-3.

The item in the left column is selected, and the middle mouse button pressed. Select the SET option from the menu that pops up.

You will then be prompted for the new value. Type it in.

The item in the right column is updated to the value of what you typed in.

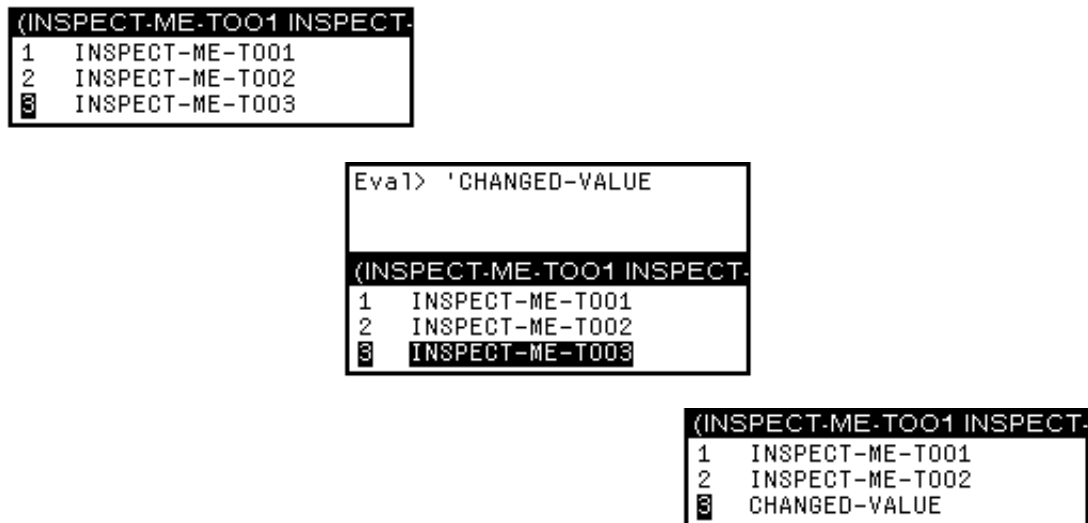


Figure 17-3. Steps Involved in Changing Value in Right Column of Inspector Window

## Inspector Example

---

This example will use ideas discussed in Chapter 21. An example, `ANIMALGRAPH`, is created in that section. You do not need to know the details of how it was created, but the structure is examined in this chapter.

If you type

```
(INSPECT ANIMAL.GRAPH)
```

and then choose the Inspect option from the menu, a display appears as shown in Figure 17-4. `ANIMAL.GRAPH` is being inspected as a list. Note the numbers in the left column of the inspectorwindow.

```

(((FISH & --)(BIRD & --)(CAT & --))T NIL NIL --
1 ((FISH & NIL NIL --) (BIRD & NIL NIL
2 T
3 NIL
4 NIL
5 NIL
6 NIL
7 NIL
8 NIL
9 NIL
10 NIL
11 NIL
12 NIL

```

Figure 17-4. Inspector Window For ANIMAL.GRAPH, Inspected as List

If you choose the "As A Record" option, and choose "GRAPH" from the menu that appears, the inspector window looks like Figure 17-5. Note the fieldnames in the left column of the inspector window.

```

(((FISH & --)(BIRD & --)(CAT & --))T NIL NIL --) Inspector
GRAPH.PROPS NIL
GRAPH.CHANGELABELFN NIL
GRAPH.INVERTLABELFN NIL
GRAPH.INVERTBORDERFN NIL
GRAPH.FONTCHANGEFN NIL
GRAPH.DELETELINKFN NIL
GRAPH.ADDLINKFN NIL
GRAPH.DELETENODEFN NIL
GRAPH.ADDNODEFN NIL
GRAPH.MOVENODEFN NIL
DIRECTEDFLG NIL
SIDESFLG T
GRAPHNODES ((FISH & NIL NIL --) (BIRD & NIL NIL

```

Figure 17-5. Inspector Window for ANIMAL.GRAPH, Inspected as Instance of GRAPH Record

The remaining examples will use ANIMAL.GRAPH inspected as a list. When the first item in the Inspector window is chosen with the left mouse button, the Inspector window looks like Figure 17-6.

```

(((FISH & --)(BIRD & --)(CAT & --))T NIL NIL --
1 (((FISH & NIL NIL --) (BIRD & NIL NIL
2 T
3 NIL
4 NIL
5 NIL
6 NIL
7 NIL
8 NIL
9 NIL
10 NIL
11 NIL
12 NIL

```

Figure 17-6. Inspector Window for ANIMAL.GRAPH With First Element Selected

When you use the middle mouse button to inspect the selected list element, the display looks like Figure 17-7.

```

(((FISH & --) (BIRD & --) (CAT & --) --) T NIL NIL --
1 ((FISH & NIL NIL --) (BIRD & NIL NIL
2 T
3 NIL
4 (((FISH (102 . 48) NIL --) (BIRD (102 . 32) NIL --) (C
5 1 (FISH (102 . 48) NIL NIL NIL --)
6 2 (BIRD (102 . 32) NIL NIL NIL --)
7 3 (CAT (186 . 24) NIL NIL NIL --)
8 4 (DOG (178 . 10) NIL NIL NIL --)
9 5 ((MAMMAL DOG CAT) (109 . 16) NIL NIL
10 6 ((ANIMAL & BIRD FISH) (22 . 32) NIL
11 NIL
12 NIL

```

Figure 17-7. Inspector Window for ANIMAL.GRAPH and for First Element of ANIMAL.GRAPH

How you can see that six items make up the list, and you can further choose to inspect one of these items. Notice that this is also inspected as a list. As usual, it could also have been inspected as a record.

Select item 5 - MAMMAL DOG CAT - with the left mouse button. Press the middle mouse button. Choose "Inspect" to inspect your choice as a list. The Inspector now displays the values of the structure that makes up MAMMAL DOG CAT. (See Figure 17-8.)

```

((MAMMAL DOG CAT) (109 . 16)
1 (MAMMAL DOG CAT)
2 (109 . 16)
3 NIL
4 NIL
5 NIL
6 45
7 16
8 (DOG CAT)
9 ((ANIMAL & BIRD FISH))
10 {FONTCLASS}#74,61752
11 MAMMAL
12 NIL

```

Figure 17-8. Inspector Window for Element 5 From Figure 17.7 That Begins ( (MAMMAL DOG CAT).



## 18. MASTERSCOPE

---

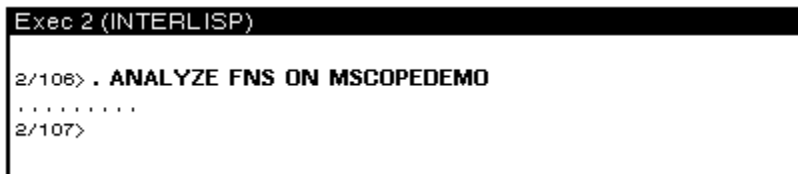
Masterscope is a tool that allows you to quickly examine the structure of complex programs. As your programs enlarge, you may forget what variables are global, what functions call other functions, and so forth. Masterscope keeps track of this for you.

To use Masterscope, first load `MASTERSCOPE.DFASL` and `EXPORTS.ALL`.

Suppose that `JVTO` is the name of a file that contains many of the functions involved in a complex system and that `LINTRANS` is the file containing the remaining functions. The first step is to ask Masterscope to analyze these files. These files must be loaded. All Masterscope queries and commands begin with a period followed by a space, as in

```
. ANALYZE FNS ON MSCOPEDEMO
```

The `ANALYZE` process takes a while, so the system prints a period on the screen for each function it has analyzed. (See Figure 18-1)



```
Exec 2 (INTERLISP)
2/106> . ANALYZE FNS ON MSCOPEDEMO
.....
2/107>
```

Figure 18-1. Executive Window After Analyzing Files

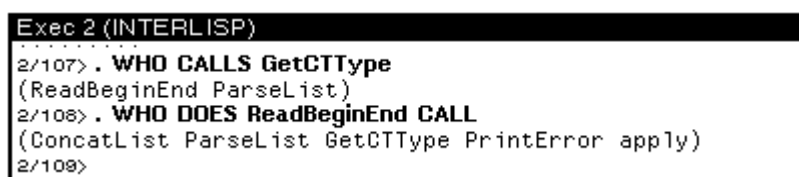
If you are not quite sure what functions were just analyzed, type the file's `COMS` variable (see the File Variables section in Chapter 7) into the Executive Window. The names of the functions stored on the file will be a part of the value of this variable.

A variety of commands are now possible, all referring to individual functions within the analyzed files. Substantial variation in exact wording is permitted. Some commands are:

- . SHOW PATHS FROM ANY TO ANY
- . EDIT WHERE ANY CALLS *functionname*
- . EDIT WHERE ANY USES *variablename*
- . WHO CALLS WHOM
- . WHO CALLS *functionname*
- . BY WHOM IS *functionname* CALLED
- . WHO USES *variablename* AS FIELD

Note that the function is being called to invoke each command. Refer to the *IRM* for commands not listed here.

Figure 18-2 shows the Executive Window after the commands `. WHO CALLS GobbleDump` and `. WHO DOES JVL inScan CALL`.



```
Exec 2 (INTERLISP)
2/107> . WHO CALLS GetCTType
(ReadBeginEnd ParseList)
2/108> . WHO DOES ReadBeginEnd CALL
(ConcatList ParseList GetCTType PrintError apply)
2/109>
```

Figure 18-2. Sample Masterscope Output

## SHOW DATA Command and GRAPHER

When the library package GRAPHER is loaded (to load this package, type (FILESLOAD GRAPHER)), Masterscope's SHOWPATHS command is modified. The command will be changed to generate a tree structure showing how the program's functions interact instead of a tabular printout into the Executive window. For example, typing:

```
. SHOW PATHS FROM ProcessEND
```

produced the display shown in Figure 18-3.

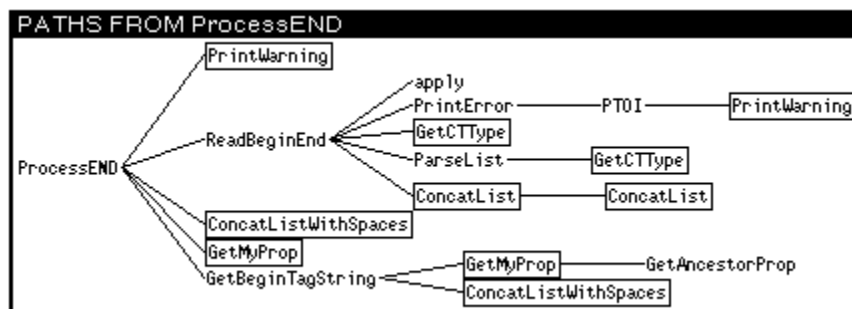
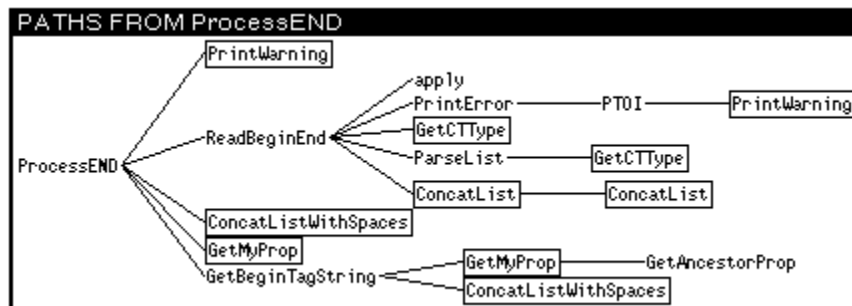


Figure 18-3. SHOW PATHS Display Example

All the functions in the display are part of this analyzed file or a previously analyzed file. Boxed functions indicate that the function name has been duplicated in another place on the display.

Selecting any function name on the display will pretty print the function in a window (see Figure 18-4).



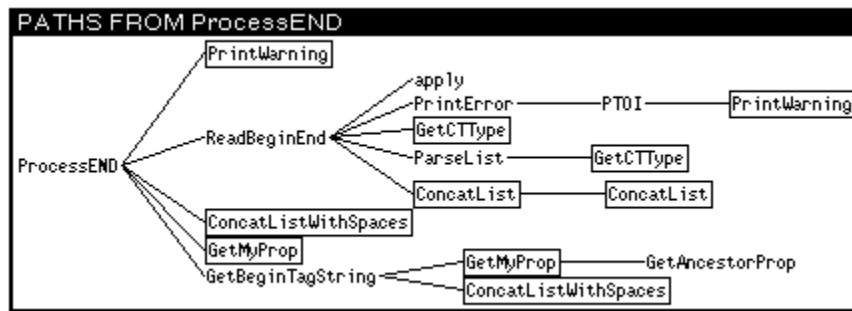
```

Browser print out window
(GetMyProp
[LAMBDA (PropName) ; Edited 20-Feb-92 22:14 by
 ; welch
 (GetAncestorProp PropName (CAR T0Istack]))

```

Figure 18-4. Browser Printout Example

Selecting it again with the left mouse button will produce a description of the function's role in the overall system (see Figure 18-5).



**Browser describe window**

```

(GetMyProp PropName)
calls: GetAncestorProp
called by: ProcessEND,
 GetBeginTagString
uses free: T0Istack

```

Figure 18-5. Browser Description Example

## 19. WHERE DOES ALL THE TIME GO? SPY

---

SPY is an Lisp library package that shows you where you spend your time when you run your system. It is easy to learn, and very useful when trying to make programs run faster.

### How to Use Spy with the SPY Window

---

The function `SPY.BUTTON` brings up a small window which you will be prompted to position. Using the mouse buttons in this window controls the action of the `SPY` program. When you are not using `SPY`, the window appears as in Figure 19.1.



Figure 19.1. SPY Window When SPY is Not Being Used

To use `SPY`, click either the left or middle mouse button with the mouse cursor in the `SPY` window. The window will appear as in Figure 19.2, and means that `SPY` is accumulating data about your program.



Figure 19.2. SPY Window When SPY is Being Used

To turn off `SPY` after the program has run, again click a mouse button in the `SPY` window. The eye closes, and you are asked to position another window. This window contains `SPY`'s results. An example of the resulting window is shown in Figure 19.3.

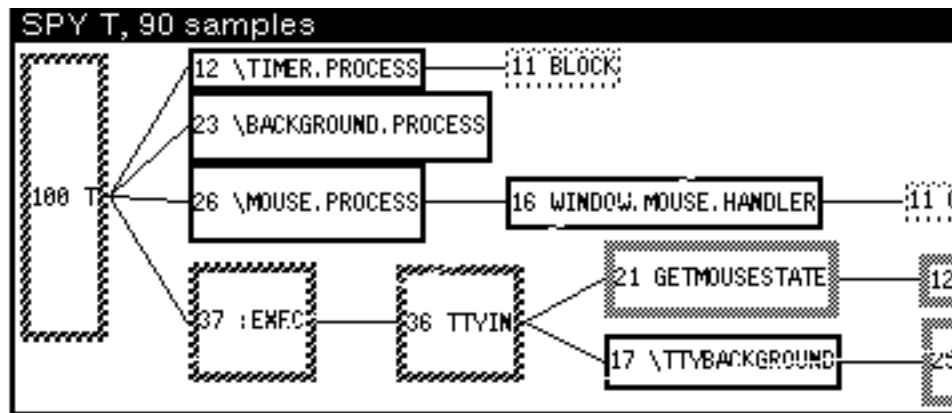


Figure 19.3. Window Produced After Running SPY

This window is scrollable horizontally and vertically. This is useful, since the whole tree does not fit in the window. If a part that you want to see is not shown, you can scroll the window to show the part you want to see.

## How to Use SPY from the Lisp Top Level

---

SPY can also be run while a specific function or system is being used. To do this, type the function `WITH.SPY`:

```
(WITH.SPY form)
```

The expression used for `form` should be the call to begin running the function or system that SPY is to watch. If you watch the SPY window, the eye will blink! To see your results, run the function `SPY.TREE`. To do this, type:

```
(SPY.TREE)
```

The results of the last running of SPY will be displayed. If you do this, and `SPY.TREE` returns (no SPY samples have been gathered), your function ran too fast for SPY to follow.

## Interpreting SPY's Results

---

Each node in the tree is a box that contains, first, the percentage of time spent running that particular function, and second, the function name. There are two modes that can be used to display this tree.

The default mode is cumulative. In this mode, each percentage is the amount of time that function spent on top of the stack, plus the amount of time spent by the functions it calls. The second mode is individual. To change the mode to individual, point to the title bar of the window, and press the middle mouse button. Choose `Individual` from the menu that appears. In this mode, the percentage shown is the amount of time the function spent on the top of the stack.

To look at a single branch of the tree, point with the mouse cursor at one of the nodes of the tree, and press the right mouse button. From the menu that appears, choose the

option `SubTree`. Another SPY window will appear, with just this branch of the tree in it.

Another way to focus within the tree is to remove branches from the tree. To do this, point to the node at the top of the branch you would like to delete. Press the middle mouse button, and choose `Delete` from the menu that appears.

There are also different amounts of "merging" of functions that can be done in the window. A function can be called by another function more than once. The amount of merging determines where the subfunction, and the functions that it calls, appear in the tree, and how often. (For a detailed explanation of merging, see the *Lisp Library Packages Manual*.)

## 20. FREE MENUS

---

Free Menu is a library package that is even more flexible than the regular menu package. It allows you to create menus with different types of items in them, and formats them as you require. Free menus are particularly useful when you want a "fill in the form" type interaction with the user.

Each menu item is described with a list of properties and values. The following example will give you an idea of the structure of the description list, and some of your options. The most commonly used properties, and each type of menu item will be described in the Parts of a Free Menu Item and Types of Free Menu Items section below.

### Free Menu Example

---

Free menus can be created and formatted automatically! It is done with the function `FM.FORMATMENU`. This function takes one argument, a description of the menu. The description is a list of lists; each internal list describes one row of the free menu. A free menu row can have more than one item in it, so there are really lists of lists of lists! It really isn't hard, though, as you can see from the following example:

```
(SETQ ExampleMenu
 (FM.FORMATMENU
 '(((TYPE TITLE LABEL TitlesDoNothing)
 (TYPE 3STATE LABEL Example3State))
 ((TYPE EDITSTART LABEL PressToStartEditing
 ITEMS (EDITITEM))
 (TYPE EDIT ID EDITITEM LABEL ""))
 (WINDOWPROPS TITLE "Example Does Nothing"))))
```

The first row has two items in it: one is a `TITLE`, and the second is a `3STATE` item. The second row also has two items. The second, the `EDIT` item, is invisible, because its label is an empty string. The caret will appear for editing, however, if the `EDITSTART` item is chosen. `Windowprops` can appear as part of the description of the menu, because a menu is, after all, just a special window. You can specify not only the title with `WINDOWPROPS`, but also the position of the free menu, using the "left" and "bottom" properties, and the width of the border in pixels, with the "border" property. Evaluating this expression will return a window. You can see the menu by using the function `OPENW`. The following example illustrates this:

Figure 20.1. Example Free Menu

The next example shows you what the menu looks like after the `EDITSTART` item, `PressToStartEditing`, has been chosen.

Figure 20.2. Free menu after `EDITSTART` Item Chosen

The following example shows the menu with the `3STATE` item in its `T` state, with the item highlighted. (In the previous bitmaps, it was in its neutral state.)

Figure 20.3. Free menu with 3STATE Item in its T State

Finally, Figure 20.4 shows the 3STATE item in its NIL state, with a diagonal line through the item

Figure 20.4 Free menu with the 3STATE item in its NIL State

If you would like to specify the layout yourself, you can do that too. See the *Lisp Library Packages Manual* for more information.

---

## Parts of a Free Menu Item

There are eight different types of items that you can use in a free menu. No matter what type, the menu item is easily described by a list of properties, and values. Some of the properties you will use most often are listed below:

|            |                                                                                                                         |
|------------|-------------------------------------------------------------------------------------------------------------------------|
| LABEL      | Required for every type of menu item. It is the atom, string, or bitmap that appears as a menu selection.               |
| TYPE       | One of eight types of menu items. Each of these are described in the section below.                                     |
| MESSAGE    | The message that appears in the prompt window if a mouse button is held down over the item.                             |
| ID         | An item's unique identifier. An ID is needed for certain types of menu items.                                           |
| ITEMS      | Used to list a series of choices for an NCHOOSE item, and to list the ID's of the editable items for an EDITSTART item. |
| SELECTEDFN | The name of the function to be called if the item is chosen.                                                            |

---

## Types of Free Menu Items

Each type of menu item is described in the following list, including an example description list for each one.

|           |                                                                                                                                                                                                                                                                                                                    |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MOMENTARY | <p>This is the familiar sort of menu item. When it is selected, the function stored with it is called. A description for the function that creates and formats the menu looks like this:</p> <pre>(TYPE MOMENTARY   LABEL Blink-N-Ring   MESSAGE "Blinks the screen and rings bells"   SELECTEDFN RINGBELLS)</pre> |
| TOGGLE    | <p>This menu item has two states, T and NIL. The default state is NIL, but choosing the item toggles its state. The following is an example description list, without code for the SELECTEDFN function, for this type of item:</p> <pre>(TYPE TOGGLE</pre>                                                         |



|           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           | <pre>         LABEL DwimDisable         SELECTEDFN ChangeDwimState) </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 3STATE    | <p>This type of menu item has three states, NEUTRAL, T, and NIL. NEUTRAL is the default state. T is shown by highlighting the item, and NIL is shown with diagonal lines. The following is an example description list, without code for the SELECTEDFN function, for this type of item:</p> <pre> (TYPE 3STATE   LABEL CorrectProgramAllOrNoSpelling   SELECTEDFN ToggleSpellingCorrection) </pre>                                                                                                                                                                                                                                                                |
| TITLE     | <p>This menu item appears on the menu as dummy text. It does nothing when chosen. An example of its description:</p> <pre> (TYPE TITLE LABEL "Choices:") </pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| NWAY      | <p>A group of items, nnly one of which can be chosen at a time. The items in the NWAY group should all have an ID field, and the ID's should be the same. For example, to set up a menu that would allow the user to choose between Helvetica, Gacha, Modern, and Classic fonts, the descriptions might look like this (once again, without the code for the SELECTEDFN):</p> <pre> (TYPE NWAY ID FONTCHOICE   LABEL Helvetica   SELECTEDFN ChangeFont) (TYPE NWAY ID FONTCHOICE   LABEL Gacha   SELECTEDFN ChangeFont) (TYPE NWAY ID FONTCHOICE)   LABEL Modern   SELECTEDFN ChangeFont) (TYPE NWAY ID FONTCHOICE   LABEL Classic   SELECTEDFN Changefont) </pre> |
| NCHOOSE   | <p>This type of menu item is like NWAY except that the choices are given to the user in a submenu. The list to specify an NCHOOSE menu item that is analogous to the NWAY item above might look like this:</p> <pre> (TYPE NCHOOSE   LABEL FontChoices   ITEMS Helvetica Gacha Modern Classic)   SELECTDFN Changefont) </pre>                                                                                                                                                                                                                                                                                                                                      |
| EDITSTART | <p>When this type of menu itein is chosen, it activates another type of item, an EDIT item. The EDIT item or items associated with an EDITSTART item have their ID's listed on the EDITSTART's ITEMS property. An example description list is:</p> <pre> (TYPE EDITSTART LABEL "Function to add?" ITEMS (Fn)) </pre>                                                                                                                                                                                                                                                                                                                                               |
| EDIT      | <p>This type of menu item can actually be edited by you. It is often associated with an EDITSTART item (see above), but the caret that prompts for input will also appear if the item itself is chosen. An EDIT item follows the same editing conventions as editing in Executive Window:</p> <p><b>Add characters</b> by typing them at the caret.</p>                                                                                                                                                                                                                                                                                                            |

**Move the caret** by pointing the mouse at the new position, and clicking the left button.

**Delete characters** from the caret to the mouse by pressing the right button of the mouse. Delete a character behind the caret by pressing the backspace key.

**Stop editing** by typing a carriage return, a Control-X, or by choosing another item from the menu.

An example description list for this type of item is:

```
(TYPE EDIT ID Fn LABEL **)
```

## 21. THE GRAPHER

---

### Say it with Graphs

---

Grapher is a collection of functions for creating and displaying graphs, networks of nodes and links. Grapher also allows you to associate program behavior with mouse selection of graph nodes. To load this package, type

```
(FILESLOAD GRAPHER)
```

Figure 21-1 shows a simple graph.

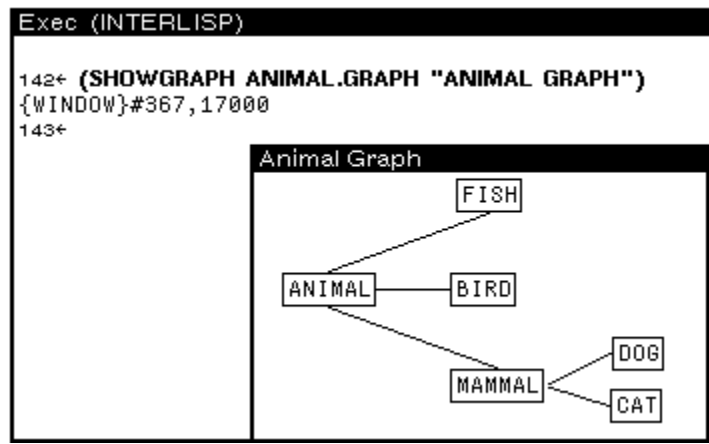


Figure 21-1. Simple Graph

In Figure 21-1 there are six nodes (ANIMAL, MAMMAL, DOG, CAT, FISH, and BIRD) connected by five links. A GRAPH is a record containing several fields. Perhaps the most important field is GRAPHNODES—which is itself a list of GRAPHNODE records. Figure 21-2 illustrates these data structures. The window on top contains the fields from the simple graph. The window on the bottom an inspection of the node, DOG.

| (((FISH & --)(BIRD & --)(CAT & --)--) T NIL NIL --) Inspector |                                      |
|---------------------------------------------------------------|--------------------------------------|
| GRAPH.PROPS                                                   | NIL                                  |
| GRAPH.CHANGELABELFN                                           | NIL                                  |
| GRAPH.INVERTLABELFN                                           | NIL                                  |
| GRAPH.INVERTBORDERFN                                          | NIL                                  |
| GRAPH.FONTCHANGEFN                                            | NIL                                  |
| GRAPH.DELETELINKFN                                            | NIL                                  |
| GRAPH.ADDLINKFN                                               | NIL                                  |
| GRAPH.DELETENODEFN                                            | NIL                                  |
| GRAPH.ADDNODEFN                                               | NIL                                  |
| GRAPH.MOVENODEFN                                              | NIL                                  |
| DIRECTEDFLG                                                   | NIL                                  |
| SIDESFLG                                                      | T                                    |
| GRAPHNODES                                                    | ((FISH & NIL NIL --) (BIRD & NIL NIL |

| (DOG (178 . 10) NIL NIL --) Inspector |                                       |
|---------------------------------------|---------------------------------------|
| NODEBORDER                            | NIL                                   |
| NODELABEL                             | DOG                                   |
| NODEFONT                              | (HELVETICA 10 (MEDIUM REGULAR REGULAR |
| FROMNODES                             | ((MAMMAL DOG CAT))                    |
| TONODES                               | NIL                                   |
| NODEHEIGHT                            | 14                                    |
| NODEWIDTH                             | 31                                    |
| NODELABELSHADE                        | NIL                                   |
| NODELABELBITMAP                       | NIL                                   |
| NODEPOSITION                          | (178 . 10)                            |
| NODEID                                | DOG                                   |

Figure 21-2. Inspecting a Graph and a Node

The GRAPHNODE data structure is described by its text (NODEID), what goes into it (FROMNODES), what leaves it (TONODES), and other fields that specify its looks. The basic model of graph building is to create a bunch of nodes, then layout the nodes into a graph, and finally display the resultant graph. This can be done in a number of ways. One is to use the function NODECREATE to create the nodes, LAYOUTGRAPH to lay out the nodes, and SHOWGRAPH to display the graph. The primer shows you two simpler ways, but please see the *Library Packages Manual* for more information about these other functions. The primer's first method is to use SHOWGRAPH to display a graph with no nodes or links, then interactively add them. The second is to use the function LAYOUTSEXPR, which does the appropriate NODECREATES and a LAYOUTGRAPH, with a list.

The function SHOWGRAPH displays graphs and allows you to edit them. The syntax of SHOWGRAPH is

```
(SHOWGRAPH graph window lefbuttonfn middlebuttonfn
topjustifyflg alloweditflg copybuttoneventfn)
```

Obviously the graph structure is very complex. Here's the easiest way to create a graph.

```
(SETQ MY.GRAPH NIL)
(SHOWGRAPH MY.GRAPH "My Graph" NIL NIL NIL T)
```

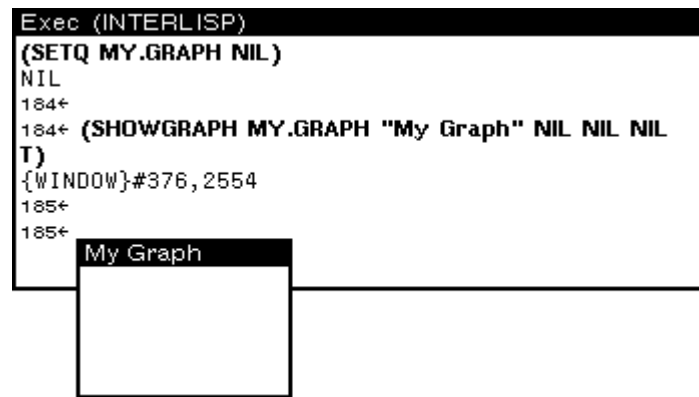


Figure 21-3. My Graph

You will be prompted to create a small window as in Figure 21-3. This graph has the title My Graph. Hold down the right mouse button in the window. A menu of graph editing operations will appear as in Figure 21-4.

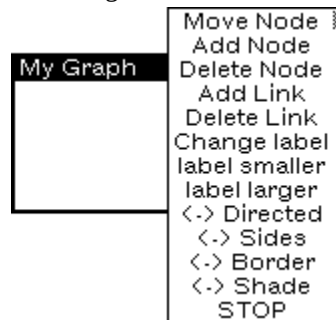


Figure 21-4. Menu of Graph Editing Operations

Here's how to use this menu. The commands in this menu are easy to learn. Experiment with them!

### Add a Node

Start by selecting Add Node. Grapher will prompt you for the name of the node (see Figure 21-5.) and then its position.

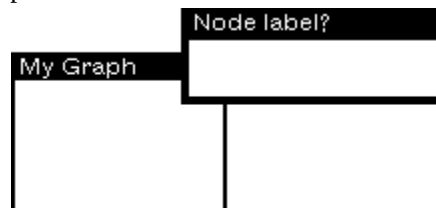


Figure 21-5. Grapher Prompts for Name of Node to add after Add Node is Chosen from Graph Editing Menu.

Position the node by moving the mouse cursor to the desired location and clicking a mouse button. Figure 21-6 shows the graph with two nodes added using this menu.

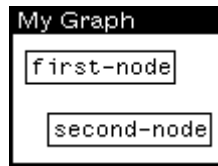


Figure 21-6. Two Nodes Added to MY GRAPH Using GraphEditing Menu

### Add a Link

Select **Add Link** from the graph editing menu. The Prompt window will prompt you to select the two nodes to be linked. (See Figure 21-7.) Do this, and the link will be added.

```
Prompt Window
Specify the link by selecting the FROM node, then the TO node.
FROM?
```

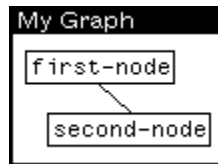


Figure 21-7. Prompt Window Requesting Selection of Two Nodes to Link, and Result

### Delete a Link

Select **Delete Link** from the graph editing menu. The Prompt window will prompt you to select the two nodes that should no longer be linked. (See Figure 21-8.) Do this, and the link will be deleted.

```
Prompt Window
Specify the link by selecting the FROM node, then the TO node.
FROM?
```

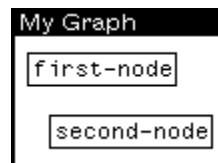


Figure 21-8. Prompt Window Requesting Selection of Link to Delete, and Result

### Delete a Node

Select **Delete Node** from the graph editing menu. The Prompt window will prompt you to select the node to be deleted. (See Figure 21-9.) Do this, and the node will be deleted.

```
Prompt Window
Select node to be deleted.
```

Figure 21-9. Prompt to Delete a Node

## Move a Node

Select `Delete Node` from the graph editing menu. Choose a node pointing to the it with the mouse cursor, and pressing and holding the left mouse button. When you move the mouse cursor, the node will be dragged along. When the node is at the new position, release the mouse button to deposit the node.

## Making a Graph from a List

---

Typically, a graph is used to display one of your program's data structures. Here is how that is done.

`LAYOUTSEXPR` takes a list and returns a `GRAPH` record. The syntax of the function is

```
(LAYOUTSEXPR sexpr format boxing font motherd personald famlyd)
```

For example:

```
(SETQ ANIMAL.TREE ' (ANIMAL (MAMMAL DOG CAT) BIRD FISH))
(SETQ ANIMAL.GRAPH
 (LAYOUTSEXPR ANIMAL.TREE 'HORIZONTAL))
(SHOWGRAPH ANIMAL.GRAPH "My Graph" NIL NIL NIL T)
```

This is how Figure 21.1 was produced.

## Incorporating Grapher into Your Program

---

The Grapher is designed to be built into other programs. It can call functions when, for example, a mouse button is clicked on a node. The function `SHOWGRAPH` does this:

```
(SHOWGRAPH graph window leftbuttonfn middlebuttonfn
 topjustifyflg alloweditflg copybuttoneventfn)
```

For example, the third argument to `SHOWGRAPH`, *leftbuttonfn*, is a function that is called when the left mouse button is pressed in the graph window. Try this:

```
(DEFINEQ (MY.LEFT.BUTTON.FUNCTION
 (THE.GRAPHNODE THE.GRAPH.WINDOW)
 (INSPECT THE.GRAPHNODE)))

(SHOWGRAPH FAMILY.GRAPH "Inspectable family"
 (FUNCTION MY.LEFT.BUTTON.FUNCTION)
 NIL NIL T)
```

In the example above, `MY.LEFT.BUTTON.FUNCTION` simply calls the inspector. The function should be written assuming it will be passed a `graphnode` and the window that holds the graph. Try adding a function of your own.

## More of Grapher

---

Some other Library packages make use of the Grapher. (Grapher needs to be loaded with the packages to use these functions.)

- MASTERSCOPE: The Browser package modifies the Masterscope command, . SHOW PATHS, so that its output is displayed as a graph (using Grapher) instead of simply printed.
- GRAPHZOOM: allows a graph to be redisplayed larger or smaller automatically.



## 22. RESOURCE MANAGEMENT

---

### Naming Variables and Records

---

You will find times when one environment simultaneously hosts a number of different programs. Running a demo of several programs, or reloading the entire Medley environment from floppies when it contains several different programs, are two examples that could, if you aren't careful, provide a few problems. Here are a few tips on how to prevent problems:

- If you change the value of a system variable, MENUHELDDWAIT for example, or connect to a directory other than {DSK}<LISPFILES>, write a function to reset the variable or directory to its original value. Run this function when you are finished working. This is especially important if you change any of the system menus.
- Do not redefine Medley functions or CLISP words. Remember, if you reset an atom's value or function definition at the top level (in the Executive Window), the message (*Some.Crucial.Function.Or.Variable* redefined), appears. If this is not what you wanted, type **UNDO** immediately!

If, however, you reset the value or function definition of an atom inside your program, a warning message will not be printed.

- Make the atom names in your programs as unique as possible. To do this without filling your program with unreadable names that noone, including you, can remember, prefix your variable names with the initials of your program. Even then, check to see that they are not already being used with the function BOUNDP. For example, type:

```
(BOUNDP 'BackgroundMenu)
```

This atom is bound to the menu that appears when you press the left mouse button when the mouse cursor is not in any window. BOUNDP returns T. BOUNDP returns NIL if its argument does not currently have a value.

- Make your function names as unique as possible. Once again, prefixing function names with the initials of your program can be helpful in making them unique, but even so, check to see that they are not already being used. GETD is the Interlisp-D function that returns the function definition of an atom, if it has one. If an atom has no function definition, GETD returns NIL. For example, type:

```
(GETD 'CAR)
```

A non-NIL value is returned. The atom CAR already has a function definition.

- Use complete record field names in record FETCHes and REPLACES when your code is not compiled. A complete record field name is a list consisting of the record declaration name and the field name. Consider the following example:

```
(RECORD NAME (FIRST LAST))
(SETQ MyName (create Name FIRST←'John LAST←'Smith))
(FETCH (NAME FIRST) OF MyName)
```

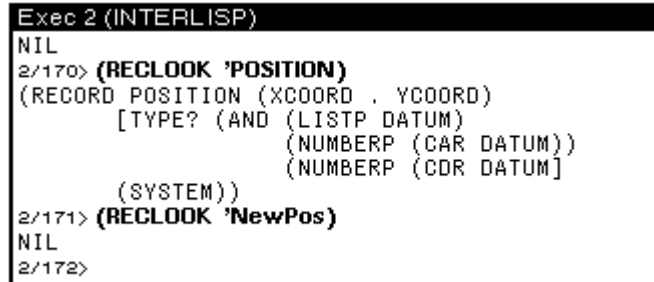
- Avoid reusing names that are field names of Lisp system records. A few examples of system records follow. Do not reuse these names.

```
(RECORD REGION (LEFT BOTTOM WIDTH HEIGHT))
```

```
(RECORD POSITION (XCOORD YCOORD))
(RECORD IMAGEOBJ (- BITMAP -)))
```

- When you select a record name and field names for a new record, check to see whether those names have already been used.

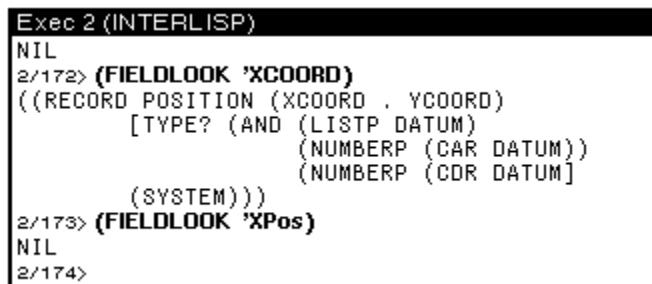
Call the function `RECLOOK`, with your record name as an argument, in the Executive Window (see Figure 22-1). If your record name is already a record, the record definition will be returned; otherwise the function will return `NIL`.



```
Exec 2 (INTERLISP)
NIL
2/170> (RECLOOK 'POSITION)
(RECORD POSITION (XCOORD . YCOORD)
 [TYPE? (AND (LISTP DATUM)
 (NUMBERP (CAR DATUM))
 (NUMBERP (CDR DATUM))
 (SYSTEM)))
2/171> (RECLOOK 'NewPos)
NIL
2/172>
```

Figure 22-1. Response to `RECLOOK`

Call the function `FIELDLOOK` with your new field name in the Executive Window (see Figure 22-2). If your field name is already a field name in another record, the record definition will be returned; otherwise the function will return `NIL`.



```
Exec 2 (INTERLISP)
NIL
2/172> (FIELDLOOK 'XCOORD)
((RECORD POSITION (XCOORD . YCOORD)
 [TYPE? (AND (LISTP DATUM)
 (NUMBERP (CAR DATUM))
 (NUMBERP (CDR DATUM))
 (SYSTEM)))
2/173> (FIELDLOOK 'XPos)
NIL
2/174>
```

Figure 22-2. Response to `FIELDLOOK`

---

## Some Space and Time Considerations

In order for your program to run at maximum speed, you must efficiently use the space available on the system. The following section points out areas that you may not know are wasting valuable space, and tips on how to prevent this waste.

Often programs are written so that new data structures are created each time the program is run. This is wasteful. Write your programs so that they only create new variables and other data structures conditionally. If a structure has already been created, use it instead of creating a new one.

Some time and space can be saved by changing your `RECORD` and `TYPERECORD` declarations to `DATATYPE`. `DATATYPE` is used the same way as the functions `RECORD` and `TYPERECORD`. In addition, the same `FETCH` and `REPLACE` commands can be used with the data structure `DATATYPE` creates. The difference is that the data structure `DATATYPE` creates cannot be treated as a list the way `RECORDs` and `TYPERECORDs` can.

## Global Variables

Once defined, global variables remain until Lisp is reloaded. Avoid using global variables if at all possible! One specific problem arises when programs use the function `GENSYM`. In program development, many atoms are created that may no longer be useful. Hints:

- Use

```
(DELDEF atomname 'PROP)
```

to delete property lists, and

```
(DELDEF atomname 'VARS)
```

to have the atom act like it is not defined.

These not only remove the definition from memory, but also change the appropriate `fileCOMS` that the deleted object was associated with so that the file package will not attempt to save the object (function, variable, record definition, and so forth) the next time the file is made. Just doing something like

```
(SETQ (arg atomname) 'NOBIND)
```

looks like it will have the same effect as the second `DELDEF` above, but the `SETQ` does not update the file package.

- If you are generating atom names with `GENSYM`, try to keep a list of the atom names that are no longer needed. Reuse these atom names, before generating new ones. There is a (fairly large) maximum to the number of atoms you can have, but things slow down considerably when you create lots of atoms.
- When possible, use a data structure such as a list or an array, instead of many individual atoms. Such a structure has only one pointer to it. Once this pointer is removed, the whole structure will be garbage-collected and space will be reclaimed.

## Circular Lists

If your program is creating circular lists, a lot of space may be wasted. (Many crosslinked data structures end up having circularities.) Hints when using circular lists:

- Write a function to remove pointers that make lists circular when you are through with the circular list.
- If you are working with circular lists of windows, bind your main window to a unique global variable. Write window creation conditionally so that if the binding of that variable is already a window, use it, and only create a new window if that variable is unbound or `NIL`.

Here is an example that illustrates the problem. When several auxiliary windows are built, pointers to these windows are usually kept on the main window's property list. Each auxiliary window also typically keeps a pointer to the main window on its property list. If the top level function creates windows rather than reusing existing ones, there will be many lists of useless windows cluttering the work space. Or, if such a main window is closed and will not be used again, you will have to break the links by deleting the relevant properties from both the main window and all of the auxiliary windows first. This is usually done by putting a special `CLOSEFN` on the main window and all of its auxiliary windows.

## When You Run Out of Space

Typically, if you generate a lot of structure that won't get garbage collected, you will eventually run out of space. The important part is being able to track down those structures and the code that generates them to become more space efficient.

Use the Lisp Library Package `GCHAX.DCOM` to track down pointers to data structures. The basic idea is that `GCHAX` will return the number of references to a particular data structure.

A special function exists that allows you to get a little extra space so that you can try to save your work when you get toward the edge (usually noted by a message indicating that you should save your work and load a new Medley environment). The `GAINSPACE` function allows you to delete non-essential data structures. To use it, type:

`(GAINSPACE)`

into the Executive Window. Answer `N` to all questions except the following.

- Delete edit history
- Delete history list.
- Delete values of old variables.
- Delete your `MASTERSCOPE` database
- Delete information for undoing your greeting.

Save your work and reload Lisp as soon as possible.

## 23. SIMPLE INTERACTIONS WITH THE CURSOR, A BITMAP, AND A WINDOW

---

The purpose of this chapter is to show you how to build a moderately tricky interactive interface with the various Medley display facilities. In particular how to move a large bitmap (larger than 16 x 16 pixels) around inside a window. To do this, you will change the `CURSORINFN` and `CURSOROUTFN` properties of the window. If you would also like to then set the bitmap in place in the window, you must reset the `BUTTONEVENTFN`. This chapter explains how to create the mobile bitmap.

### GETMOUSESTATE Example Function

---

One function that you will use to "trace the cursor" (have a bitmap follow the cursor around in a window) is `GETMOUSESTATE`. This function finds the current state of the mouse, and resets global system variables, such as `LASTMOUSEX` and `LASTMOUSEY`.

As an example of how this function works, create a window by typing

```
(SETQ EXAMPLE.WINDOW (CREATEW))
```

into the Executive Window, and sweeping out a window. Now, type in the function

```
(DEFINEQ (PRINTCOORDS (W)
 (PROMPTPRINT "(" LASTMOUSEX " , " LASTMOUSEY ")")
 (BLOCK)
 (GETMOUSESTATE)))
```

This function calls `GETMOUSESTATE` and then prints the new values of `LASTMOUSEX` and `LASTMOUSEY` in the prompt window. To use it, type

```
(WINDOWPROP EXAMPLE.WINDOW 'CURSORMOVEDFN 'PRINTCOORDS)
```

The window property `CURSORMOVEDFN`, used in this example, will evaluate the function `PRINTCOORDS` each time the cursor is moved when it is inside the window. The position coordinates of the mouse cursor will appear in the prompt window. (See Figure 23.1.)

Figure 23.1. Current Position Coordinates of Mouse Cursor in Prompt Window

### Advising GETMOUSESTATE

---

For the bitmap to follow the moving mouse cursor, the function `GETMOUSESTATE` is advised. When you advise a function, you can add new commands to the function without knowing how it is actually implemented. The syntax for advise is

```
(ADVISE fn when where what)
```

*fn* is the name of the function to be augmented. *when* and *where* are optional arguments. *when* specifies whether the change should be made before, after, or around the body of the function. The values expected are `BEFORE`, `AFTER`, or `AROUND`.

*what* specifies the additional code.

In the example, the additional code, *what*, moves the bitmap to the position of the mouse cursor. The function `GETMOUSESTATE` will be `ADVISED` when the mouse moves into the window. This will cause the bitmap to follow the mouse cursor. `ADVISE` will be undone when the mouse leaves the window or when a mouse button is pushed. The `ADVISEing` will be done and undone by changing the `CURSORINFN`, `CURSOROUTFN`, and `BUTTONEVENTFN` for the window.

## Changing the Cursor

---

One last part of the example, to give the impression that a bitmap is dragged around a window, the original cursor should disappear. Try typing:

```
(CURSOR (CURSORCREATE (BITMAPCREATE 1 1) 1 1])
```

into the Executive Window. This causes the original cursor to disappear. It reappears when you type

```
(CURSOR T)
```

When the cursor is invisible, and the bitmap moves as the cursor moves, the illusion is given that the bitmap is dragged around the window.

## Functions for Tracing the Cursor

---

To actually have a bitmap trace (follow) the cursor, the environment must be set up so that when the cursor enters the tracing region the trace is turned on, and when the cursor leaves the tracing region the trace is turned off. The function `Establish/Trace/Data` will do this. Type it in as it appears (include comments that will help you remember what the function does).

```
(DEFINEQ (Establish/Trace/Data
 [LAMBDA (wnd tracebitmap cursor/rightoffset cursor/heightoffset
 GCGAGP)

 (* * This function is called to establish the data to trace
 the desired bitmap. "wnd" is the window in which the tracing
 is to take place, "tracebitmap" is the tracing bitmap,
 "cursor/rightoffset" and "cursor/heightoffset" are integers
 which determine the hotspot of the tracing bitmap.
 As "cursor/heightoffset" and "cursor/rightoffset" increase
 the cursor hotspot moves up and to the right.
 If GCGAGP is non-NIL, GCGAG will be disabled.)

 (PROG NIL

 (if (OR (NULL wnd)
 (NULL tracebitmap))
 then (PLAYTUNE (LIST (CONS 1000 4000)))
 (RETURN))

 (if GCGAGP
 then (GCGAG))

 (* * Create a blank cursor.)

 (SETQ *BLANKCURSOR* (BITMAPCREATE 16 16))
 (SETQ *BLANKTRACECURSOR* (CURSORCREATE *BLANKCURSOR*))
```

```

(* * Set the CURSOR IN and OUT FNS for wnd to the
following:)

(WINDOWPROP wnd (QUOTE CURSORINFN)
 (FUNCTION SETUP/TRACE))
(WINDOWPROP wnd (QUOTE CURSOROUTFN)
 (FUNCTION UNTRACE/CURSOR))

(* * To allow the bitmap to be set down in the window by
pressing a mouse button, include this line.
Otherwise, it is not needed)

(WINDOWPROP wnd (QUOTE BUTTONEVENTFN)
 (FUNCTION PLACE/BITMAP/IN/WINDOW))
(WINDOWPROP wnd (QUOTE CURSOROUTFN))

(* * Set up Global Variables for the tracing operation)

(SETQ *TRACEBITMAP* tracebitmap
(SETQ *RIGHTTRACE/OFFSET*(OR cursor/rightoffset 0))
(SETQ *HEIGHTTRACE/OFFSET*(OR cxursor heightoffset 0))
(SETQ *OLDBITMAPPOSITION*(BITMAPCREATE (BITMAPWIDTH
 tracebitmap)
 (BITMAPHEIGHT
 tracebitmap)))
(SETQ *TRACEWINDOW* wnd]))

```

When the function Establish/Trace/Data is called, the functions SETUP/TRACE and UNTRACE/CURSOR will be installed as the values of the window's WINDOWPROPS, and will be used to turn the trace on and off. Those functions should be typed in, then:

```

(DEFINEQ (SETUP/TRACE
 [LAMBDA (wnd)

 (* * This function is wnd's CURSORINFN.
 It simply resets the last trace position and the current
 tracing region. It also readvises GETMOUSESTATE to perform
 the trace function after each call.)

 (if *TRACEBITMAP*
 then (SETQ *LAST-TRACE-XPOS* -2000)
 (SETQ *LAST-TRACE-YPOS* -2000)
 (SETQ *WNDREGION* (WINDOWPROP wnd (QUOTE REGION)))
 (WINDOWPROP wnd (QUOTE TRACING)
 T)

 (* * make the cursor disappear)

 (CURSOR *BLANKTRACECURSOR*)
 (ADVISE (QUOTE GETMOUSESTATE)
 (QUOTE AFTER)
 NIL
 (QUOTE (TRACE/CURSOR]))

 (DEFINEQ (UNTRACE/CURSOR
 [LAMBDA (wnd)

 (* * This function is wnd's CURSOROUTFN. The function first
 checks if the cursor is currently being traced; if so, it
 replaces the tracing bitmap with what is under it and then
 turns tracing off by unadvising GETMOUSESTATE and setting the
 TRACING window property of *TRACEWINDOW* to NIL.)

 (if (WINDOWPROP *TRACEWINDOW* (QUOTE TRACING))

```

```
then (BITBLT *OLDBITMAPPOSITION* 0 0 (SCREENBITMAP)
 (IPLUS (CAR *WNDREGION*) *LAST-TRACE-XPOS*)
 (IPLUS (CADR *WNDREGION*) *LAST-TRACE-YPOS*))
 (WINDOWPROP *TRACEWINDOW* (QUOTE TRACING)
 NIL))

(* * replace the original cursor shape)

(CURSOR T)

(* * unadvise GETMOUSESTATE)

(UNADVISE (QUOTE GETMOUSESTATE]))
```

The function SETUP/TRACE has a helper function that you must also type in. It is TRACE/CURSOR:

```
(DEFINEQ (TRACE/CURSOR
 [LAMBDA NIL

 (* * This function does the actual BITBLTing of the tracing
 bitmap. This function is called after a GETMOUSESTATE, while
 tracing.)

 (PROG ((xpos (IDIFFERENCE (LASTMOUSEX *TRACEWINDOW*
 RIGHTTRACE/OFFSET))

 (ypos (IDIFFERENCE (LASTMOUSEY *TRACEWINDOW*
 HEIGHTTRACE/OFFSET))

 (* * If there is an error in the function, press the right
 button to unadvise the function. This will keep the machine
 from locking up.)

 (if (LASTMOUSESTATE RIGHT)
 then (UNADVISE (QUOTE GETMOUSESTATE)))
 (if (AND (NEQ xpos *LAST-TRACE-XPOS*)
 (NEQ ypos *LAST-TRACE-YPOS*))
 then

 (* * Restore what was under the old position of the trace
 bitmap)
 (BITBLT *OLDBITMAPPOSITION* 0 0 (SCREENBITMAP)
 (IPLUS (CAR *WNDREGION*) *LAST-TRACE-XPOS*)
 (IPLUS (CADR *WNDREGION*) *LAST-TRACE-YPOS*))

 (* * Save what will be under the position of the new trace
 bitmap)
 (BITBLT (SCREENBITMAP)
 (IPLUS (CAR *WNDREGION*)
 xpos)
 (IPLUS (CADR *WNDREGION*)
 ypos) *OLDBITMAPPOSITION* 0 0)

 (* * BITBLT the trace bitmap onto the new position of the
 mouse)
 (BITBLT *TRACEBITMAP* 0 0 (SCREENBITMAP)
 (IPLUS (CAR *WNDREGION*)
 xpos)
 (IPLUS (CADR *WNDREGION*)
 ypos)
 NIL NIL (QUOTE INPUT)
 (QUOTE PAINT))

 (* * Save the current position as the last trace position.)
 (SETQ *LAST-TRACE-XPOS* xpos)
 (SETQ *LAST-TRACE-YPOS* ypos]))]
```

The helper function for UNTRACE/CURSOR, called UNDO/TRACE/DATA, must also be added to the environment:



```
(DEFINEQ (UNDO/TRACE/DATA
 [LAMBDA NIL
 (* * The purpose of this function is to turn tracing off
 and to free up the global variables used to trace the
 bitmap so that they can be garbage collected.)

 (* * Check if the cursor is currently being traced.
 It so, turn it off.)

 (UNTRACE/CURSOR)
 (WINDOWPROP *TRACEWINDOW* (QUOTE CURSORINFN)
 NIL)
 (WINDOWPROP *TRACEWINDOW* (QUOTE CURSOROUTFN)
 NIL)
 (SETQ *TRACEBITMAP* NIL)
 (SETQ *RIGHTTRACE/OFFSET* NIL)
 (SETQ *HEIGHTTRACE/OFFSET* NIL)
 (SETQ *OLDBITMAPPOSITION* NIL)
 (SETQ *TRACEWINDOW* NIL)

 (* * Turn GCGAG on)

 (GCGAG T]))
```

Finally, if you included the `WINDOWPROP` to allow the user to place the bitmap in the window by pressing a mouse button, you must also type this function:

```
(DEFINEQ (PLACE/BITMAP/IN/WINDOW
 [LAMBDA (wnd)

 (UNADVISE (GETMOUSESTATE))
 (BITBLT *TRACEBITMAP* 0 0 (SCREENBITMAP)
 (IPLUS (CAR *WNDREGION*)
 xpos)
 (IPLUS (CADR *WNDREGION*)
 ypos)
 NIL NIL (QUOTE INPUT)
 (QUOTE PAINT])
```

That's all the functions!

---

## Running the Functions

To run the functions you just typed in, first set a variable to a window by typing something like

```
(SETQ EXAMPLE.WINDOW (CREATEW))
```

into the Executive Window, and sweeping out a new window. Now, set a variable to a bitmap, by typing, perhaps,

```
(SETQ EXAMPLE.BTM (EDITBM))
```

Type

```
(Establish/Trace/Data EXAMPLE.WINDOW EXAMPLE.BTM)
```

When you move the cursor into the window, the cursor will drag the bitmap.

(If you want to be able to make menu selections while tracing the cursor, make sure that the hotspot of the cursor is set to the extreme right of the bitmap. Otherwise, the menu will be destroyed by the BITBLTs of the trace functions.)

To stop tracing, do one of the following:

- Move the mouse cursor out of the window
- Press the right mouse button
- Call the function `UNTRACE/CURSOR`

## 24. GLOSSARY OF GLOBAL SYSTEM VARIABLES

---

As you can tell by now, there are many system variables in Medley that are useful to know. The following sections gather many of the important variables together into groups relating to directory searching, system flags, history lists, system menus, windows, and, of course, the catchall miscellaneous category.

### Directories

---

#### DISPLAYFONTDIRECTORIES

Its value is a list of directories to search for the bitmap files for display fonts. Usually, it contains the FONT directory where you copies the bitmap files (see Chapter 16), and the current connected directory. The current connected directory is specified by the atom NIL. Here is an example value of DISPLAYFONTDIRECTORIES.

A screenshot of a text editor window. The title bar at the top reads "\* SEdit DISPLAYFONTDIRECTORIES Package: INTERLISP". The text area contains three lines of code: ("{DSK}<usr<local<lde<Fonts>display>presentation>" followed by "{DSK}<usr<local<lde<Fonts>display>publishing>" followed by "{DSK}<usr<local<lde<Fonts>display>printwheel>").

```
* SEdit DISPLAYFONTDIRECTORIES Package: INTERLISP
("{DSK}<usr<local<lde<Fonts>display>presentation>"
 "{DSK}<usr<local<lde<Fonts>display>publishing>"
 "{DSK}<usr<local<lde<Fonts>display>printwheel>")
```

Figure 24.1. Value for the Atom DISPLAYFONTDIRECTORIES

#### INTERPRESSFONTDIRECTORIES

Is set to a list of directories to search for the font width files for InterPress fonts.

#### DIRECTORIES

This variable is bound to a list of the directories you will be using (see Figure 24-2). The system uses this variable when it is trying to find a file to load. It checks each directory in the list, until the file is found. NIL in list means to check the current connected directory.

#### LISPUSERSDIRECTORIES

Its value is a list of directories to search for library package files.

## Flags

---

### DWIMIFYCOMPFLG

This flag, if set to T, will cause all expressions to be completely dwinified before the expression is compiled (see Chapter 9). In this state, when the system does not recognize a function of keyword, it will compare the word to a system maintained list to determine whether the word is a macro, CLISP word, or misspelled user-defined variable.

An example of swinifying before compilation is to convert an IF call to a COND. before they are compiled. Undwimified expressions can cause inaccurate compilation. This flag is set by the system to NIL. Normally, you want this set to T. For more information on DWIM, refer to the *IRM*.

### SYSPRETTYFLAG

When set to T, all lists returned to the executive window are pretty printed. This flag is originally set by the system to NIL.

### CLISPIFTRANFLG

When set to T, keeps the IF expression, rather than the COND translation in your code.

### PRETTYTABFLG

When set to T, the pretty printer puts out a tab character rather than several spaces to try to make code align. If NIL, it uses space characters instead.

### FONTCHANGEFLG

If NIL, then when pretty printing no font changes will happen (e.g., a smaller font for comments, bold for clip words, and so forth). The default is the atom ALL, so different fonts are used where appropriate.

### AUTOBACKTRACEFLG

There are many possible values for this variable. They affect when the back trace window appears with the break window, and how much detail is included in it. The values of this variable include:

- NIL, its initial value. The back trace window is not brought up when an error is generated, until you open it yourself.
- T, which means that the back trace BT window is opened for error breaks
- BT! brings up a back trace window with more detail, BT!, window for error breaks
- ALWAS brings up a backtrace BT window for both error breaks, and breaks caused by calling the function BREAK
- ALWAYS! brings up a backtrace window with more detail, BT!, for both error breaks and breaks caused by calling the function

**NOSPELLFLG**

Is initially bound to `NIL`, so that `DWIM` tries to correct all spelling errors, whether they are in a form you just typed in or within a function being run. If the variable is `T`, then no spelling correction is performed. This variable is automatically reset to `T` when you are compiling a file. If it has some other non-`NIL` value, then spelling correction is only performed on type-in.

---

## History Lists

---

**LISPMXHISTORY**

Originally set to the list `(NIL 0 30 100)`, with the following argument interpretation. The `NIL` is the list (implemented as a circular queue) to which the top level commands append. `0` is the current prompt number. `30` is the maximum length of the history list. `100` is the highest number used as a prompt. This is a system maintained list used by the programmers assistant commands `REDO`, `UNDO`, `FIX`, and `??` use to retrieve past function calls.

To delete the history list, reset the variable `LISPMXHISTORY` to its original value of `(NIL 0 30 100)`.

Setting this variable to `NIL` disables all the programmers assistant features.

**EDITHISTORY**

This is also set to `(NIL 0 30 100)`, and has the same description as `LISPMXHISTORY`. This list allows you to `UNDO` edits. You reset this the same way as `LISPMXHISTORY`.

---

## System Menus

---

System menus are all bound to global variables and are easy to modify. If the menu name is set to the `NIL` value, the menu will be recreated using an items list bound to a global variable.

To change a system menu, edit the items list bound to the appropriate global variable (system menus use this items list with the default `WHENSELECTEDFN`), then set the value of the name to `NIL`. The next time you need the menu, it will be created from the items list you just edited. The names of system menus and the items lists follow.

**BackgroundMenu**

This is the variable bound to the menu this displays when you press the right button in the grey background area of the screen.

**BackgroundMenuCommands**

This list is used for the list of `ITEMS` for the background menu when it is created.

**WindowMenu**

This is the variable bound to the default window menu displayed when the right mouse button is pressed inside of a window.

### WindowMenuCommands

This is the list of ITEMS for the WindowMenu.

### BreakMenu

The menu displayed when the middle mouse button is pressed in a break window.

### BreakMenuCommands

The list of ITEM for the BreakMenu.

## Windows

---

### PROMPTWINDOW

Global name of the prompt window.

### T

Although the value T has several meanings (such as universal TRUE), it also stands for the standard output stream. As this is usually the executive window, it may be used as the name for the TTY Window at the top level. Mouse processes have their own TTY Windows. A reference to the window T in a mouse driven function (e.g., a WHENSELECTEFN, Chapter 12) will open a TTY Window for Mouse.

## Miscellaneous

---

### CLEANUPOPTION

This is a list of options that you set to automate clean-up after a work session. Example options are listing files, or recompilation. You will want to keep this set to NIL until you become comfortable with the machine.

### FILELST

The list of all the files you loaded.

### SYSFILES

The list of all the files you loaded for the SYSOUT file.

### INITIALS

An atom you can bind to your name. If bound, the editor will add your name, in addition to the date, in the editor comment at the beginning of each function.

### FIRSTNAME

If this variable is set, the system will use it to greet you personally when you log on to your machine.

**INITIALSLST**

A list of elements of the form (USERNAME . INITIALS) or (USERNAME FIRSTNAME INITIALS). This list is used by the function GREET to set your INITIALS, and your FIRSTNAME when you log in.

**#CAREFULCOLUMNS**

An integer. PRETTYPRINT estimates the number of characters in an atom, instead of computing it, for efficiency. Unfortunately, for very long atom names, errors can occur. #CAREFULCOLUMNS is the number of columns from the right within which PRETTYPRINT should compute the number of characters in each atom, to prevent these errors. Initially this is set to zero. PRETTYPRINT never computes the number of characters in an atom. If you set it to 20 or 30, when PRETTYPRINT comes within 20 or 30 columns of the right of the window, it will begin computing exactly how many characters are in each atom. This will prevent errors.

**DWIMWAIT**

Bound to the number of seconds DWIM should wait before it uses the default response, FIXSPELLDEFAULT, to answer its question.

**FIXSPELLDEFAULT**

Bound to either Y or N. Its value is used as the default answer to questions asked by DWIM that you don't answer in DWIMWAIT seconds. It is initially bound to Y, but is rebound to N when DWIMIFYing.

**\TimeZoneComp**

This is the global variable set to the absolute value of the time offset from Greenwich. For EST, \TimeZoneComp should be set to 5.

## 25. OTHER USEFUL REFERENCES

---

Here are some references to works that will be useful to you in addition to this primer. Some of these you have already been referred to, such as:

- The Interlisp-D Reference Manual (IRM)
- The Library Packages Manual
- The User's Guide to SKETCH

In addition, you can learn more about Lisp with the books:

- **Essential LISP** by John Anderson, Albert Corbett, and Brian Reiser. This book was published in 1986 by John Wiley and Sons, NY.
- **Essential LISP** by John Anderson, Albert Corbett, and Brian Reiser. This book was published in 1986 by Addison Wesley Publishing Company, Reading, MA. It was informed by research on how beginners learn LISP.
- **The Little Lisper** by Daniel P. Friedman and Matthias Felleisen. The second edition of this book was published in 1986 by SRA Associates, Chicago. This book is a deceptively simple introduction to recursive programming and the flexible data structures provided by LISP.
- **LISP** by Patrick Winston and Berthold Horn. The second edition of this book was published in 1985 by the Addison Wesley Publishing Company, Reading, MA.
- **LISP: A Gentle Introduction to Symbolic Computation** by David S. Touretzky. This book was published in 1984 by the Harper and Row Publishing Company, NY.

Finally, there are three articles about the Interlisp Programming environment:

- Power Tools For Programmers by BeauSheil. It appeared in *Datamation* in February, 1983, Pages 131 - 144.
- The Interlisp Programming Environment by Warren Teitelman and Larry Masinter. It appeared in April, 1981, in *IEEE Computer*, Volume 14:1, Pages 25 - 34.
- Programming In an Interactive Environment, the LISP Experience by Erik Sandewall. It appeared in March, 1978, in the *ACM Computing Surveys*, Volume 10:1, pages 35 - 71.

Each of these articles was reprinted in the book **Interactive Programming Environments** by David R. Barstow, Howard E. Shrobe, and Erik Sandewall. This book was published in 1984 by McGraw Hill, NY. The first article can be found on pages 19 - 30, the second on pages 83 - 96, and the third on pages 31 - 80.



## TABLE OF CONTENTS

~)19 ~'~

|                                            |     |
|--------------------------------------------|-----|
| 1. A Brief Glossary                        | 1.1 |
| 2. The Mouse and the Keyboard              | 2.1 |
| 2.1. The Mouse                             | 2.1 |
| 2.1.1. 2and3ButtonMice                     | 2.1 |
| 2.2. The Keyboard                          | 2.2 |
| 2.2.1. The 1186 Keyboard                   | 2.2 |
| 2.2.2. The 1108 Keyboard                   | 2.2 |
| 3. Turning On Your Lisp Machine            | 3.1 |
| 3.1. Turning on the 1186                   | 3.1 |
| 3.2. Turning on the 1108                   | 3.2 |
| 3.3. Loading Interlis-D from the Hard Disk | 3.3 |
| 3.4. After Booting Lisp                    | 3.5 |
| 3.5. Restarting Lisp After Logging Out     | 3.5 |
| 4. If You Have a Fileserver                | 4.1 |
| 4.1. Turning on your 1108                  | 4.1 |
| 4.2. Turning on your 1186                  | 4.1 |
| 4.3. Location of Files                     | 4.2 |
| 4.4. The Timeserver                        | 4.2 |
| 5. Logging Out And Turning the Machine Off | 5.1 |
| 5.1. Logging Out                           | 5.1 |
| 5.2. Turning The Machine Off               | 5.2 |
| 6. Typing Shortcuts                        | 6.1 |
| 6.1. If you make a Mistake                 | 6.3 |
| 7. Using Menus                             | 7.1 |
| 7.1. Making a Selection from a Menu        | 7.2 |
| 7.2. Explanations of Menu Items            | 7.2 |
| 7.3. Submenus                              | 7.3 |
| 8. How to use Files                        | 8.1 |
| 8.1. Types of Files                        | 8.1 |

TABLE OF CONTENTS To',  
1

----

.-----.-.-.-

## TABLE OF CONTENTS

|                                             |      |
|---------------------------------------------|------|
| 8.2. Directories                            | 8.1  |
| 8.3. Directory Options                      | 8.2  |
| 8.4. Subfile Directories                    | 8.3  |
| 8.5. To See What Files Are Loaded           | 8.3  |
| 8.6. Simple Commands for Manipulating Files | 8.3  |
| \                                           |      |
| 8.7. to a                                   | 8.4  |
| -'.~--y Connecting Directory                |      |
| \ s 8.8. File Vetting Numbers               |      |
| 8.4                                         |      |
| 9. FileBrowser                              | 9.1  |
| 9.1. Calling the FileBrowser                | 9.1  |
| 9.2. FileBrowser Commands                   | 9.3  |
| 10. ffose Wondertul Windows!                | 10.1 |
| 10.1. Windows provided by Interlis-D        | 10.1 |

|                   |                                                                        |      |
|-------------------|------------------------------------------------------------------------|------|
| 10.2.             | _ Creating a _ window _                                                | 10.2 |
| 10.3.             | _ The Right _ Button DefaultWindow _ Menu _                            | 10.2 |
| 10.4.             | _ An _ explanation of each _ menu _ item _                             | 10.3 |
| 10.5.             | _ krollable Windows _                                                  | 10.3 |
| 10.6.             | Other Window Functions                                                 | 10.5 |
| 10.6.1.           | PROMPTPRINT                                                            | 10.5 |
| 10.6.2.           | _ WHICHW _                                                             | 10.6 |
| 11.               | Editing and Saving                                                     | 11.1 |
| 11.1.             | _ Defining _ Functions _                                               | 11.1 |
| 11.2.             | Simple Editing in the Interlis~D Executive Window                      | 11.2 |
| YA8~ OF cottnritt |                                                                        |      |
| iA.3.             | _ Wøys to Stop Exøcution from thø Køyboard, called _ 1ørøøhlng LIzp5 _ | 14.3 |
| t4.4.             | _ Programming _ Brøaks and Døbugging Codø _                            | 14.4 |
| 14.5.             | Break Monu                                                             | 14.4 |
| 14.6.             | _ Returning to Top Lovøl _                                             | '4.5 |
| 15.               | _ On-Line _ Help _ with _ Interlisp-D: _ HELPSYS and _ DINFO _ ~ _     | 15.1 |
| 15.1.             | _ HelpSys _                                                            | 15.1 |
| 15.2.             | DInfo                                                                  | 15.1 |
| 16.               | Floppy Disks /                                                         | 16.1 |
| 16.1.             | _ Buying Floppy Disks _                                                | 16.1 |
| 16.2.             | _ Basic Floppy Disk Information _                                      | 16.1 |
| 16.3.             | _ Care of Floppies _                                                   | 16.2 |
| 16.4.             | _ Write Enabling and Write Protecting _ Floppies _                     | 16.3 |
| 16.4.1.           | _ Write Enabling an _ 1108's _ Floppy _ Disk _                         | 16.3 |
| 16.4.2.           | _ Write Protecting an _ 1186's Floppy Disk _                           | 16.3 |
| 16.5.             | _ Inserting _ Floppies _ intothe _ Floppy Drive _                      | 16.3 |
| 16.6.             | _ Functions for Floppy Disks _                                         | 16.4 |
| 16.6.1.           | _ Formatting Floppies _                                                | 16.4 |
| 16.6.2.           | _ Available Space on a Floppy Disk _                                   | 16.4 |
| 16.6.3.           | _ The Name ofa Floppy Disk _                                           | 16.4 |
| 16.6.4.           | _ FLOPPY.MODE _                                                        | 16.5 |
| 17.               | Duplicating Floppy Disks                                               | 17.1 |
| 17.1.             | _ Supplies _                                                           | 17.1 |
| 17.2.             | _ Preparabon _                                                         | 17.1 |
| 17.2.1.           | _ Handling _ Floppy _ Disks _                                          | 17.1 |
| 17.2.2.           | _ Setup _                                                              | 17.1 |
| 17.3.             | _ Copying _ Floppy Disks _                                             | 17.2 |
| 18.               | Sysout Files                                                           | 18.1 |
| 18.1.             | _ Loading SYSOUT Filri _                                               | 18.1 |
| 18.1.1.           | _ Loading a _ SYSOUTfile on the _ 1108 _                               | 18.1 |
| 18.1.2.           | _ Loading a SYSOuTfileonthe _ 1186 _                                   | 18.2 |
| 18.2.             | _ Making _ Your Own SYSOUT File _                                      | 18.3 |
| 19.               | Using the Epson FX80 Printer ~                                         | 19.1 |
| 19.1.             | _ Initializing the RS232 Port _                                        | 19.1 |
| 19.2.             | _ Power upthe Printer _                                                | 19.1 |
| 19.3.             | _ to Align Top of Page _                                               | 19.1 |

YA8~ OF CONTENTS TOC.3

TABLE OF CONTEND yl

|         |                                           |      |
|---------|-------------------------------------------|------|
| 19.4.   | _ Fundions To Print Filri and _ Bitmapf _ | 19.2 |
| 19.4.1. | _ RS232.Print _                           | 19.2 |
| 19.4.2. | _ FXWSTREAM _                             | 19.2 |
| 19.4.3. | _ Printing a Portion of the Screen _      | 19.3 |
| 20.     | R5232 File Transfer With a VAX            | 20.1 |
| 20.1.   | _ Prerequisites _                         | 20.1 |
| 20.2.   | _ Using Chat to Transfer Filri _          | 20.1 |
| 21.     | Ethernet File Transfer                    | 21.1 |
| 21.1.   | _ Prerequisites _                         | 21.1 |

21.2. File Transfer 21.1

22. WhatToDoIf...

22.1

23. The Text Editor, TEdit 23.1

23.1. Using TEdit 23.1

23.2. Managing the edit Window 23.2

23.3. Selecting Text 23.3

23.4. Deleting, Copying, and Moving Text with edit 23.4

23.4.1. Deleting Text From a File 23.4

23.4.2. Copying Text 23.4

23.4.3. Moving Text 23.5

23.5. TEdit Menus 23.6

23.5.1. Finding and Substituting Text with edit 23.7

23.5.1.1. Finding Text 23.7

23.5.1.2. Substituting Text 23.8

23.5.2. Text Formatting 23.10

23.5.2.1. Choosing Fonts 23.10

23.5.2.2. Paragraph Formatting 23.11

23.5.3. Adding Bitmaps and Sketches to your TEdit File 23.13

23.5.3.1. Adding a Bitmap to your TEdit file 23.13

23.5.3.2. Adding a Sketch to your TEdit file 23.14

23.5.4. Getting and Including Filenames 23.14

23.5.4.1. Get 23.14

23.5.4.2. Include 23.14

23.5.5. Saving and Printing Files 23.15

24. Records May Be Your Favorite Data Structure! 24.1

24.1. Interlisp Record Structures 24.1

Table of Contents

TABLE OF CONTENTS

24.2. Example 24.3

24.3. Appendixes 24.4

25. Local Variables - Using LET and PROG 25.1

25.1. LET 25.1

25.2. PROG 25.3

25.3. Porillon's vottus Sequential Varioblo Binding 25.6

25.3.1. L 25.6

25.3.2. PROG 25.7

26. Iterative statements 26.1

26.1. General Structure and Use 26.1

26.2. Local Variables 26.2

26.3. Iteration On Lists 26.3

26.4. Parallel Iteration 26.4

26.5. Conditional Iteration 26.5

26.6. More Iteration 26.6

27. Window and Regions 27.1

27.1. Windows 27.1

27.1.1. CREATEW 27.1

27.1.2. WINDOWPROP 27.2

27.1.3. Getting windows to do things 27.3

27.1.3.1. BUFLONEVENTFN 27.4

27.1.4. Looking at a window's properties 27.5

27.2. Regions 27.5

28. What Are Menus? 28.1

28.1. Displaying Menus 28.1

28.2. Getting Menus to DO Stuff 28.2

28.2.1. The WHENHELD and WHENSELECTED fields of a

menu 28.4

28.3. \_ Looking \_ at a \_ menu's fields \_ 28.5

29. Bitmaps 29.1

30. Displaystreams 30.1

30.1. \_ Drawing \_ on a \_ Displaystream \_ 30.1

30.1.1. \_ DliWUNE \_ 30.1

30.1.2. \_ DliWTO \_ 30.2

30.1.3. \_ DliWaRCLE \_ 30.3

TABS OF CONTENTff TOC.5

l

TABS OF CON~Nfl

30.1.3.1. \_ FILLGRCLE \_ 30.3

30.2. \_ Locating and \_ Changing \_ Your Position \_ in \_ a \_ Displaystream \_ 30.4

30.2.1. \_ DSPXP0SifION \_ 30.5

30.2.2. \_ DSPYPOSIBON \_ 30.5

30.2.3. \_ MOVETO \_ 30.5

31. Fonts 31.1

31.1. \_ WhatmakesupaFONn \_ 31.1

31.2. \_ Fontdescriptors, and \_ FONTCREATE \_ 31.2

31.3. \_ Display Fonts-Theirfiles, and how to find them \_ 31.3

31.4. \_ Interpress \_ Fonts- Their files, and \_ how to find them \_ 31.4

31.5. \_ Functions for Using Fonts \_ 31.4

31.5.1. \_ FOHTPROP - \_ Looking at Font Properties \_ 31.4

31.5.2. \_ SffllNGWIDTH \_ 31.5

31.5.3. \_ DSPFONT- Changing the Font in \_ One Window \_ 31.6

31.5.4. \_ GIo~llyChanging Fonts \_ 31.7

31.5.5. \_ Pettonalizing \_ Your Font Profile \_ 31.7

32. The Inspetror 32.1

32.1. \_ Calling the Inspector \_ 32.1

32.2. \_ Using \_ the \_ Inspector \_ 32.2

32.3. \_ Inspector \_ Example \_ 32.2

33. Masterscope 33.1

33.1. \_ The SHOW DATA command and GRAPHER \_ 33.2

33.2. Databasefns: Automatic Conrtruction and Upkeep of a Mastettcope  
Data~se \_ 33.3

34. Where Does All the Time Go? SPY 34.1

34.1. \_ How to use Spy with the SAY Window \_ 34.1

34.2. \_ How to use \_ SPY from the \_ Lisp Top Level \_ 34.2

34.3. \_ Interpreting \_ SPY's Results \_ 34.2

35. SKETCH 35.1

35.1. \_ Starting \_ Sketch \_ 35.1

35.2. \_ Selecting \_ Sketch elements \_ 35.1

35.3. \_ Drawing \_ with \_ Sketch \_ 35.2

35.3.1. \_ Simplø Shapes: \_ Circles, Ellipsriø and \_ Boxes \_ 35.3

35.3.1.1. \_ Drawing \_ Circlri \_ 35.3

35.3.1.1 \_ Ellllpsri \_ 35.3

TA.G TAILE0fC0NFENrt '/'

----- Next Message -----

Date: 19 Dec 91 14:18 PST

From: sybalsky:PARC:Xerox

To: sybalsky

Message-ID: <<91Dec19.141853pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11640>;

Thu, 19 Dec 1991 14:19:05 PST

Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 14:18:53 -0800

From: John Sybalsky <sybalsky.PARC@xerox.com>

-----RFC822 headers----->

## TABLE OF CONTENTS

|                                                                                       |      |
|---------------------------------------------------------------------------------------|------|
| 42. _ Simple _ Interactions _ with _ the _ Cursor, _ a _ Bitmap, _ and _ & _ Window _ | 42.1 |
| 42.1. _ An _ Example Function _ Using _ GETHOUSESTATE _                               | 42.1 |
| 42.2. _ Advising _ GETMOUSESTATE _                                                    | 42.2 |
| 42.3. _ Changing the Cursor _                                                         | 42.2 |
| 42.4. _ Functions for _ "Tracing the cursor" _                                        | 42.3 |
| 42.5. _ Running the Functions _                                                       | 42.6 |
| 43. Glossary of Global System Variables                                               | 43.1 |
| 43.1. _ Directories _                                                                 | 43.1 |
| 43.2. _ Flags _                                                                       | 43.2 |
| 43.3. _ History Lists _                                                               | 43.3 |
| 43.4. _ System Menus _                                                                | 43.3 |
| 43.5. _ Windows _                                                                     | 43.4 |
| 43.6. _ Miscellaneous _                                                               | 43.4 |
| 44. Other References that will be Useful to You                                       | 44.1 |

## TABLE OF CONTENTS

## PREFACE

it was dawn and the locd told him it was down the road a piece,  
left the first fishing bridge in the country right at the apple  
tree stump, and onto the dirt road just before the hill. At  
midnight he knew he was lost.  
-Anonymous

Welcome to the Interlisp-D programming environment! The  
Interlisp-D environment truly must be one of the most  
sophisticated and powerful tools in use by human beings.  
Overall, it is flexible, well thought out, and full of pleasant  
surprises: "Wow, here are exactly the set of functions I thought  
I'd need to write." Unfortunately, along with the power comes  
mind-numbing complexity. The Interlisp Reference Manual  
describes the functions and some of the tools available in the  
Interlisp-D environment. To do this takes three large volumes.  
Other volumes are needed to document the library packages and  
other newly written tools. Needless to say, it is very difficult to  
learn such a huge amount of material when there is no way to  
determine where to start!

We developed this primer to provide a starting point for new  
Interlisp-D users, to enhance your excitement and challenge you  
with the potential before you. We assume you know a little  
about LISP, most likely received from taking a survey course in  
Artificial Intelligence (AI), and have seen a demonstration of  
how Interlisp-D runs on your 1186 or 1108. We further assume  
that your machine is not on a network system with a file server -  
though this is addressed, and that you will be working from  
floppy disks and the hard disk that is part of the machine. If this  
describes your situation, you are ready to sit down in front of  
your machine and follow the step-by-step examples provided in  
this primer.

The primer is broken into many small chapters, and these  
chapters are organized into five parts. You may want to read

Parts 1 through 3 straight through, since they describe the basics of using the machine. Each chapter in Sections 4 and 5, however, can be used to learn a specific skill whenever you are ready to for it

Part one, "Introduction", includes Chapters 1 and 2. Part two, "Getting Into/Out of Interlisp", includes Chapters 3 through 5. Part three, "The Interlisp~D language and Programming Environment", includes Chapters 6 through 15. These chapters discuss primary elements in Interlisp~D, and orient you in relation to those elements. Part four, "Important Other Things to Know to Work Successfully", includes Chapters 16 through 31. Part five, "More Language and Environment and Packages", includes Chapters 32 through 44.

PREFACE v

----- Next Message -----

Date: 19 Dec 91 14:20 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.142054pst.43009@origami.parc.xerox.com>.:>

<-----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11636>; Thu, 19 Dec 1991 14:21:05 PST

Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 14:20:54 -0800

From: John Sybalsky <sybalsky.PARC@xerox.com>

<-----RFC822 headers----->

## PREFACE

Through out we make reference to the Interlisp~D Reference Manual by section and page number. The material in the primer is just an introduction. When you need more depth use the detailed treatment provided in the manual.

While only you can plot your ultimate destination, you will find this primer indispensable for clearly defining and guiding you to the first landmarks on your way.

**Acknowledgements** The early inspiration and model for this primer came from the Intelligent Tutoring Systems group and the Learning Research and Development Center at the University of Pittsburgh. We gratefully acknowledge their pioneering contribution to more effective artificial intelligence.

This primer was developed by Computer Possibilities, a company committed to making AI technology available. Primary development and writing was done by Cynthia Cosic, with technical writing support provided by Sam Zordich.

At Xerox Artificial Intelligence Systems, John Vittal managed and directed the project. Substantial assistance was provided by many members of the AIS staff who provided both editorial and systems support.

PREFACE ~ 01

----- Next Message -----

Date: 19 Dec 91 14:33 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.143340pst.43009@origami.parc.xerox.com>.?::>

<-----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11653>;  
 Thu, 19 Dec 1991 14:33:46 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 14:33:40 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 -----RFC822 headers----->

## 1. ABRIEFGLOSSARY

The following definitions will acquaint you with general terms used throughout this primer. You will probably want to read through them now, and use this chapter as a reference while you read through the rest of the primer.

**advising** An Interlis~D facility for specifying function modifications without necessarily knowing how a particular function works or even what it does. Even system functions can be changed with advising.

**argument** An argument is a piece of information given to an Interlis~D function so that it can execute successfully. When a function is explained in the primer, the arguments that it requires will also be given. Arguments are also called Parametert.

**atom** The smallest rtrvcture in Lisp; like a variable in other programming languages, but can also have a property list and a function definition.

**Background Menu** The menu that appears when the mouse is not in any window and the right mouse button is pressed. A typical background menu is shown in Figure I.1.

Loops Icon  
 FileB'owser

**Figure 1.1.** The Menu that appeort when the mouse is not in any window, and the right mouse button is pressed. Your background menu may have some different items in it

**binding** The value of a variable. It could be either a local or a global variable. See unbound.

**bitmap** A rectangular array of '0 pixels, '0 each of which is on or off representing one point in the bitmap image.

**BREAK** An Interlisp function that causes a function to stop executing, open a Break window, and allow the user to find out what is happening while the function is halted.

**Break Window** A window that opens when an error is encountered while running your program (i.e., when your program has broken). There are tools to help you debug your program from this window. This is explained further in Chapter 14, Page 14.1.  
**browse** To examine a data strvcture by use of a display that allows the user to "move" around within the data rtructure.  
**button**

## A BRIEF GLOSSARY 1.1

1

## A BRIEF GLOSSARY

(1) (n.) A key on a mouse.

(2) (v.t.) To depress one of the mouse keys when making a selection.  
 CAR A function that returns the head or first element of a list. See  
 CDR.

caret The small blinking arrowhead that marks where text will appear  
 when it is typed in from the keyboard. An example of the caret  
 in the Interlisp-D Executive Window is shown in Figure 1.2.

NIL

B6+(PLUS 3A

Figure 1J. The caret is to the right of the number 3. When a character is typed  
 at the keyboard, it will appear at the caret

CDR A function that returns the tail (that is, everything but the first  
 element) of a list. See CAR.

CLISP A mechanism for augmenting the standard Lisp syntax. One such  
 augmentation included in Interlisp is the iterative statement.  
 See Section 13.1.

cr Please press your carriage return key.  
 datatype

(1) The kind of a datum. In Interlisp, there are many System-defined  
 datatypes e.g. Floating Point, Integer, Atom, etc.

(2) A datatype can also be user-defined. In this case it is like a record  
 made up from system types and other user-defined datatypes.  
 DWIM D~what I mean. Many errors made by Interlisp users could be  
 corrected without any information about the purpose of the  
 program or expression in question (e.g. misspellings, certain  
 kinds of parenthesis error). The DWIM facility is called  
 automatically whenever an error occurs in the evaluation of an  
 Interlisp expression. If DWIM is able to make a correction, the  
 computation continues as though no error had occurred;  
 otherwise, the standard error mechanism is invoked.

error Occasionally, while a program is running, an error may occur  
 which will stop the computation. Interlisp provides extensive  
 facilities for detecting and handling error conditions, to enable  
 the testing, debugging, and revising of imperfect programs.  
 evaluate or EVAL Means to find the value of a form. For example, if the variable X  
 is bound to 5, we get 5 by evaluating X. Evaluation of a Interlisp  
 function involves evaluating the arguments and then applying  
 the function.

file package A set of functions and conventions that facilitate the  
 bookkeeping involved with working in a large system consisting  
 of many source code files and their compiled counterparts.

Essentially, the file package keeps track of where things are and

1,

A0R1EFGLOSS~y

1



## A BRIEF GLOSSARY

When things have changed. In 4150 kbps track of which files have been modified and need to be updated and recompiled.  
 form Another way of saying expression. An Interlisp-D expression can be evaluated.

function A Lisp function is a piece of Lisp code that executes and returns a value.

history The programmer's assistant is built around a memory structure called the history list. The history functions (e.g. FIX, UNDO, REDO) are part of this assistant. These operations allow you to conveniently rework previously specified operations.

History List As you type on the screen, you will notice a number followed by a prompt arrow. Each number, and the information on that line, is sequentially stored as the History List Using the History List, you can easily reexecute lines typed earlier in a worksession. See Chapter 6.

icon A pictorial representation, usually of a shrunken window.

Interlisp-D Executive Window This is your main window, where you will run functions and develop your programs. See Figure 1.3. This is the window that the caret is in when you turn on your machine and load Interlisp-D.

NIL

8~#iPRO\*PTPRINT "HELLO" A

Four terminal window

inspector An interactive display program for examining and changing the parts of a data structure. Interlisp-D has inspectors for lists and other data types.

iterative statement (also called i.s.) A statement in Interlisp that repetitively executes a body of code. (E.g. (forx from to do (PRINT x)) is an i.s.)

iterative variable (also called i.v.) Usually, an iterative statement is controlled by the value that the i.v. takes on. In the iterative statement example above,  
 x

is the iterative variable because its value is being changed by each cycle through the loop. All iterative variables are local to the iterative statement where they are defined.

LISP Family of languages invented for "list processing." These languages have in common a set of basic primitives for creating and manipulating symbol structures. Interlisp-D is an implementation of the LISP language together with an environment (set of tools) for programming, and a set of packages that extend the functionality of the system.

list A collection of atoms and lists; a list is denoted by surrounding its contents with a pair of parentheses.

A BRIEF GLOSSARY II

1

A BRIEF GLOSSARY

**Loading LJSP** This is the process of bringing Interlis~D from floppy disks, hard disks, or some other secondary storage into your main, or working, memory. You will need to load (i.e., install, and boot) Interlis~D if you have not logged off the machine at the end of a session. The process of loading Interlis~D is explained in Chapter3.

**Maintenance Panel Codes** Should you have a problem with your equipment, these codes will indicate the status of your processor. On the 1108, these are the red LED numbett under the floppy drive door. There is a cover over these numbers. Pull down the cover located immediately under the floppy door button. The code numbers are defined for the 1108 in the 1108 Useri Guide, in the MP Codeschapter.

If there is a problem with the 1186, the mouse curtør will change from its normal arrow to the code number that describes the problem. The code numbert are defined for the 1186 in the 1166 User's Guide in the Curtør Codes subsection of the Diagnostics Chapter.

**Marterscope** A program analysis tool. When told to analyze a program, Masterscope creates a data base of information about the program. In particular, Marterscope knows which functions call other functions and which functions use which variables. Masterscope can then answer questions about the program and display the information with a browser.

**menu** A way of graphically presenting the user with a set of options. There are two kinds of menus: p0~up menus are created when needed and disappear after an item has been selected; permanent menus remain on the screen after use.

**mouse** The Mouse is the box to the right of your keyboard. It controls the movement of the cursor on your screen. As you become familiar with the mouse, you will find it much quicker to use the mouse than the keyboard. See Figure 1.4. (Note: Some mice have three buttons; the button in the center is known as the middle mouse button. If your mouse has only two buttons, you can simulate a middle button by pressing the left and right buttons simultaneously.).

**Fw- 1.& Mousø**

**Mouse Curtør** The small arrow on the screen that points to the northwest. See Figure 1.5.

**F~m I.L Mousø c~~~**  
**Mousø Curtør Icons**

**I.A A lIEF GLos~Y**  
**l**

## A IRIEF GLOSSARY

**I Wait** Tho processor is busy.

The processor is saving a anpashot or your cureent system session. This is usually donø when tbc procffssor hos ~n idle for a while.

The "Mouse Confirm Cursor". It appeatt when you have to confirm that the choice you just made was correct. If it was, press the left button. If the choice was not tight, press the right button to abort.

F='\*x This means "sweep out" the shape of the window. To do this, move the mouse to a position where you want a corner. Press the left mouse button, and hold it down. Move the mouse diagonally to sketch a rectangle. When the rectangle is the desired size and shape, release the left button.

```
r-'l
l

l

l
l
```

- This is the "move window" prompt. Move the mouse so that the large "ghost" rectangle is in the position where you want the window. When you click the left mouse button, the window will appear at this new location.

NIL NIL is the InterLisp~D symbol for the empty list. It can also be represented by a left paren followed by a right paren: (). It is the only expression in InterLisp~D that is both an atom and a list. Pixel stands for PICTURE Element. The screen of your Lisp Machine is made up of a rectangular array of pixels. Each pixel corresponds to one bit. When a bit is turned on, i.e. set to 1, the pixel on the screen represented by this bit is black.

pretty printing Pretty printing refers to the way InterLisp~D functions are printed with special indentation, to make them easier to read. Functions are pretty printed in the structure editor, DEdit (See Section 11.3, Page 11.4). You can pretty print uncompiled functions by calling the function PP with the function you would like to see as an argument, i.e. (PP function-name). For an example of this, see Figure 1.6.

96.(PP HEAD)

```
[LANBDA (LST) <lambda>G; 'JlinH13;3&0)
(CAR LsTJ)
(HEAD)
97.'
```

Figure 1.6. An example of the pretty printing 'unmon FP

A BRIEF GLOSSARY 1.5  
I

A BRIEF GLOSSARY

Programmer's Assistant The programmer's assistant accesses the History List to allow you to FIX, UNDO, and/or REDO your previous expressions typed to the InterLisp~D executive window. (See Chapter 6.)

Promptwindow The skinny black window at the top of the screen. It displays system prompts, or prompts you have developed. (See Figure 1.7.)

Figure 1.7. Prompt window

property list A list of the form ( <property-name> <property-value> <property-name2> <property-value2> ...) associated with an atom. It is accessed by the functions GETPROP and PUTPROP.

**record** A record is a data-structure that consists of named "fields". Accessing elements of a record can be separated from the details of how the data structure is actually stored. This eliminates many programming details. A record definition establishes a record template, describing the form of a record. A record instance is an actual record storing data according to a particular record template. (See datatype, second definition.)

**Right Button Default Window Menu** This is the menu that appears when the mouse is in a window, and the right mouse button is pressed. It looks like the menu in Figure 1.8. If this menu does not appear when you depress the right button of the mouse and the mouse is in the window, move the mouse so that it is pointing to the title bar of the window, and press the right button.

Clone  
Snap  
Paint  
Clear  
Bury

Redisplay  
Hardcopy  
Move  
Shade  
Shrink

Figure 1.1. The Right Button Default Window Menu

**Symbolic expression** Short for "symbolic expression." In Lisp, this refers to any well-formed collection of left parentheses, atoms, and right parens.

**stack** A pushdown list Whenever a function is entered, information about that specific function call is pushed onto (i.e. added to the front of) the stack; this information includes the variable names and their values associated with the function call. When the function is exited, that data is popped off the stack.

**storage devices** Information is stored for your Lisp machine on floppy disks, or on the hard disk. They are referred to as (FLOPPY) and (DISK) respectively.

**system** A file containing the Lisp environment: namely, the Lisp system, everything the user has defined or loaded into the environment, the

1.6 A Brief Glossary  
I

## A BRIEF GLOSSARY

**workspace** The area of the screen that appears on the screen, the amount of memory used, and so on. Everything is stored in the system file exactly as it was when the function SYSW was called).

**TRACE** A function that creates a trace of the execution of another function. Each time the traced function is called, it prints out the values of the arguments it was called with, and prints out the value it returns upon completion.

**Unbound** Without value; an atom is unbound if a value has never been assigned to it

**window** A rectangular area of the screen that acts as the main display area for some Interlisp process,

## A BRIEF GLOSSARY 1.7

1

----- Next Message -----

Date: 19 Dec 91 14:42 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.144256pst.43009@origami.parc.xerox.com>.?::>

&lt;-----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11642>;  
 Thu, 19 Dec 1991 14:43:07 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 14:42:56 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 -----RFC822 headers----->

## 6. TYPING SHORTCUTS

Once you have logged it, as per Chapters 3 or 4, you are in Interlis~D. The functions you type into the Interlisp-D executive window will now execute, that is, perform the designated task. Please note that Interlisp-D is case-sensitive; often it matters whether text is typed in capital- or lower-case letters. The shifflock key is above the left shift key; when it is pressed (on the 1186, the red LED will be on; on the 1108, the key will be depressed), everything typed is in capital letters. You must type all Interlisp-D functions in parentheses. The Interlis~D interpreter will read from the left parenthesis to the closing right parenthesis to determine both the function you want to execute, and the arguments to that function. Executing this function is called evaluation. When the function is evaluated it returns a value, which is then printed in the Interlis~D executive window. This entire process is called the read-eval-print loop, and is how most LISP interpreters, including the one for Interlis~D, run.

The prompt in Interlis~D is a number followed by a left pointing arrow (see Figure 6.3). This number is the function's position on the History List -- a list that stores your interactions with the Interlis~D interpreter. Type the function (PLUS 3 4), and notice the number the History List assigns to the function (the number immediately to the left of the arrow). Interlis~D reads in the function and its arguments, evaluates the function, then prints the number 7.

In addition to this read-eval-print loop, there is also a programmer's assistant<sup>00</sup>. It is the programmer's assistant that prints the number as part of the prompt in the Interlis~D executive window, and uses these numbers to reference the function calls typed after them.

When you issue commands to the programmer's assistant, you will not use parentheses as you do with ordinary function calls. You simply type the command, and some specification that indicates which item on the history list the command refers to. Some programmer's assistant commands are FIX, REDO, and UNDO. They are explained in detail below.

Programmer's assistant commands are useful only at the Interlis~D top level, that is, when you are typing into the



Here are a few more examples of using the programmer's assistant:

G.a TYPING SHORrCUff  
1

TYPING SHORrCUTS  
NIL

54k[PLUS 4 5)  
9

55~REDO  
9

56#??

54 +(PLUC~ 4 5)  
9

56~(SETQ B '80Y)  
BOY  
5'~B  
BOY

59" UNDO cETQ  
SETQ undone.  
59'.B

UN8OUND nTOM  
B

SBkREDO 56  
BOY  
6IkB  
BOY  
62#

### Fqøurø 6.3. Some Applications of the Programmer's Assistant

#### 6.1 If you make a Mistake

Editing in the Interlisp-D Executive Window is explained in Section 11.2, Page 11.2. In this section, only a few of the most useful commands will be repeated.

To move the caret to a new place in the command being typed, point the mouse cursor at the appropriate position, and press the left mouse button.

To move the caret back to the end of the command being typed, press CONTROL-X. (Hold the CONTROL key down, and type ø.X'.')  
The way you choose to delete an error may depend on the amount you need to remove. To delete:

The character behind the caret simply press the backspace key  
The word behind the caret press CONTROL-W. (Hold the CONTROL key down, and rype 'øW'ø'.)

Any part of the command, first move the caret to the appropriate place in the command. Hold the right mouse button down and move the mouse cursor over the text. All of the blackened text between the caret and mouse cursor is deleted when you release the right mouse button.

## TYPING SHORTCUTS 63

### IF YOU MAKE A MISTAKE

The entire command press CONTROL-U. (Hold the CONTROL key down, and type in".)  
 Deletions can be undone. Just press the UNDO key.  
 To add more text to the line, move the caret to the appropriate position, and just type. Whatever you type will appear at the caret.

### 6.4 TYPING SHORTcUTS

----- Next Message -----

Date: 19 Dec 91 14:48 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.144827pst.43009@origami.parc.xerox.com>.:>

<----RFC822 headers-----  
 Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11544>;  
 Thu, 19 Dec 1991 14:48:38 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 14:48:27 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 -----RFC822 headers----->

### 7. USING MENUS

The purpose of this chapter is to show you how to use menus. Many things can be done more easily using menus, and there are many different menus provided in the Interlisp-D environment. Some are "po~up" menus, that are only available until a selection is made, then disappear until they are needed again. An example of one of these is the "background menu", that appears when the mouse is not in any window and the right mouse button is pressed. A background menu is shown in Figure 7.1. Yours may have different items in it.

```
SkGtL'h
LUop3 Icon
CHAT
F.lle0ro~er
```

```
sav"VM
5nap
```

Figure 7.1. A hackground menu.

Another common pop-up menu is the right button default window menu. This menu is explained more in Section 10.4, Page 10.3.

Other menus are more permanent, such as the menu that is always available for use with the Interlisp-D Filebrowser. This menu is shown in figure Figure 7.2, and the specifics of its use with the filebrowser is explained in Chapter 9).

```
Dnjelsta
Rcname
Hor~'UpJ
-='ffl.e
```



Compil'.  
E~prnng

Recrjm Ut.fl,L'

Figure 7.2. The menu that is available when using the Filehrowser

USING MENUS 71

I

## MAKING A SELECTION FROM A MENU

### 7.1 Making a Selection from a Menu

To make a selection from a menu, point with the inouse to the item you would like to selert If one of the moU5e buttons is already pressed, the menu item 5hould blacken. If it is a permanent menu, you must press the leff mouse button to blacken the item. When you release the button, the item will be chosen. Figure 7.3 shows a menu with the item "Undo" chosen.

ø ø1  
.lffer  
Bpfor',  
GeIer~,  
Replace  
'witch  
(

'3 LIt.

Find  
'=w~  
pcpflnt  
Edt

Edfl-Um

0~ik

Eva  
E.xit

Figure 73. A menu with the item "Undo" chosen

### 7.2 Explanations of Men.u Items

Many menu items have explanations associated with them. If you are not sure what the consequences of choosing a particular menu item will be, blacken the menu item, and do not release the leff button. If the menu item has an explanation associated with it, the explanation will be printed in the prompt window. Figure 7.4 shows the explanation associated with the item "Snap" from the background menu.

ile0row~or

Flguvø 7.& The explanation associated with the cliosen item, Snip, is displayed in the prom pt window.

## 7.2 USING NENUS

I

## SUBMENUS

### 7.3 Submenus

Some menu items have submenus associated with them. This means that, for these items, you can make even more precise choices if you would like to.

A submenu can also be found in one of two ways. One is to point to the item with the mouse cursor, and press the middle mouse button. If there is a submenu associated with that item, it will appear. (See Figure 7.5.)

ø  
Atter  
8e?are  
DoloCe  
Replace  
Yvitch

'ut  
l)nda  
Find  
cap  
Repnnt  
Edit

EditL'om  
Break

Eva OK

TOP

Figure 7.5. The submenu associated with the menu item Exit It appeared when the mouse cursor pointed to the menu item, and the middle mouse button was pressed.

A submenu can be indicated by a gray arrow to the right of the menu item, like the one to the right of the "Hardcopy" choice in Figure 7.1. To see the submenu, blacken the menu item, and move the mouse to follow the arrow. An example of this is shown in Figure 7.6. Choosing an item from a submenu is done in the same way as choosing an item from the menu. Any submenus that might be associated with the items in the submenu are indicated in the same way as the submenus associated with the items in the menu.

Dnclelete .~ .  
Copy

Rename  
Harjcopv

.=.ee ~~e~;

Loa.d c'.Ee!T.:  
E,puni~e

P',com Ll!eø

Figure 7.6. The submenu associated with the menu item Edit - It appeared when the menu item was blackened, and the mouse was moved to follow the gray arrow.

In summary, here are a few rules of thumb to remember about the interactions of the mouse, and system menus:

ø Press the left mouse button to select an item of a menu

Ø Press the middle mouse button to get more options - one of the ways to find a submenu

## USING MENUS 73

### SUBMENU5

Ø Press the right mouse button to see the default right button window menu, and the background menu

## 7.4 using MENUS

----- Next Message -----

Date: 19 Dec 91 14:56 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.145658pst.43009@origami.parc.xerox.com>.:>

<-----RFC822 headers-----  
 Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11659>;  
 Thu, 19 Dec 1991 14:57:09 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 14:56:58 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 -----RFC822 headers----->

## 8. HOW TO USE FILES

### 8.1 Types of Files

A program file, or lisp file, contains a series of expressions that can be read and evaluated by the Interlis~D interpreter. These expressions can include function or macro definitions, variables and their values, properties of variables, and soon. How to save Interlis~D expressions on these files is explained in Section 11.6, Page 11.7. Loading a file is explained below, in Section 8.6, Page 8.4.

Not all files, however, have Interlis~D expressions stored on them. For example, TEdit files (see Chapter 23) store text; sketches are stored on files made with the package Sketch (see Chapter 35), or can be incorporated into TEdit files. These files are not loaded directly into the environment, but are accessed with the package used to create them, such as TEdit or Sketch. When you name a file, there are conventions that you should follow. These conventions allow you to tell the type of a file by the extension to its name. If a file contains:

Interlis~D expressions, it should not have an extension. For example, a file called "MYCODE" should contain Interlis~D expressions;

compiled code, it should have the extension ".DCOM'Ø. For example, a file called 'ØMYCODE.DCOM" should contain compiled code;

a Sketch, then its extension should be ".SKETCHØØ. For example, a file called Ø'MOUNTAINS.SKETCH" should contain a Sketch;

text, it should have the extension ".TEDITØ'. For example, a file called Ø'REPORT.TEDIT'Ø should contain text that can be edited with the editor TEDIT.

### 8.2 Directories

This section focuses on how you can find files, and how you can easily manipulate files. To see all the files listed on a device, use the function DIR. For example, to see what files are stored on the hard disk, type  
(DIR (DSK))

## HOW TO USE FILES B1

### DIR—G0R1E5

To see what files are stored on the floppy disk inside of the floppy drive, type  
(DIR (FLOPPY))

Partial directory listings can be gotten by specifying a file name, rather than just a device name. The wildcard "\*" can be used to match any number of unknown characters. For example, the command  
(DIR (DSK)T\*)

will list the names of all files stored on the hard disk that begin with the letter T. An example using the wildcard is shown in Figure 8.1

```
'DIR '(P\h',(LI.\PFIL—.'.PRIMER~T';l
'LPQh"/.LI."l FILE.C\,'PRIMER\
Tsi'REF.>.TEP[1]2
T6LICINT.TEDIT,1
```

Figure 8.1. Using the function DIR with a wildcard

## 8.3 Directory Options

Various words can appear as extra arguments to the DIR command. These words give you extra information about the files.

(1) SIZE displays the size of each file in the directory. For example, type

(DIR (DSK) SIZE)

(2) DATE displays the creation date of each file in the directory. An example of this is shown in Figure 8.2

```
35~(DIR (DSK).<LI=PF1LEs>PRIMER~T* DATE)
CREATIDNOATE
```

```
(ØDSK)'LI5PFILES~PRIFLER?
```

```
TA'1"REF~TEPIT;2 26-lun-R5 19:A,O:R2
TBLnrnNT.TEDIT;1 26-lun'66 ja:4R~Ø?
```

```
3Lq~
```

```
.....:.....
```

Figure 8.2. An example using the directory option DATE

(3) DEL deletes all the files found by the directory command

## G.a HOW TO USE FILES

### SUØFILE DIREOORIES

## 8.4 Subfile Directories

Subfile directories are very helpful for organizing files. A set of files that have a single purpose, for example all the external documentation files for a system, can be grouped together into a subfile directory.

To associate a subfile directory with a filename, simply include the desired subfile directory as part of the name of the file. Subfile directories are specified after the device name and before the simple filename. The first subfile directory should be between less-than and greater-than signs < >, with nested subdirectory names only followed by a greater-than sign >. For example:

```
[DSK]<D1røctory>SubOlmctory>SrnSubDirøctor-y>..>fi1on~
```

## 8.5 To See What Files Are Loaded

If you type FILELIST<CR>, the names of all the files you loaded will display.

Type SYFILES<CR>, to see what files are loaded to create the SYSOUT.

## 8.6 Simple Commands for Manipulating Files

The following commands will work with the (FLOPPY) and other devices, but have been shown with (DSK) for simplicity.

To have the contents of a file displayed in a window:

(SEE '[DSK]f11onrn)

To copy a file: (COPYFILE '[~]o1dfi1on~ '[DSF]ne,r,,ilonrn)

An example of this is shown in Figure 8.3

```
(sOPvFILE 'T~0r,R—Fc.TEDIT 'PF;IMEFRøEFOø.T—DITJ
t'Dcxl,(LIsPFILes.PRIM—P.,ø.PRIMEP.fiEFs.TEDIT;1
```

Figure 8.3. An example of the use of the function COPYFILE

To delete a file: (DEL.FIL— '[~]fi1on~)

An example of this is shown in Figure 8.4.

```
ø,, OELFIL— 'L'AMPLE.TEPITJ
```

```
ø. \l.. I'PFILE;'. "PRIMER?>AnPLE.T—PIT;1
```

Figure 8.4. The function DELFILE

To rename a file: (RENAMEFILE ø(osK)oldftlonrn '(rSF]ner,r11on~)

## HOW TO USE FILES 83

1

## SIMPLE COMMANDS FOR MANIPULATING FILES

"LOAD" a file: Files that contain Interlisp-D expressions can be loaded into the environment. That means that the information on them is read, evaluated, and incorporated into the Interlisp-D environment.

To load a file, type:

```
(LUG '[DSff]filenm)
```

When using these functions, always be sure to specify the full filename, including subfile directories if appropriate.

## 8.7 Connecting to a Directory

Often, each person or project has a subdirectory where their files are stored. If this is your situation, you will want any files you

create to be put into this directory automatically. This means you should "connect" to the directory.

CONK is the Interlisp-D form that connects you to a directory. For example, COILK in the following figure:

- 1 1 11

```
29#(L'OtJN "CDv~K1,.LIv"PFIL~\PP,IMER7IM\!,I
t'OS'Y96)cLIy'PFIL—Cv;~PRIh1—R.~IM>
30#
```

Figure 8.5. COILK connects you to the subdirectory "PRIMERs srnbsu~i'edory „IM" connects you to the subsubdirectory iM, in the subdirectory PRIMER, in the directory LISPFILES, on the device D5K. This information, the device and the directory names down to the subdirectory you want to be connected to, is called the "path" to that subdirectory. co:: expects the path to a directory as an argument.

Once you are connected to a directory, the command DIR will assume that you want to see the files in that directory, or any of its subdirectories.

Other commands that require a filename as an argument (e.g., SEE, above) will assume, if there is no path specified with the filename, that the file is in the connected directory. This will often save you typing.

## 8.8 File Version Numbers

When stored, each file name is followed by a semicolon and a number.

```
ffile.teoiy;1
```

The number is the version number of the file. This is the system's way of protecting your files from being overwritten. Each time the file is written, a new file is created with a version number one

## 8.1 HOW TO USE FILES

### FILE VERSION NUMBERS

greater than the last. This new file will have everything from your previous file, plus all of your changes.

In most cases, you can exclude the version number when referencing the file. When the version is not specified, and there is more than one version of the file on that particular directory, the System generally uses your most recent version. An exception is the function DELFILE, which deletes the oldest version (the one with the lowest version number) if none is specified.

### HOW TO USE FILES as

----- Next Message -----

Date: 19 Dec 91 15:03 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.150359pst.43009@origami.parc.xerox.com>.:>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11664>;

Thu, 19 Dec 1991 15:04:10 PST

Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 15:03:59 -0800

From: John Sybalsky <sybalsky.PARC@xerox.com>

-----RFC822 headers----->

tO.THOSE WONDERFUL WINDOWS!

A window is a designated area on the screen. Every rectangular box on the screen is a window. While Interlisp-D supplies many of the windows (such as the Interlis~D executive window), you may also create your own. Among other things, you will type, draw pictures, and save portions of your screen with windows.

### 10.1 Windows provided by Interlisp-D

Two important windows are available as soon as you enter the Interlis~D environment. One is the Interlis~D executive window, the main window where you will run your functions. It is the window that the caret is in when you turn on your machine, and load Interlis~D. It is shown in Figure 10.1.

Figure 10.1. Interlisp-D Executive Window

The other window that is open when you enter Interlisp-D is the "Prompt Window". It is the long thin black window at the top of the screen. It displays system prompts, or prompts you have associated with your programs. (See Figure 10.2.)

Figure 10.2. Prompt Window

Other programs, such as the editors, also use windows. These windows appear when the program starts to run, and close (no longer appear on the screen) when the program is done running.

## THOSE WONDERFUL WINDOWS' 101

### CREATING A WINDOW

#### 10.2 Creating a window

To create a new window, type: (CREATEil). The mouse cursor will change, and have a small square attached to it. (See Figure 10.3.)

Figure 10.3. The mouse cursor asking you to sweep out a window  
There may be a prompt in the prompt window to create a window. Press and hold the left mouse button. Move the mouse around, and notice that it sweeps out a rectangle. When the rectangle is the size that you'd like your window to be, release the left mouse button. More specific information about the creation of windows, such as giving them titles and specifying their size and position on the screen when they are created, is given in Section 27.1.2, Page 27.2.

#### 10.3 The Right Button Default Window Menu

Position the cursor inside the window you just created, and press and hold the right mouse button. A menu of commands should appear (do not release the right button!), like the one in figure 10.4. To execute one of the commands on this menu, choose the item. Making a choice from a menu is explained in Section 7.1, Page 7.2.

clQ1,ø/

Pant

Boar  
Bury

RoJisplay  
Hardcopy~  
Move  
Shape  
shrink

#### Figure 10.5 The Right Button Default Window Menu

As an example, select "Move" from this menu. The mouse cursor will become a ghost window (just an outline of a window, the same size as the one you are moving), with a square attached to one corner, like the one shown in Figure 10.5.

~1

~1

Figure 10.1 The mouse cursor for moving a window

Move the mouse around. The ghost window will follow. Click the left mouse button to place the window in a new location.

#### 10.1 THE RIGHT BUTTON DEFAULT WINDOW MENU

f

#### THE RIGHT BUTTON DEFAULT WINDOW MENU

Choose "Shape", and notice that you are prompted to sweep out another window. Your original window will have the shape of the window you sketch out.

#### 10.4 An explanation of each menu item

The meaning of each right button default window menu item is explained below:

Close removes the window from the screen;  
Snap copies a portion of the screen into a new window;  
Paint allows drawing in a window;

Clear clears the window by erasing everything within the window boundaries;

Bury puts the window beneath all other windows that overlap it;  
Redisplay redisplay the window contents;

Hardcopy sends the contents of the window to a printer or to a file;  
Move allows the window to be moved to a new spot on the screen;  
Shape repositions and/or reshapes the window;  
Shrink reduces the window to a small black rectangle called an icon.  
(See Figure 10.6.)

#### Figure 10.6 An example icon

Expand changes an icon back to its original window. Position the mouse cursor on the icon, depress the right button, and select Expand. Or, just button the icon with the middle mouse button. These right-button default window menu selections are available in most windows, including the Interlis-D Executive window. When the right button has other functions in a window (as in an editor window), the right button default window menu should be accessible by pressing the Right button



in the black border at the top of the window.

## 10.5 Scrollable Windows

Some windows in Interlisp-D are "scrollable". This means that you can move the contents of the window up and down, or side to side, to see anything that doesn't fit in the window.

Point the mouse cursor to the left or bottom border of a window. If the window is scrollable, a "scroll bar" will appear.

THOSE WONDERFUL WINDOWS' 103

### SCROLLABLE WINDOWS

The mouse cursor will change to a double headed arrow. (See Figure 10.7.)

. 1 , 1

Figure 10.7. The scroll bar of a scrollable window. The mouse cursor changes to a double headed arrow.

The scroll bar represents the full contents of the window. The example scroll bar is completely white because the window has nothing in it. When a part of the scroll bar is shaded, the amount shaded represents the amount of the window's contents currently shown. If everything is showing, the scroll bar will be fully shaded. (See Figure 10.8.) The position of the shading is also important. It represents the relationship of the section currently displayed to the full contents of the window. For example, if the shaded section is at the bottom of the scroll bar, you are looking at the end of the file.

1 0 .

The amount of :>hadin~ in  
A::;. the scroll bar represents  
the amount of the file

;>hown in the window. Most  
of the file is visible.  
Because the shading is at  
the top of the scroll bar,  
you know you are looking  
at the top of the file.

Figure 10.1 The amount of shading in the scroll bar represents the amount of the file shown in the window. Most of the file is visible. Because the shading is at the top of the scroll bar, you know you are looking at the top of the file.

When the scroll bar is visible, you can control the section of the window's contents displayed:

0 To move the contents higher in the window (scroll the contents up in the window), press the left button of the mouse, the mouse cursor changes to look like this:

Fun 10.1. upward scrolling cursor.

The contents of the window will scroll up, making the line that the cursor is beside the topmost line in the window.

10.4 THOSE WONDERFUL WINDOWS' 103

## SCROLLASLE MN00~S

ø To move the contents lower in the window (scroll the contents "down" in the window), press the right button of the mouse, and the mouse cursor changes to look like this:

Figure 10.10. Downward scrolling cursor

The contents of the window scroll down, moving the line that is the topmost line in the window to beside the cursor.

ø To show a specific section of the window's contents, remember that the scroll bar represents the full contents of the window. Move the mouse cursor to the relative position of the section you want to see (e.g., to the top of the scroll bar if you want to see the top of the window's contents.). Press the middle button of the mouse. The mouse cursor will look like this:

Figure 10.11. Proportional scrolling cursor.

When you release the middle mouse button, the window's contents at that relative position will be displayed.

## 10.6 Other Window Functions

### 10.6.1 PROMTPRINT

Prints an expression to the black prompt window.  
For example, type

```
(P~PTPRIKT ØTNIS SILL BE PRIKTED I* THE PAT UIKOoS')
```

The message will appear in the prompt window. (See Figure 10.12.)

1 . ø1 ll

```
43 lpPROMTPRINT 'THIS WILL BE PRINTED IN THE
PROMPT WINDOW')
```

Figure 10.12. PROMTPRINTing

## THOSE WONDERFUL WINDOWS' 10.5

## OTHER WINDOW FUNCTIONS

### 10.6.2 WHICHW

Returns as a value the name of the window that the mouse cursor IS in.

(WHICHW) can be used as an argument to any function expecting a window, or to reclaim a window that has no name (that is not attached to some particular part of the program.).

## 10.6 THOSE WONDERFUL~N00vn'

----- Next Message -----

Date: 19 Dec 91 15:18 PST  
From: sybalsky:PARC:Xerox  
To: sybalsky  
Message-ID: <<91Dec19.151815pst.43009@origami.parc.xerox.com>.:>

```
<----RFC822 headers-----
Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11655>;
Thu, 19 Dec 1991 15:18:21 PST
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 15:18:15 -0800
From: John Sybalsky <sybalsky.PARC@xerox.com>
-----RFC822 headers-----
```

tl. —DITUNG AND SAVING

This chapter explains how to define functions, how to edit them, and how to save your work.

## 11.1 Defining Functions

DEFINEQ can be used to define new functions. The syntax for it is:

```
(HFiffEQ (<tunctionname> (<parameterlist↓
c~y-offrnnction>j>
```

New functions can be created with DEFINEQ by ryping directly into the Interlis~D executive window. Once defined, a function is a part of the Interlis~D environment. For example, the function EXANPLE-ADDER is defined in Figure 11.1.

-

HIL

```
46=(OEFINEQ (E.~AMPLE-rt"D&ER (~" B cJ
(PRINT 'THE SUM OF THE
THREE NUMBERS Is ")
(IPLUS n" B CJJJ
(EXn~MPLE-~&DERj
47-
```

Flure 11.1. Defining the function EXAMPLEøADDER

Now that the function is defined, it can be called from the Interlis~D executive window:

ø . -

NIL

```
49'. cEX~-MPLE-ffD&ER 3 4 'J;
"THE SUM OF THE THREE NljMBERS 15
12
```

c~g

Fq'rnre IIJ. After EXAMPL—øADDER is defined, it can he executed The function returns 12, after printing out the message. Functions can also be defined using the editor Dedit described above. To do this, simply type

```
(DF furttiornvnamej
```

EDITING AND SAVING 111

1

## DEFINING FUNCTIONS

You will be asked whether you would like to edit a Dummy definition. A dummy definition is a standard template for your

function definition. Answer by typing Y for Yes, and you will be able to define the function in the editor. (See Figure 11.3. The use of the editor is explained in Section 11.3, Page 11.4.)

```

ø h.'1,flF PJ—~HOT'E'['øTi
ø Ho FH~, dean ;or oil NflT-Eøøø 1:-7, on øoU Ui ';'øh ro ?dlr a 'lu
60A "Fløø;:øl

```

Figurn 11.1 Using DEdit to define a function

II \_ 2 \_ Simple \_ Editing \_ in \_ the \_ Interlisp-D \_ Executive Window

First, type in an example function to edit:

```

51~(øEFlxEQ (Y~R-FIRST-fuKTIrn (A B)
(if (GREATERP A B

```

```

thøn TNE FIR T IS GREATER
elsø THE SECO*O IS 6REATE)))

```

To run the function, type (YOUR-FIR<sup>ø</sup>T-FUflcTIOa 3 5).

```
52~(Y~R-FIRST-Fu~TI: 3 5)
```

```
(TNE SEC~ Is GREATER)
```

Now, let's alter this. Type:

```
53~FIZ 51 cr
```

Notf that your original function is redisplayed, and ready to edit. (SeeFigure 11.4.)

```
llJ EO1Y1~ AHO SAVING
```

```
r,
```

```
SIMPLE EDITING IN TNE INTERLISPøD EXECUTIVE WINDOW
```

```
NIL
```

```
53~FI~ 51
+(DEFINEQ
```

```
[YOUR-FIRST-FUNCTION
(A B) (ø edited;
```

```
"~1-Dec-GB 19;"8")
(IF (GREaTERPøA B)
THEN (QUOTE (THE FiRST Is
UREATERj))
```

```
ELSE (QUOTE (THE SECOND IS
u'RE~~TER] 1A
```

f~urø11.& Using FIX to editafundion

Move the tert cursor to the appropriate place in the function by positioning the mouse cursor and pressing the Jeff mouse button.

Delete text by moving the caret to the beginning of the section to be deleted. Hold the right mouse button down and move the mouse cursor over the text. All of the blackened text between the caret and mouse cursor is deleted when you release the right mouse button.

If you make a mistake deletions can be undone. On an 1108, press the OPEN key to. UNDO the deletion. On an 1108, press the UNDO key on the

keypad to the left of the keyboard.  
Now change GREATER to BIGGER:

(1) Position the mouse cursor on the G of GREATER, and click the left mouse button. The text cursor is now where the mouse cursor is.

(2) Next, press the right mouse button and hold it down. Notice that if you move the mouse cursor around, it will blacken the characters from the text cursor to the mouse cursor. Move the mouse so that the word "GREATER" is blackened.

(3) Release the right mouse button and GREATER is deleted.

(4) Without moving the cursor, type in BIGGER.

(5) There are two ways to end the editing session and run the function. One is to type CONTROL-X. (Hold the CONTROL key down, and type "X".) Another is to move the text cursor to the end of the line and press CR. In both cases, the function has been edited!

Try the new version of the function by typing:

```
58~(Y~FZIRST-F~Tzrn 8 9)
```

```
(TN— sEc~ Is BIKER)
```

and get the new result, or you can type:

```
5~RE00 52cr
```

```
(TNE SEc~ Is BIKER)
```

## EDITING AND SAVING 11.3

### USING THE LIST STRUCTURE EDITOR

#### 11.3 Using The List Structure Editor

If the function you want to edit is not readily available (i.e. the function is not in the Interlisp-D Executive window, and you can't remember the history list number, or you simply have a lot of editing), use the List Structure Editor, often called DEdit. This editor is evoked with a call to OF:

```
81~(DF YWR-FIRST-f~Tla)
```

Your function will be displayed in an edit window, as in Figure 11.5.

If there is no edit window on the screen, you will be prompted to create a window. As before, hold the left mouse button down, move the mouse until it forms a rectangle of an acceptable size and shape, then release the button. Your function definition will automatically appear in this edit window.

```
!L~nb&A IA Bj (* OJtfJ' 00:010O:cw '~;'~"0 .~.tr~r
(IF 113'REATEPP A B'i EqV;r~
THEN iO0UUTE "THE ::.p0'.T f ~Ir,GER),l cl,,t'
ELSE 1~UUTE THE .:=0E.u'N& j:. eluh'ER;J)) 4ep~:c
/'tC.h
```

```
.
Un~io
Find
```

```
Rcorint
cit.
```

```
EOlf/C T7~
Sr:ok
E0.. y
```

E..t.

#### Figure 11.1 An Edit Window

Many changes are easily done with the structure editor. Notice that by pressing the left mouse button, different expressions are underlined. Underline BIGGER as in Figure 11.5. Release the left mouse button.

To add an expression that doesn't appear in the edit window, (i.e. it can't simply be underlined), just type it in. Doing this will create an edit buffer below the DEdit window. For example,

type LARGER and hit `cr` (Remember to `cr`! You won't be able to do anything in the editor until you `cr` - this can fool you at first, so beware.) A new window opens up at the bottom for the new expression. (See Figure 11.6.)

LARGER now has the bold line underneath it, while BIGGER has a dotted line.

A

#### 11.4 EDITING ~O ~VING

##### USING THE LIST STRUCTURE EDITOR

```
, LAMDOA VA B \ ~ø ødltød 'ø3' øOøc 00 l F;3Q'ø) ArtOr
VV (OREATERP A B) Befom
~ VQUOTE -THE FIRST Is 816OER)) cOIOTo
(15 (QUOTE VTHE SEL'ONO IS BIW~Ry, \.i Ropl&ce
witch
()
```

```
y)out
Unoo
Find
wap
```

#### Figure 11.1 Edit Window with Edit Buffer

DEdit keeps track of items you have chosen by Using a stack. The underlines tell you the order of the items on the stack. The solid underline indicates the item on the top of the stack; the dotted underline indicates the second to the top. (BIGGER was pushed on first. When LARGER was pushed on, BIGGER became the second element in the "stack", and LARGER the first.)

Many commands operate with two items on the stack. Some of them are listed below:

Atter pops the stack, and adds this top item (in this example, LARGER) to the edit window after the second item on the stack (in this example, BIGGER). The item that was at the top of the stack, LARGER, will now appear in both the original and the new position.

Before pops the stack, and adds this top item (in this example, LARGER) to the edit window before the second item on the stack. (See Figure 11.7.)

```
(LAKBDA VA 8' C' oJ'lfG '3,-Oocø~ ~F;l.Oøø ,~rtOr
(IF VGREATERp A 8J E~'inre
~ (QUOTE (THE FIRST IS ~'R ,8øIUGEP); cOIgte
```

ELI (QUOTE (THE SECOND IS 8I,øb'E .j! ,1J F!Gplace  
itch  
r

tJut  
Undo  
Find  
,=-.i,1r  
P.O~rir,t  
Eda

fiUre 11.7. The command Before is chosen; the word LARGER appear  
Iefore the word BIGGER

Replace pops the stack, and substitutes this top item for the second item  
on the stack.

Sat tch changes the position of the first and second items on the stack in  
the edit window.

Find pops the stack, and searches this top expression for an occurrence  
of the second item on the stack. If the item is found, it is  
underlined with a solid line, that is, pushed on the stack. To find  
the next occurrence, simply choose "Find" again. If the  
expression is not found, the prompt window will blink, and a

EDITING AND SAVING 115  
1

## USING ~E LIST STRUCTURE EDITOR

øspøc1ø11~ asøfa1 If yri ant to &pøcø ~r coøants)  
There are other editor commands which can be very UsefUl. To  
learn about them, read to the Intertis~D Reffrence Manual,  
Volume 2, Section 16, on DEDIT.

it .4 \_ File \_ Functions and \_ Variables - \_ How to \_ See Them \_ and \_ Save Them  
With Interlis~D, all work is done inside the "Lisp Environment".  
There is no "Operating System" or "Command Level" other than  
the Interlis~D Executive Window. All functions and data  
strUctures are defined and edited using normal Interlisp-D  
commands. This sersion describes tools in the Interlisp-D  
environment that will keep track of any changes that you make  
in the environment that you have not yet saved on files, such as  
defining new functions, changing the values of variables, or  
adding new variables. And it then has you save the changes in a  
file you specify.

### 11.5 File Variables

Certain system-defined global variables are used by the file  
package to keep track of the environment as it stands. You can  
get system information by checking the values of these variables.  
Two important variables follow.

ø FILELST evaluates to a list, all files that yoU have loaded into  
the Interis~D environment.

ø filenameC0liS (Each file loaded into the Lisp environment has  
associated with it a global variable, whose name is formed by  
appending "COMS" to the end of the filename.) This variable  
evaluates to a list of all the functions, variables, bitmaps,  
windows, and soon, that are stored on that particular file.  
For example, if you type:

~FILEC0\*s

the system will respond with something like:  
FKS YouR-FZrst-Fu\*CTil )  
VARs))

### 11.6 Saving Interlisp-D on Files

The functions (FILES?) and (NAKEFILE 'filename) are useful when it is time to save function, variables, windows, bitmaps, records and whatever else to files.

## EDITING AND SAVING 117

I

### USING THE LIST STRUCTURE EDITOR

message that the item was not found will appear. (See Figure 11.8 for an example of an item, the atom THIRD, not appearing in the function, YOUR-FIRST-FUKTION.

ø1

```
L.flFBø~P~ø~T\ P _ B! (,'-.J'l-.J. _ .z' _ P...n _ 1-
THEN 'c1.lcTE _ 'THE _ FIPT _ ~I.'i'.EP']
ELSE 1øtlJJTE _ HE _ '/E/I)MID
```

TrtI,,,v Sr.i  
El

ET.

Figwø 11.& The atom THIRD is not in the fundion being edited Saap changes places, on the stack, of the first and second items on the stack. The edit window does not change, except that the expression that had a solid underline now has a dotted underline, and vice versa.

Delete works on only the top item of the stack. Delete removes the solid underlined expression from the edit window. Undo undoes the last editor command.

Completing the example begun earlier, here's how to have the word LARGER that you typed into the edit buffer appear in place of the BIGGER that you selected from the DEdit window: select the SWITCH command. Notice that the two items are switched, and the stack is popped. Now select EXIT and to leave the editor, and your function will again be redefined.

### 11.3.1 Commenting Fundions

Tert can be marked as a comment by nesting it in a set of parentheses with a star immediately after the left parenthesis.  
(ø This ii thø Von of ø c~rtt)

Inside an editor window, the comment will be printed in a smaller font and may be moved to the far right of the code. Sometimes, however, centered comments are more appropriate. To center a comment, type ,, .... after the left parenthesis.  
ø This co.oortt ø111 rtot bø rnd to thø ?ør ri9ht of thø  
co5oø but ø111 bø cørttørd)

It is also possible to insert linebreaks within a comment. A dash should be placed in the comment whcrevør A carriagø return is needed. Thii feoturø allows several commønt1 to bø placed



inside one S.t of parentheses.

(This column has 111 characters at the lineos. -

## 11.6 FIRING AND LAYING

### SAVING INTERLISP-D ON FILES

(FILES?) displays a list of variables that have values and are not already a part of any file, and then the functions that are not already part of any file.

Type:

(FILES?)

the system will respond with something like:

the variables: ~.VARIABLE CURRENT.turtle.. to be dumped.

the functions: RIGHT LEFT FOIAff LICK\*Aa CLEAR-uREEl.. to be dumped.

so to say where the above go?

If you type Y, the system will prompt with each item. There are three options:

(1) To save the item, type the filename (unquoted) of the file where the item should be placed. (This can be a brand new file or an existing file.)

(2) To skip the item, without removing it from consideration the next time (FILES?) is called, type cr This will allow you to postpone the decision about where to save the item.

(3) If the item should not be saved at all, type J. NoilhQ re will appear after the item.

Part of an example interaction is shown in the following figure:

HIL

u31~(FILES,)

The variables: MY-'y'AR. To be dumped.

the functions: MY-SECU NO-FUTLLJTJN,

YJUP-FIPOT'FUNi)TIJN

to be dumped.

want to say where the above 30 of 'ye'

(variables)

NY-VAR Nowhere

(functions)

NY-SELrnNO-FUN&'TION File name: E;~AMPL~

F~11.9. Part of an interaction using the function FILES?

(FILES?) assembles the items by adding them to the appropriate file's COMS variable. (See Section 11.5, Page 11.7.)

(FILES?) does NOT write the file to secondary storage (disks or floppies). It only updates the global variables discussed in Section 11.5.

(NAKEFILE 'Tl lenaø) actually writes the file to secondary storage. Files should only be written when the time is set. If the time is not set, you will run into problems, such as not being able to copy your file. To check

the time, type  
(return)

If the date is correct, you can safely use IRE FILE. If it is not correct, set the time with the function SETTIME. To use it, type (SETTIME date), where date is a string such as the one shown in Figure 11.10.

it. a Eomlfill ANC SAVING  
I

SAVING INTERUSP~ ON FILES

NIL

```
97;k(SETTIME "10-Jul-86 15:08 2<8)
"16-Jul-86 15:08:22 EDT"
98+
```

Figure 11.10. Using the SETTIME function to set the date and time. Once the time is set correctly, use the function MAKEFILE. Type: (MAKEFILE 'P.FILE.~)

and the system will create the file. The function returns the full name of the file created. (i.e. (DSK)MY.FILE.NAME.; 1).  
Note: Files written to (DSK) are permanent files. They can be removed only by the user deleting them or by reformatting the disk.

Other file manipulation functions can be found in Section 8.6, Page 8.3.

EDITING AND SAVING 119  
I

----- Next Message -----

Date: 19 Dec 91 15:20 PST  
From: sybalsky:PARC:Xerox  
To: sybalsky  
Message-ID: <<91Dec19.152031pst.43009@origami.parc.xerox.com>.:>

<-----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11670>;  
Thu, 19 Dec 1991 15:20:42 PST  
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 15:20:31 -0800  
From: John Sybalsky <sybalsky.PARC@xerox.com>  
<-----RFC822 headers----->

13. FLEXIBILITY AND FORGIVENESS:  
CLISP AND DWIM

CLISP, (Conversational Lisp), and DWIM, (Do What I Mean), are two Interlisp utilities that make life easier.

### 13.1 CLISP

CLISP allows the machine to understand and execute commands given in a non-standard way. For example, Figure 13.1 contains an example expression (4 + 5).

NIL

```
b'4-iJ + 5;
9
```

```
85'
```

Figure 13.1. cLisp allows the use of infix notation

Without CLISP, you would need to type this using the notation (PLUS 4 5). CLISP allows you to use expressions such as (4 + 5) for all arithmetic expressions.

CLISP also allows you to use more readable forms instead of standard Lisp control structures. Expressions like IF-THEN-ELSE statements can replace COND statements. For example, instead of:

```
(CIO 1J6RE(APLTUESRPBA B (PLUS A 10))
10
```

the following can be used:

```
(if (A ~ B) then (A + 10) else (B + 10))
```

The system translates this CLISP code into Interlisp-D code. Setting flags will allow you to either save the CLISP code, or save the translation. One such flag is CLISPIFTRANFLG; if it is set to t, all the IF statements will be replaced with the equivalent COND statements. This means that when you Dedit the function, the IF will be removed and replaced with the COND. Typically, flags such as this one are set in your INIT file. These flags are discussed in the Interlisp-D Reference Manual in Volume 2, Section 21.

FLEXIBILITY AND FORGIVENESS. cLisp AND DWIM 13 I

OWIM

13.2 DWIM

DWIM tries to match unrecognized variable and function names to known ones. This allows Lisp to interpret minor typing errors or misspellings in a function, without causing a break. Line 87 of Figure 13.2 illustrates how the misspelled 0ANNANNA was replaced by 8ANANA before the expression was evaluated.

```
NIL
```

```
a7(8ETQ0 8~0N.HA 'FRUITj
FRUIT
```

```
38'8nNN,,~NNA
=8,,H,,NA
FRUIT
39'
```

Figure 13.2. Examples of CLISP and DWIM features

Sometimes DWIM may alter an expression you didn't want it to. This may occur if, for example, a hyphenated function name (eg. (NY-FUNCTION)) is misused. If the system doesn't recognize it, it may think you are trying to subtract "FUN~ION" from "MY". DWIM also takes the liberty of updating the function, so it will

have to be fixed. However, this is as much a blessing as a curse, since it points out the misused expression!

13.2 F~11UM AND ~ROVENESα: cub AND OWN  
I

----- End Forwarded Messages -----

Figure 13.2. Examples of CLISP and DWIM features

Sometimes DWIM may alter an expression you didn't want it to. This may occur if, for example, a hyphenated function name (eg. (NY-FUNCTION)) is misused. If the system doesn't recognize it, it may think you are trying to subtract "FUN~ION" from "MY". DWIM also takes the liberty of updating the function, so it will have to be fixed. However, this is as much a blessing as a curse, since it points out the misused expression!

13.2 F~11UM AND ~ROVENESα: cub AND OWN  
I

----- End Forwarded Messages -----

Second Group

Date: 19 Dec 91 18:11 PST (Thursday)  
Posted-Date: 19 Dec 91 18:19 PST  
From: John Sybalsky:PARC:Xerox  
Subject: more primer files.  
To: porter:mv:envos

>>CoveringMessage<<

----- Begin Forwarded Messages -----

Date: 19 Dec 91 15:28 PST  
From: sybalsky:PARC:Xerox  
To: sybalsky  
Message-ID: <<91Dec19.152817pst.43009@origami.parc.xerox.com>.:>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11662>;  
Thu, 19 Dec 1991 15:28:23 PST  
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 15:28:17 -0800  
From: John Sybalsky <sybalsky.PARC@xerox.com>

-----RFC822 headers----->

#### F- 14. BREAKPACliGE

The Break Package is a part of Interlisp that makes debugging your programs much easier.

##### 14.1 Break WindoNT

A break is a function either called by the programmer or by the system when an error has occurred. A separate window opens for each break. This window works much like the Interlisp-D Executive Window, except for extra menus unique to a break window. Inside a break window, you can examine variables, look at the call stack at the time of the break, or call the editor. Each successive break opens a new window, where you can execute functions without disturbing the original system stack. These windows disappear when you resolve the break and return to a higher level.

##### 14.2 Break Package Example

This example illustrates the basic break package functions. A more complete explanation of the breaking functions, and the break package will follow.

The correct definition of FAGTORIAL is:  
(DEFIKEY (FMTORIAL (xj  
then 1

(if0f50 (ITIES x (f~ToRIAL (sue, xj

To demonstrate the break package, we have edited in an error:  
DUffKY in the IF statement is an unbound atom, it lacks a value.  
(D—FIKEY (F~T0RIAL (xj  
then ~

(if~[~~ (ITIKES x (FACTORIAL ~suei xj

The evaluated function  
(F~T0RI~ 4)

should return 24, but the above function has an error. DUMMY

is an unbound atom, an atom without an assigned value, so Lisp will "break". A break window appears (Figure 14.1), that has all the functionality of the typing Interlisp-D expressions into the Interlisp-D executive window (The top level), in addition to the break menu functions. Each consecutive break will move to another level "down".

BREAK PACKAGE 141

BREAK PACKAGE EXAMPLE

51+(PP Fllu'T&RIAL)  
cFACTORIAL

[LA'NBOR ! 'j "lrOMn—NT~t  
(if (EROP '~  
i,,ien Dummy

6Jil (lTiflEc A !FR~TORIAL !.UB1 :~j;  
!FACTCPIALj  
5?(FALTORIAL 4,1

DUMMY (in FAi',TORIALJ in =ERDP P1!t4flY  
only br'okøon!

Figuro I..l. Break window

Move the mouse cursor into the break window and hold down the middle mouse button. The Break Menu will appear. Choose BT. Another menu, called the stack menu, will appear beside the break window. Choosing stack items from this menu will display another window. This window displays the function's local variable bindings, or values. (See Figure 14.2) This new window, titled FACTORIAL Frame, is an inspector window. (See inspector Chapter 32).

Sr

fau'TUR[AL

EP.PoM5ET  
fiRE&1

UNBOUND ATOM LflQ  
DUMMY (in fAcTORIAL) in \ (ZEROP x) DUMMY) cob ø  
FLøiDRI~

(DUMMY broken) cob  
FkWRI~  
L.OB

F,c~RI~  
L'4M0

Figun 14.3. Back Yraco of trio 5system Stack

From the break window, you can call the editor for the function FACTORIAL by typing  
(OF F~15IL)

Underline X. Choose EVAL from the zditor menu. The value of X at the time of thff break will appear in the edit buffer below tho editor window. Any list or atom can be evaluated in this way (See Figure 14.3.)

## 14.1 IRF~PACMA'GF

### BREAK PACKAGE EXAMPLE

#### UNBOUND ATOM

ø DUMMY (in FACTORIAL ~ (ITIKES x \øfASTORIAL ~SUB1 X)))) Replace switch

()

ø (DUMMY broken) ()cUt

OF FAL'TORIAL) Undo

Find  
Swop  
Reprint  
Edit

EatCam  
Break  
E~a1  
E.t

Figure 14.3. Editing from the Break Window

Replace the unbound atom DuffNY with 1 ø Exit the editor with the EXIT command on the editor menu.

The function is fixed, and you can restart it from the last call on the stack (It does not have to be started again from the Top Level) To begin again from the last call on the stack, choose the last (top) FACTORIAL call in the BT menu. Select REVERT from the middle button break window, or type it into the window. TThe break window will close, and a new one will appear with the message: FACTORIAL broken.

To start execution with this last call to FACTORIAL, choose OK from the middle button break menu. The break window will disappear, and the correct answer, 24, will be returned to the top level.

14.3 \_ Ways to \_ Stop \_ Execution \_ from the \_ Keyboard, called \_ "Breaking \_ Lisp"  
There are ways you can stop execution from the keyboard. They differ in terms of how much of the current operating state is saved:

Control-G provides you with a menu of processes to Interrupt. Your process will usually be ø' EXEC". Choose it to break your process. A break window will then appear.

Control-B causes your function to break, saves the stack, then displays a break window with all the usual break functions.  
For information on other interrupt characcers, see the Interlisp Reference Manual, volume 111, page 30.1.

8BREAKPacKAG— 14.3  
I

## PROGRAMMING BREAKS AND DEBUGGING CODE

### 14.4 Programming Breaks and Debugging Code

PrOgramming breaks are put into code to cause a break when

that section of code is executed. This is very useful for debugging code. There are 2 basic ways to set programming breaks:

(BREAK functionname) This function call made at the top level will cause a break at the start of the execution of "functionname". This is helpful in checking the values of parameters given to the function.

Setting a break in the editor Take the function that you want to break into the editor. Underline the expression that should break before it is evaluated. Choose BREAK on the editor command menu. Exit the editor. The function will break at this spot when it is executed.

Once the function is broken, an effective way to use the break window for debugging is to put it into the editor window. (See Section 14.2, Page 14.2.) All the local bindings still exist, so you can use the editor's EVAL command to evaluate lists, variables, and expressions individually. Just underline the item in the usual way (move the mouse to the word or parenthesis and press the left mouse button), then choose EVAL from the command menu. (See Section 14.2 for more detail.)

Both kinds of programmed breaks can be undone using the (UNBREAK) function. Type (~KBRDF functionnm)

Calling (UNBREAK) without specifying a function name will unbreak all broken functions.

#### 14.5 Break Menu

Move the mouse cursor into the break window. Hold the middle button down, and a new menu will pop up, like the one in Figure 14.4.

OK  
BT  
BY!  
"a

Figure 14.4 The middle button menu in the Break window  
Five of the selections are particularly important when just starting to use InterlispD:

BT Stack Trace displays the stack in a menu beside the break window. Back Trace is a very powerful debugging tool. Each function call is placed on the stack and removed when the execution of that function is complete. Choosing an item on the stack will open another window displaying that item's local

1( 8~xpAcl:AGE

E~

#### BREAK MENU

variables and their bindings. This is on inspector window that offers all the power of the inspector. (For details, see the section on the Inspector, Chapter 32).

Before you use this menu option, display the stack by choosing BT from this menu, and choose a function from it. Now, choose 7: It will display the current values of the arguments to the



function that has been chosen from the stack.

~ Move back to the previous break window, or if there is no other break window, back to the top level, the InterlispØD Executive Window.

REVERT Move the point of execution back to a specified function call before the error. The function to revert back to is, by default, the last function call before the break. If, however, a different function call is chosen on the BT menu, revert will go back to the start of this function and open a new break window. The items on the stack above the new starting place will no longer exist. This is used in the tutorial example. (See Section 14.2, Page 14.1.) OK Continue execution from the point of the break. This is useful if you have a simple error, i.e. an unbound variable or a nonnumeric argument to an arithmetic function. Reset the variable in the break window, then select OK. (See Section 14.2.) (Note: In addition to being available on the middle button menu of the break window, all of these functions can be typed directly into the window. Only ST behaves differently when typed. It types the stack into the trace window instead of opening a new window.)

#### 14.6 Returning to Top Level

Typing Control-D will immediately take you to the top level from any break window. The functions called before the break will stop, but any side effects of the function that occurred before the break remain. For example, if a function set a global variable before it broke, the variable will still be set after typing Control-D.

BREAK PACKAGE 14.5

1

----- Next Message -----

Date: 19 Dec 91 15:51 PST  
From: sybalsky:PARC:Xerox  
To: sybalsky  
Message-ID: <<91Dec19.155149pst.43009@origami.parc.xerox.com>.?::>

<-----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11668>;  
Thu, 19 Dec 1991 15:51:54 PST  
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 15:51:49 -0800  
From: John Sybalsky <sybalsky.PARC@xerox.com>  
-----RFC822 headers----->

## 27. WINDOWS AND REGIONS

### 27.1 Windows

Windows have two basic parts: an area Ofi the screen containing a collection of pixels, and a property list. The window properties determine how the window looks, the menus that can be accessed from it, what should happen when the mouse is inside the window and a mouse button is pressed, and soon.

#### 27.1.1 CREATEW

Some of the window's properties can be specified when a

window is created with the function CREATEW. In particular, it is easy to specify the size and position of the window; its title; and the width of its borders.

(CREATEW region title borderwidth)

Region is a record, named REGION, with the fields left, bottom, width, and height. A region describes a rectangular area on the screen, the window's dimensions and position. The fields left and bottom refer to the position of the bottom left corner of the region on the screen. Width and height refer to the width and height of the region. The usable space inside the window will be smaller than the width and height, because some of the window's region is consumed by the title bar, and some is taken by the borders.

Title is a string that will be placed in the title bar of the window. Borderwidth is the width of the border around the exterior of the window, in number of pixels.

For example, typing:

```
(SETQ ~.WIN~ CREATEW
(CREATEW REGION 100 150 300 200)
THIS IS ~.WIN~)
```

produces a window with a default borderwidth. Note that you did not need to specify all the window's properties. (See Figure 27.1.)

WINDOWS AND REGIONS 27

## WINDOWS

```
0 .,J[1.lGJ'J (0flf,,Tfff i'0CRE"TEPE,IrnN jvjw 5- '9,, 0"~~i
"TriI;' I> My ij'IN 'ff Ibl&U'M' ii
```

```
(~[N&JwlM2.65554
```

Figure 27.1. Creating a Window

In fact, if (CREATEW) is called without specifying a region, you will be prompted to sweep out a region for the window. (See Section 10.2, Page 10.2.)

### 27.1.2 WINDOWPROP

The function to access or add to any property of a window's property list is WINDOWPROP.

(WINDOWPROP window property <value>)

When you use WINDOWPROP with only two arguments - window and property - it returns the value of the window's property. When you use WINDOWPROP with all three arguments - window, property and value - it sets the value the window's property to the value you inserted for the third argument.

For example, consider the window, MY WINDOW, created using (CREATEW). TITLE and REGION are both properties. Type (WINDOWPROP ~.WIN~ 'TITLE)

and the value of MY WINDOW's TITLE property is returned, "THIS 15 MY OWN WINDOW". To change the title, use the WINDOWPROP function, and give it the window, the property title, and the new title of the window.

```
(WINDOWPROP ~.WIN~ 'TITLE 0P FIRST ilIK~0)
```

automatically changes the title and automatically updates the window. Now the window looks like Figure 27.2.

## 27.1 WINDOWS AND REGIONS

### WINDOWS

```
70t'WINDOWfROP NV WINDOW TITLE)
IS NV OWN WINDOW"
```

```
s.(WINDOWPROP NY.WINDO!4 TITLE '0QY FIRST WINDOW")
THIS IS M\ ' OWN WINDOW"
4'.
```

Figure 27.2. TITLE is a Window Property

Altering the region of the window, NY. VINDOV, is also be done with vINDOWWPROP, in the same way you changed the title. (Note: changing either of the first two numbers of a region changes the position of the window on the screen. Changing either of the last two numbers changes the dimensions of the window itself.)

### 27.1.3 Getting windows to do things

Four basic window properties will be discussed here. They are CURSORINFN, CURSOROUTFN, CURSORffOVEDFN, and BUTTONEVENTFN.

A function can be stored as the value of the CURSORINFN property of a window. It is called when the mouse cursor is moved into that window.

Look at the following example:

(1) First, create a window called MY.WINDOW. Type:  
(SETQ P.WINDQW  
(CREATEI

```
(cCREATERE6Ia 200 200 200 200)
"THIS WIllDOW WILL IREMI0))
```

This creates a window.

(2) Now define the function SCREAMER. It will be stored on the property CURSOR1NFN. (Notice that this function has one argument, WINDOWNAME. All functions called from the property CURSOR1NFN are passed the window it was called from. So the value of MY. WINDOW is bound to WINDOWNAME. When it is called, SCREAMER simply rings bells.  
(DEFIN—Q (ScREMER (WIK~~E)  
RIilBELLS)

```
PROIPTPRIIT TAT - IT WDRFSI")
```

```
RIKBELLS)))
```

(3) Now, alter that window's CURSORINFN property, so that the system calls the function SCREAMER at the appropriate time. Type:

## WINDow5 AND REGIONS 273

### WINDOWS

```
(WIN~PRoP P.wINI;0II 'cuR~RIafø
(F~IIk:TIK IR~R))
```

(4) After this, when you move the mouse cursor into MY.WINDOW, the CURSORINFk property's function is called, and it rings beJls tvvice.

CURSORINFN is one of the many window properties that come with each window - just as REGION and TITLE did. Other properties include:

**CURSOROUTFN** The function that is the value of this property is executed when the cursor is moved out of a window;

**CURSORMOVEDFN** the function that is the value of this property is executed when the cursor is moved while it is inside the window;

**BUTTONEVENTFN** the function that is the value of this property is executed when either the left or middle mouse buttons are pressed (or released).

Figure 27.3 shows MY.WINDOW's properties. Notice that the CURSORINFk has the function SCREAMER stored in it. The properties were shown in this window using the function INSPECT. INSPECT is covered in Chapter 32.

```
.. ' 1 ø
GREEN NIL

HI NOø'rtENTR'[FN O liE. TT'(PRObES
PRfLESS NIL
',181)ROER 4
NEWREL'I)NF4 NIL

'NTITLE øTHIS 'ffiINDOW 'tILL .QCREAn!"
MOIEfN NIL
CLOSEFN NIL
HORIZOCROLL'.yIND1)',t NIL
"ER1L'ROLLNINøO'ff NIL
c.u'ROLLFN NIL
H)RI=J-'cRILLREG NIL
":'ERTSCR)LLREU NIL
USERDATA NIL
E!'!TENT NIL
REOH4PEFN NIL
REPAINTFN NIL
L'URSORttOvEDFN NIL
CURSOROUTFN NIL
CURSORINFN SCCE'øThER
RIGHTBUTTONFN NIL
BU1FONEVENTFN TOTOPU
REG 12J0 "L)9 øJ~ '36!
SavE (BITMAP~øł3,1jo52ł
NE~('t (WiflD1)'-'1j55,1'lj'..ø8
DSP ~5TRE>M\,ø~øF,jjjj~4
```

Figure 27.3. Inspecting MY.wlNDow for MouseRelated Window Properties  
You can define functions for the values of the properties CURSOROUTFK and CURSORMOVEDfN in much the same way as you did for CURSORINFN. The function that is the value of the property BUTTOHEVENTFN, however, can be specialized to respond in different ways, depending on which mouse button is pressed. This is explained in the next section.

### 27.1.3.1 BUtrON—VENTFN

BUTTONEVENTFK is another property of a window. The function that is stored as the value of this property is called when the mouse is inside the window, and a mouse button is pressed. As an example of how to use it type:

```
27A ~N00WS AND REGIONS
```

```
witurows
```

```
(wI~PKP :iIK~ 'euTTW"EKTtr
(F~T15 ScREAøER))
```

When the mouse cursor is moved into the window, bells will ring because of the CURSORINFN, but it will also ring bells when either the left or middle mouse button is pressed. Notice that the right mouse button functions as it usually does, with the window manipulation menu. If only the left button should evoke the function SCREAMER, then the function can be written to do just this, using the function MOUSESTATE, and a form that only MOUSESTATE understands, ONLY. For example:

```
(DEFIKEQ
```

```
(SCREIERZ WIK~)
(if ESTATE (aLY LEFT))
thøa (RIKB—LLS)))
```

In addition to (ONLY LEFT), MOUSESTATE can also be passed (ONLY MIDDLE), (ONLY RIGHT) or combinations of these (e.g. (OR (ONLY LEFT) (ONLY MIDDLE))). You do not need to use ONLY with MOUSESTATE for every application. ONLY means that that button is pressed and no other. If you do write a function using (ONLY RIGHT), be sure that your function also checks position of the mouse cursor. Even if you want your function to be executed when the mouse cursor is inside the window and the right button is pressed, there is a convention that the function DOVINDOWCOM should be executed when the mouse cursor is in the title bar or the border of the window and the right mouse button is pressed. Please program your windows using this tradition! For more information, please see the Intertisp-D Reference Manual, Volume 3, Chapter 28, Pages 7 and 28.

Please refer to the Intertisp Reference Manual, Volume 3, Chapter 28, for more detail and other important functions.

#### 27.1.4 Looking at a window's properties

INSPECT is a function that displays a list of the properties of a window, and their values. Figure 27.3 shows the INSPECT function run with MYØWINDOW. Note the properties introduced in CREATEW: WBORDER is the window's border, REG is the region, and WTITLE is the window's title.

#### 27.2 Regions

A region is a record, with the fields LEFT, BOTTOM, WIDTH, AND HEIGHT. LEFT and BOTTOM refer to where the bottom left hand corner of the region is positioned on the screen. WIDTH and HEIGHT refer to the width and height of the region. CREATEREGION creates an instance of a record of type REGION. Type:

```
(SETO ~.REG1a (CREATERES11 15 100 200 450))
```

WINDOWS AND REGIONS 275

## REGIONS

to create a record of type REGION that denotes a rectangle 200 pixels high, and 450 pixels wide, whose bottom left corner is at position (15, 100). This record instance can be passed to any function that requires a region as an argument, such as CREATEV, above.

a., WIN00WS ANO REGIONS

----- Next Message -----

Date: 19 Dec 91 15:59 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.155935pst.43009@origami.parc.xerox.com>.:>

<-----RFC822 headers-----  
 Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11672>;  
 Thu, 19 Dec 1991 15:59:45 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 15:59:35 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 -----RFC822 headers----->

## 28. WHAT ARE MENUS?

While Interlisp-D provides a number of menus of its own (see Section 7.1, Page 7.2), this section addresses the menus you wish to create. You will learn how to create a menu, display a menu, and define functions that make your menu useful. Menu's are instances of records (see Chapter 24). There are 27 fields that determine the composition of every menu. Because Interlisp-O provides default values for most of these descriptive fields, you need to familiarize yourself with only a few that we describe in this section.

Two of these fields, the TITLE of your menu, and the ITEMS you wish it to contain, can be typed into the Interlisp-D Executive window as shown below:

NIL

```
33'(ETO MY. MEN (CRE""TE ME/lb
TiTLE „PLE~~SE CHCio8— ONE OF THE
ITEMS"
```

```
ITEMS (0,LIIT NE,T-l)UE;STION
NE;~T-TOPIL SEE-TOPIC;5'JJJ
,rMENU!,#c4, ij'ø:'3jH
```

Figure 28.1. Creating a menu

Note that creating a menu does not display it. MY.MENU is set to an instance of a menu record that specifies how the menu will look, but the menu is not displayed.

### 28.1 Displaying Menus

Typing either the MENU or ADDMENU functions will display your menu on the screen. MENU implements pop-up menus, like the Background Menu or the Window Menu. ADDMEHU puts menus

into a semi-permanent window on the screen, and lets you select items from it.

(MENU MENU POSITION) pops-up a menu at a particular position on the screen.

Type:

(\*EKU MY.ffi KIL)

to position the menu at the end of the mouse cursor Note that the POSITION argument is NIL. In order to go on, you must either choose an item, or move outside the menu window and

WHAT ARE MENUS' 281

DISPLAYING MENUS

press a mouse button. When you do either, the menu will disappear. If you choose an item, then want to choose another, the menu must be redisplayed.

(ADONENU menu window position) positions a permanent menu on the screen, or ;n an existing window.

Type:

(ADIEKU P.\*EI)

to display the menu as shown in Figure 28.2. This menu will remain active, (will stay on the screen) without stopping all the other processes. Because ADONEliU can display a menu without stopping all other processes, it is very popular in users programs. If window is specified, the menu is displayed in that window. If window is not specified, a window the correct size for the menu is created, and the menu is displayed in that window. If position is not specified, the menu appears at the current position of the mouse cursor.

NE..TQøUESIICN

3EEToPIC> .

.

Figure 28.2. A Simple Menu, displayed with AooNriU.

## 28.2 Getting Menus to DO Stuff

One way to make a menu do things is to specify more about the menu items. Instead of items simply being the strings or atoms that will appear in the menu, items can be lists, each list with three elements. (See Figure 28.3.) The first element of each list is what will appear in the menu; the second expression is what is evaluated, and the results of the evaluation returned, when the item is selected; and the third expression is the expression that should be printed in the Prompt window when a mouse button is held down while the mouse is pointing to that menu item. This third item should be thought of as help text for the user. If the third element of the list is NIL, the system responds with "Will select this item when you release the button".

JGJ WHAT AR5 MENUS?

GErn~ MENUS TO DO STUFF

NIL

```

17+(SETQ Nv.MENU2 (SR—ATE MENU
TITLE "PLEASE LHOOSE ONE OF THE ITEMS"
I~.EMS '(VQUIT
(PRINT "STOPPEO" \
"LHOOSE THIS TO 50~ø",
(NE\T-QUESTIOH
(PRINT "HERE IS TME NE.'\T QLI—STIOH .
øu'HOOSE THIS TO ~E ISKED THE NE."T QUESTION",
iNE!~T-TOPIL
(PRINT øøHERE IS THE NE'~T TOPIL .
"C.HOOSE THIS TO KOv— OH TO THE NE'\T SueJELT" '1
(SEE-TOPICS
(PRINT "THE FOLLOYIN6 HA'\E NOT e.EEN L—ARNEO",
*CHOOSE THIS TO SEE THE TOPICS NOT YET LErtRNEO"l 'ii
ø~~MENU,#5~. '.5~5j
1qL(cl&MENL MY. MEtIU:'
,rNIN&EL'~~~4', 175350
14

```

Figure 28.3. Creating a menu that will do things, then displaying it with the function ADDNEHU

Now when an item is selected from KY.KENU2, something will happen. When a mouse button is held down, the expression typed as the third element in the item's specification will be printed in the Prompt window. (See Figure 28.4.)

```

NE7.T.'JUE'=TIE'r~J
SEE-TOPIC'

```

Figure 28.1. Mouse Button Held Down While Mouse Cursor Selected  
NEXT-QUESTION

When the mouse button is released (i.e. the item is selected) the expression that was typed as the second element of the item's specification will be run. (See Figure 28.5.)

```

Y-'OUE'TI"N
'EETOPir"
"HERE IS THE NEXT ilUETION.

```

Figure 28.5. NEXT-QUESTION Selected

WHAT ARE MENUS' 283

GEHING MENUS TO DO STUFF

28.2.1 \_ The WHENH—LDFN \_ and WENSEL—CTEDFN fields of a \_ menu  
Another way to get a menu to do things is to define functions, and make them the values of the menu's WHENHELDNFN and WHENSELECTEDFN fields. As the value of the WHENHELDNFN field of a menu, the function you defined will be executed when you press and hold a mouse button inside the menu. As the value of the WHENS—L—CTEDFN field of a menu, the function you defined will be executed when you choose a menu item. This example has the same functionality as the previous example, where each menu item was entered as a list of three items.



As an example, type in these two functions so that they can be executed when the menu is created and displayed:

```
(DEFIKEY L—CTED

(SELECTfQPiNTEENNUJSENHENELO (ITEM.S—LECTED a:. FROM BUTT:. PRESSED)
QUIT (PROMPTPRIKT ØCHOOSE THIS TO sToPØ))

NEXT-QUESTION (PROMPTPRIKT CHOOSE THIS TO BE ASKED TNE NEXT QUESTION-))
NEXT-TOPIC PROMPTPRINT ØCHOOSE THIS TO MOOE a TO THE NEXT SUBUJECTØ))
SEE-TOPICS PROMPTPRINT ØCHDOSE THIS TO SEE THE TOPICS NOT YET L—ARNEDØ))
ERROR (PROM TPRIKT NO liTCH FOUNDØ))))

(DEFINEQ WENSELECTED (ITEM.SELECTED MENU. FROM 8UTT:.PRESSED)
QUIT (PRINT ØSTOPPEDØ))

NEXT-QU RINT "HERE IS THE NEXT QUESTION...))
NEXT-T ØHERE IS THE NEXT TOPIC. .
- PICS PRINT ØTHE FOLLONIK HAVE NOT 8EEN LEARNED. ..Ø

ERROR (PRONFTPRINT NO liTCH FOUNDØ))))
```

Now, to create the menu, type:

```
(SETQ MY.NE:3 (CREATE NE:
TITLE ØPLEASE CHOOSE :E OF THE ITEMSØ
ITEK '(QUIT NEXT-QUESTION NEXT-TOPIC SEE-TOPICS)
NHENHELDFN (FUNCTIK MY.NENU3.NHENHELD)
ffIENSELECTEDFN (FUNCTION NY, .MENU3 .ffIENSELECTED)))
```

Type

```
(ADDMENU MY.MENU3)
to see your menu work.
```

NOW, due to executing the WH—NNELDNFN function, holding down any mouse button while pointing to a menu item will display an explanation of the item in the prompt window. The screen will once again look like Figure 28.4 when the mouse button is held when the mouse cursor is pointing to the item NEXT-TOPIC.

Now due to executing the WHENSELECTEDFN function, releasing the mouse button to select an item will cause the proper actions for that item to be taken. The screen will once again look like Figure 28.5 when the item NEXT-TOPIC is selected. The crucial thing to note is that the functions you defined for WHENHELDFN and WHENSELECTEDFN are automatically given the following arguments:

(1) the item that was sØlected, ITEM. SELECTED;  
(2) the menu it was selected from, MENU. FROM;  
(3) and the mousØ button that was pressed BUTTON PRESSED.  
NotØ: thesØ functions, \*Y.NENU3.ffIENfIELO and  
ffY.KEKUIJ.ilHEKSELCTEO, wØre quoted using FUKCTIOK  
instead of QUOTE both for program rØadability and so that the

21.1 ~YAR1".NUs?

GETTING MENUS TO ØØ STUFF

compiler con produce foster code when the program is compiled. It is good style to quote functions in Intertisp by using the function FUNCTION instead of QUOTE.

### 28.3 Looking at a menu's fields

INSPECT is a function that displays a list of the fields of a menu, and their values. The Figure 28.6 shows the various fields of NY .MENU3 when the function (INSPECT NY .MENU) was called. Notice the values that were assigned by the examples, and all the defaults.

```
\JN"PELT NY liENl./300l
011IHdu'wJ#1, 540scj

NENllPECICNB1:TTi=fl o

Imrni';E (0!VINDLlrt0#b1.lh5lSjl

0 t1)UIT HE0~T-Ll0L'E'TI1=1N '0'-Ti'iFl'0 ET
0 MENUPOffo0'

0 ANUEAFF'ETFLL: NIL

ffENUEQHT i:FclNTPc:::cf IpTclFt -a
TITLE '0PLEAL'E CHil.l 'HE ,iF THE ITE
0 fFEHlJoFF6ET A
LECTEDFN fly flEflJ,0 h.0rtEf:EL.FCTE—l

'1flE'fIELDFH NV flEPILl3 \0rtEHHELJP
0 ENl)NHELoFH l:LFF'RCHPt

0 flENOFEEOe4l,'r.FLG NIL
```

Figure 28.6. The Fields of MY.MENU3

WHAT ARE MENUS' 285

----- Next Message -----

Date: 19 Dec 91 16:10 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.161052pst.43009@origami.parc.xerox.com>.:>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11680>;  
 Thu, 19 Dec 1991 16:10:56 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:10:52 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 -----RFC822 headers----->

### 29. lITMAPS

A bitmap is a rectangular array of dots. The dots are called pixels (for picture elements). Each dot, or pixel, is represented by a single bit. When a pixel or bit is turned on (i.e. that bit set to 1), a black dot is inserted into a bitmap. If you have a bitmap of a floppy on your screen, (Figure Figure 29.1), then all of the bits in the area that make up the floppy are turned on, and the surrounding bits are turned off.

```
FLOPPY
(la b~JwP-
~,5,,Bh
```

```
(t-)o
```

Figure 29.1. Bitmap of a Floppy

BITMAPCREATE creates a bitmap, even though it can't be seen.  
(BITMAPCREATE width height)

If the width and height are not supplied, the system will prompt you for them.

EDITBN edits the bitmap. The syntax of the function is:  
(EDITBN bitmapname)

Try the following to produce the results in Figure 29.4:  
l~SoETiQr:~røB!sTHituPbB~I~PcRDTE eo 40))

To draw In the bitmap, move the mouse into the gridded section of the bitmap editor, and press and hold the left mouse button. Move the mouse around to turn on the bits represented by the spaces in the grid. Notice that each space in the grid represents one pixel on the bitmap

To erase Move the mouse into the gridded section of the bitmap editor, and press and hold the center mouse button. Move the mouse around to turn off the bits represented by the spaces in the gridded section of the bitmap editor.

To work on a different section Point with the mouse cursor to the picture of the actual bitmap (the upper left corner of the bitmap editor). Press and hold the

BITMAPS 291

BITMAPS

Left mouse button. A menu with the single item, move will appear. (See Figure 29.2.) Choose this item.

..

Figure 29.2. Move the mouse cursor to the picture of the bitmap. Press and hold the left mouse button, and the Move menu will appear

You will be asked to position a ghost window over the bitmap. This ghost window represents the portion of the bitmap that you are currently editing. Place it over the section of the bitmap that you wish to edit. (See Figure 29.3.)

..

.

..

....

.... I .

29.3. .. J=.. :: II.II::;;, \_ ..

figure After you choose move, you will be asked to position a ghost

window like this one. Position it by clicking the left mouse button when the

ghost window is over the part of the picture of the bitmap you would like to edit. To end the session bring the mouse cursor into the upper-right portion of the window (the grey area) and press the center button. Select OK from the menu to save your artwork.

29) .IY~

r'. alTMAps

```

.: 5"iSETQ ffy IllNAP (I[TNAPcPEATE OR GO)
j:y.IlfM&P osøt\
,A.BITMAPlø6',1.q;hO
58oi,E0iIBM my.IITNAP\

```

--

.

.

-A

fr.j:

,

= ""~

=

. ~.

.

..

Figure 29.4. Editing a Bitmap

BITBLT is the primitive function for moving bits (or pixels) from one bitmap to another. It extracts bits from the source bitmap, and combines them in appropriate ways with those of the destination bitmap. The syntax of the function is:

```

(BITBLT sourcebitmap sourceleft sourcebottom
destinationbitmap destinationleft destinationbottom width
height sourcetype operation texture clippInregion)

```

Here's how it's done - using MY.BITMAP as the sourcebitmap and MY.WINDOW as the destinationbitmap.'

```
(BITBLT rn.BITMAP NIL NIL
```

```
P.wIN~ NIL NIL KIL NIL 'INPUT 'REPUCE)
```

Note that the destination bitmap can be, and usually is, a window. Actually, it is the bitmap of a window, but the system handles that detail for you. Because of the ILLs (meaning "use the default"), MY.BITMAP will be BITBLT'd into the lower right hand corner of MY.WINDOW. (See Figure 29.5.)

BITMAPS 293

~17MAP5

```
98'(BITBLT KY Strap NIL NIL my ,1(10p,, FL 'IL NIL HIL Tipil' P.—PLlfi
```

```
(~='l',
```

Figure 29.5. 9ITBLTng a Bitmap onto a Window

Here is what each of the SITBLT arguments to the function mean:

sourcebitmap the bitmap to be moved into the destinationbitmap  
sourceleft a number, starting at 0 for the left edge of the sourcebitmap,  
that tells SITBLT where to start moving pixels from the  
sourcebitmap. For example, if the leftmost 10 pixels of

sourcebitmap were not to be moved, sourceleft should be 10. The default value is 0.

sourcebottom a number, starting at 0 for the bottom edge of the sourcebitmap, that tells BIT6LT where to start moving pixels from the sourcebitmap. For example, if the bottom 10 rows of pixels of sourcebitmap were not to be moved, sourcebottom should be 10. The default value is 0.

`destinationbitmap` the bitmap that will receive the `sourcebitmap`. This is often a window (actually the bitmap of a window, but Interlisp-b takes care of that for you).

destinationleft a number, starting at 0 for the left edge of the destinationbitmap, that tells BITBLT where to start placing pixels from the sourcebitmap. For example, to place the sourcebitmap 10 pixels in from the left, destinationleft should be 10. The default value is 0.

`destinationbottom` a number, starting at 0 for the bottom edge of the `destinationbitmap`, that tells BITBLT where to start placing pixels from the `sourcebitmap`. For example, to place the `sourcebitmap` 10 pixels up from the bottom, `destinationbottom` should be 10. The default value is 0.

width how many pixels in each row of `sourcebitmap` should be moved. The same amount of space is used in `destinationbitmap` to receive the `sourcebitmap`. If this argument is `NIL`, it defaults to the number of pixels from `sourceleft` to the end of the row of `sourcebitmap`.

height how many rows of pixels of sourcebitmap should be moved. The same amount of space is used in destinationbitmap to receive the sourcebitmap. If this argument is NIL, it defaults to the number of row; from sourcebottom to the top of the sourcebitmap.

sourcetype refers to one of three ways to convert the source bitmap for writing. For now, just use 'INPUT.

29.0 ps

[illegible]

## OILMAPS

operation refers to how the sourcebitmap gets BITBLT'd on to the destinationbitmap. 'REPLACE' will BLT the exact sourcebitmap. Other operations allow you to AND, OR or XOR the bits from the sourcebitmap onto the bits on the destinationbitmap.  
texture Just use NIL for now.  
clippingregion just use NIL for now.

For more information on these operations, see the Interlisp-D Reference Manual, Volume 3, Chapter 27, Page 14.

Sourcebitmap, sourceleft, sourcebottom, destinationbitmap, destinationleft, destinationbottom, width and height are shown in Figure 29.6.

```
Destination Bitmap
Source Bitmap
FLOPPY
tlcblffkUP'
```

3/S/Bh height

e./,o

width

Source leh. Source bottom. The "x y coordinates in terms of the source (OOforthewhoiesource).

Destination Jeff, Dertination Bottom. The „x y" coordinates in terms of the destination bitmap. (00 to put the source bitmap in the left bottom corner of the dertination bitmap).

Figure 29.6. BITBLT'ed Bitmap of a Floppy

BITMAPS 295

----- Next Message -----

Date: 19 Dec 91 16:16 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.161653pst.43009@origami.parc.xerox.com>.:>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11679>;  
 Thu, 19 Dec 1991 16:16:57 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:16:53 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 -----RFC822 headers----->

### 30. DISPLAYSTREAMS

A displaystream is a generaJized "place to display". They determine exactly what is displayed where. One example of a displaystream is a window. Windows are the only displaystreams that will be used in this chapter. If you want to draw on a bitmap that is not a window, other than with BITBLT, or want to use other types of displaystreams, please refer to the Interlisp-D Reference Manual, Volume 3, Chapter 27.

This chapter explains functions for drawing on displaystreams: DRAWLINE, DRAWTO, DRAVCIRCLE., and FILLCIRCLE. In addition, functions for locating and changIng your curreAt position in the displaystream are covered: DSPXPOSITION, DSPYPOSITION, and NOVETO.

#### 30.t Drawing on a Displaystream

Examples will show you how the functions for drawing on a display stream work. First, create a window. Windows are displaystreams, and the one you create will be used for the examples in this chapter. Type:  
 (SETO EwPLE.wIN~ (CREATEI))

##### 30.1.1 DRAWLINE

DRAWL IRE draws a line in a displaystream. For example, type:  
 (DliVLIKE 10 IS loo 150 S øIlERT ExMPLEwIN~)  
 The results should look like this:

Figure 30.1. The line drawn onto the displaystream, ExAMPLEwINDoW

### DISPLAYSTREAMS 30

#### DRAWING ON A DISPLAYsTaE:M

The syntax of DRAWLINE is

(Dli~IKE x1 y1 x2 y2 width operation stream ø)

The coordinates of the Jeff bottom corner of the displaystream areOO.

x1 and y1 are the x and y coordinates of the beginning of the line;  
x2andy2 are the ending coordinates of the line;  
width isthe width of the line, in pixels

operation is the way the line is to be drawn. INVERT causes the line to invert the bits that are already in the displaystream. Drawing a line the second time using INVERT erases the line. For other operations, see the Interlis~D Reference Manual, Volume 111, Page 27.15.

stream is the displaystream. In this case, you used a window.

#### 30.1.2 ORA~O

DRAWTO draws a line that begins at your current position in the displaystream. For example, type:

(Dli~O 120 135 5 'IrvERT E~LE.\*IH~)

The results should look like this:

Figure 30.2. Another line drawn onto the displaystream, ExAMPLEøWINDowø

The syntax of ORAWTO is

(oliilTO x y width operation stream i)

The line begins at the current position in the displaystream.

x is the x coordinate of the end of the line;

y is they coordinate of the end of the line;

width is the width of the line

operation is the way the lino is to be drawn. INVERT causes the line to invert the bits that aro already in tho displaystream. Drawing a line the second time using INVERT erases the line. For other operations, see the InteHi~O Reference Manual, Volume III, Page 27.15.

stream is the displaystream. In this case. you used a window.

### 30.2 IPLAYSTQCANT

#### DRAW1NG ON A D15PLAr5~E~

#### 30.1.3 DRAWCIRCLE

DRAWCIRCLE draws a circle on a displaystream. To use it, type:

(0li~I~LE 150 100 so '(~RTICAL 5) KIL E~LE .VI~)

Now your window, EXAMPLE.WINDOW, should look like this:

Flurø 30.3. The circle drawn onto the displaystream. EXAMPLE WINDOW

The syntax of DRAWCIRCLE is

(0li~IEL— centerx centery radius brush dashing stream)

centerx is the x coordinate of the center of the circle

centery is they coordinate of the center of the circle

radius is the radius of the circle in pixels

brush is a list. The first item of the list is the shape of the brush. Some of your options include ROUND, SQUARE, and VERTICAL. The second item of that list is the width of the brush in pixels. DASHING is a list of positive integers. The brush is "on" for the number of units indicated by the first element of the list, "off" for the number of units indicated by the second element of the list. The third element specifies how long it will be on again, and so forth. The sequence is repeated until the circle has been drawn. stream is the displaystream. In this case, you used a window.

### 30.1.3.1 FILLCIRCLE

FILLCIRCLE draws a filled circle on a displaystream. To use it, type:

```
(FILLCIRCLE 200 150 10 6liY~DE ExlPLE.wIli~)
EXAMPLE.WINDOW now looks like this:
```

```
DISPLAYSTREAMS 303
l
```

### DRAWING ON A DISPLAYSTREAM

Figure JO.t A filled circle drawn onto the displaystream, EXAMPLE WINDOW  
The syntax of FILLCIRCLE is

```
(FILLCIRCLE— centerx centery radius texture stream)
centerx is the x coordinate of the center of the circle
centery is the y coordinate of the center of the circle
radius is the radius of the circle in pixels
```

texture is the shade that will be used to fill in the circle. Interlisp-D provides you with three shades, WHITESHADE, BLACKSHADE, and GRAYSHADE. You can also create your own shades. For more information on how to do this, see the Interlisp-D Reference Manual, Volume III, Page 27.7.

stream is the displaystream. In this case, you used a window. There are many other functions for drawing on a displaystream. Please refer to the Interlisp-D Reference Manual, Volume III, Chapter 27.

Text can also be placed into displaystreams. To do this, use printing functions such as PRINT1 and PRINT2, but supply the name of the displaystream as the "file" to print to. To place the text in the proper position in the displaystream, see Section 30.2, Page 30.4.

### 30.2 Locating and Changing Your Position in a Displaystream

There are functions provided to locate, and to change your current position in a displaystream. This can help you place text, and other images where you want them in a displaystream. This primer will only discuss three of these. There are others, and they can be found in the Interlisp-D Reference Manual, Volume III, Chapter 27.

### 30.4 DISPLAYSTREAM

r.

### LOCATING AND CHANGING YOUR POSITION IN A DISPLAYSTREAM

#### 30.2.1 DISPLAYPOSITION



DSPXPOSITION is a functiOn that will either change the current x pOsition in a displaystream, or simply report it. To have the function report the current x position in EXAMPLE.WINDOW, type:

```
(OSP*PoSiTiOn NIL EXlPLE .ilINDON)
```

DSPXPOSITION expects two arguments. The first is the new x position. If this argument is NIL, the current position is not changed, merely reported. The second argument is the displaystream.

### 30.2.2 DSPYPOSITION

DSPYPOSITION is an analogous function, but It changes or reports the current y position in a displaystream. As with DSPXPOSITION, If the first argument Is a number, the current y position will be changed to that position. If it is NIL, the current position is simply reported. To have the function report the current y position in EXAMPLE.WINDOW, type:

```
(DSPYROSITIoN NIL ExlPLE.WIK-~)
```

### 30.2.3 MOVETO

The function NOVETO always changes your position in the displaystream. It expects three arguments:  
(~~ET0 xystream)

x is the new x position in the display stream  
y is the new y position in the display stream

stream is the display stream. The examples so far have used a window.

```
DISPLAYSTREAMS 30 5
```

----- Next Message -----

Date: 19 Dec 91 16:30 PST  
From: sybalsky:PARC:Xerox  
To: sybalsky  
Message-ID: <<91Dec19.163054pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11682>;  
Thu, 19 Dec 1991 16:30:58 PST  
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:30:54 -0800  
From: John Sybalsky <sybalsky.PARC@xerox.com>  
-----RFC822 headers----->

## 31. FONTS

This chapter explains fonts and fontdescriptors, what they are and how to use them, so that you can use functions requiring fontdescriptors

You have already been exposed to many fonts in Interlisp-D. For example, when you use the structure editor, DEdit, (See Section 11.3.), you noticed that the comments were printed in a smaller font than the code, and :hat CLISP words (See Section 13.1, Page 13.1.) were printed in a darker font than the other words in the

function. These are only some of the fonts that are available in Interlisp-D.

In addition to the fonts that appear on your screen, Interlisp-D uses fonts for printers that are different than the ones used for the screen. The fonts used to print to the screen are called DISPLAYFONTS. The fonts used for printing are called INTERPRESSFONTS, or PRESSFONTS, depending on the type of printer.

### 31.1 What makes up a FONT?

Fonts are described by family, weight, slope, width, and size. This section discusses each of these, and describes how they affect the font you see on the screen.

Family is one way that fonts can differ. Here are some examples of how "family" affects the look of a font:

CLASSIC This family makes the word "Able" look like this: Able  
 MODERN This family makes the word "Able" look like this: Able  
 TERMINAL This family makes the word "Able" look like this: Able  
 Weight also determines the look of a font. Once again, "Able" will be used as an example, this time only with the Classic family. A font's weight can be:  
 BOLD and look like this: Able  
 MEDIUM or REGULAR and look like this: Able  
 The slope of a font is italic or regular. Using the Classic family font again, in a regular weight, the slope affects the font like this:

ITALIC looks like this: A file  
 REGULAR looks like this: Able

FONT5 311  
 1

### WHAT MAKES UP A FONT?

The width of a font is called its "expansion". It can be COMPRESSED, REGULAR, or EXPANDED.

Together, the weight, slope, and expansion of a font specifies the font's "face". Specifically, the face of a font is a three element list:

(weight slope expansion)

To make it easier to type, when a function requires a font face as an argument, it can be abbreviated with a three character atom. The first specifies the weight, the second the slope, and the third character the expansion. For example, some common font faces are abbreviated:

MRR This is the usual face, MEDIUM, REGULAR, REGULAR;  
 MIR makes an italic font. It stands for: MEDIUM, ITALIC, REGULAR;  
 BRR makes a bold font. The abbreviation means: BOLD, REGULAR, REGULAR;

BIR means that the font should be both bold and italic. BIR stands for BOLD, ITALIC, REGULAR.

The above examples are used so often, that there are also more mnemonic abbreviations for them. They can also be used to

specify a font face for a function that requires a face as an argument. They are:

**STANDARD** This is the usual face: MEDIUM, REGULAR, REGULAR. It was abbreviated above, MRR;

**ITALIC** This was abbreviated above as MR, and specifies an italic font; **BOLD** of course, makes a bold font. It was abbreviated above, BRR; **BOLDITALIC** means that the font should be both bold and italic: BOLD, ITALIC, REGULAR. It was abbreviated above, BIR.

A font also has a size. It is a positive integer that specifies the height of the font in printers points. A point is, on an 1108 screen, about 1/72 of an inch. On the screen of an 1186, a point is 1/80 of an inch. The size of the font used in this chapter is 10. For comparison, here is an example of a **TERMINAL**, MRR, size 12 font: Able.

### 31.2 Fontdescriptors, and FONTCREATE

For InterlispØD to use a font, it must have a fontdescriptor. A fontdescriptor is a data type in InterlispØD that holds all the information needed in order to use a particular font. When you print out a fontdescriptor, it looks like this:  
[fKTDEIRIPToRjØiØ,Øs~ØØ

Fontdescriptors are created by the function FONTCREATE. For example,

```
(F~TCREATE 'fIEL~1lCA 12 '~o)
```

J:

### 31.2 FOflff

#### FONTDESCRIPTORS, AND FONTCREAIE

creates G fontdescriptor that, when used by other functions, prints in **HELVETIEA BOLD** size 12. Interlisp-D functions that work with fonts Gxpect a fontdescriptor produced with the FONTCREATE function.

The syntax of FONTCREATE is:  
(FØKTCREATE family size face)

Remember from the previous section, face is either a three element list, (weight slope expansion), a three character atom abbreviation, e.g. MRR, or one of the mnemonic abbreviations, e.g. STANDARD.

If FONTCREATE is asked to create a fontdescriptor that already exists, the existing fontdescriptor is simply returned.

### 31.3 Display Fonts - Their files, and how to find them

Display fonts require files that contain the bitmaps used to print each character on the screen. All of these files have the extension .DISPLAYFONT. The file name itself describes the font style and size that uses its bitmaps. For example:

```
~ERK12.DISPUYFRT
```

contains bitmaps for the font family MODERN in size 12 points. Initially, these files are on floppies. The files that are used most often should be copied onto a directory of your hard disk or fileserver. Usually, this directory is called FONTS.

Wherever you put your .DISPLAYFONT files, you should make this one of the values of the variable DISPLAYFONTDIRECTORIES. Its value is a list of directories to search for the bitmap files for

display fonts. Usually, it contains the "FONT" directory where you copied the bitmap files, the device (FLOPPY), and the current connected directory. The current connected directory is specified by the atom NIL. Here is an example value of DISPLAYFONTDIRECTORIES:

```
. - 11
NIL

r~':PI:='pL"'yFnNTDIP,ECTBP,IES

i;!l0o= ' . =PFIL -FnNT~." (D.~fr):!.LIT'.PFIL
fFLnPF")- NIL!i
9!0
```

Figure 31.1. A value for the atom DISPLAYFONTDIRECTORIES. When looking for a .DISPLAYFON file, the system will check the FONT directory on the hard disk, then the top level directory on the hard disk, then the floppy, then the current connected directory.

## FONTS 313

### INTERPRESS FONT5 - THEIR FILES, AND HOW TO FIND THEM

31.4 \_ Interpress \_ Fonts \_ - \_ Their files, \_ and \_ how \_ to \_ find \_ them  
Interpress is the format that is used by Xerox laser printers. These printers normally have a resolution that is much higher than that of the screen: 300 points per inch.

In order to format files appropriately for Output on such a printer, Interlisp must know the actual size for each character that is to be printed. This is done through the use of width files that contain font width information for fonts in Interpress format. Initially, these files (with extension .WD) are on floppies. The files should be copied onto a directory of your hard disk or fileserver.

For Interpress fonts, you should make the location of these files one of the 'values of the variable INTERPRESSFONTDIRECTORIES. Its value is a list of directories to search for the font widths files for Interpress fonts. Here is an example value of INTERPRESSFONTDIRECTORIES:

```
. 11
1'IL

i?IbdTEFPfiET0=:FnN7PIP:EcTnRI—~,~
.i=~~.~
j:~,~
```

Figure 31.2. A value for the atom INTERPRESSFONTDIRECTORIES. When looking for a font widths file for an Interpress font, Interlisp-D will check the hard disk.

## 31.5 Functions for Using Fonts

### 31.5.1 FONTPROP Looking at Font Properties

It is possible to see the properties of a fontdescriptor. This is done with the function FONTPROP. For the following examples, the fontdescriptor used will be the one returned by the function (DEFAULTFONT 'DISPLAY). In other words, the fontdescriptor examined will be the default display font for the system.

There are many properties of a font that might be useful for you.

Some of these are:

**FAffILY** To see the family of a font descriptor, type:  
(FKTPliP (DEFALLTFoIT 'DISPLAY) 'f~ILY)

**SIZE** As above, this is a positive integer that determines the height of the font in printer's points. As an example, the **SIZE** of the current default font is:

310 ~n

## FUNCTIONS FOR USING FONTS

. 11  
NIL

Gi,0(FnNTPROP (DEF~ULTFONT PI~~PLAY)  
'.,0,'IZE\  
is,

Figure 31.3. The value of (he font property **SIZE** of the default font  
**ASCENT** The value of this property is a positive integer, the maximum height of any character in the specified font from the baseline (bottom). The top of the tallest character in the font, then, will be at (BASELINE # ASCE[VT - l). For example, the **ASCENT** of the default font is:

0 1 11  
NIL

A0 4' ., l!0FnNTPROP if Off"" ULTFnNT 0PI~,~PL~","!  
'~e-rENT:!  
q.-

A,5~:

Figure 31.& The value of the font property **ASCENT** of the default font  
**DESCENT** The **DESCENT** is an integer that specifies the maximum number of points that a character in the font descends below the baseline (e.g. letters such as "p" and "g" have tails that descend below the baseline.). The bottom of the lowest character in the font will be at (BASELINE - DESCENT). To see the **DESCENT** of the default font, type:

(FOkTPROP (DefAULTFKT 'DISPUY) 'DESr:—KT)  
**HEIGHT** HE IGHT is equal to t'DESCENT-ASCENT).  
**FACE** The value of this property is a list of the form, (weight slope expansion). These are the weight, slope, and expansion described above. You can see each one separately, also. Use the property that you are interested in, **VEIGHT**, **SLOPE**, or **EXPANSION**, instead of **FACE** as the second argument to **FONTPROP**.

For other font properties, see the Interlisp-D Reference Manual, VolumeIII, Pages 27.27 - 27.28.

### 31.5.2 STRINGWIDTH

It is often useful to see how much space is required to print an expression in a particular font. The function **STRINGWIDTH** does this. For example, type:

(STRIKWIDTH "NV theral0 ('L'NTcREAT— 'Ucli 10 'STAKDARD))  
The number returned IS how many leff to right pixels would be

needed if the string were printed in this font. (Note that this

FIGURE 31.5

## FUNCTIONS FOR USING FONTS

doesn't just work for pixels on the screen, but for all kinds of streams. For more information about streams, see Chapter 30.) Compare the number returned from the example call with the number returned when you change GACHA to TIMESROMAN.

### 31.5.3 DSPFONT - Changing the Font in One Window

The function DSFFONT changes the font in a single window. As an example of its use, first create a window to write in. Type: (SETQ ~.FONT.WINnaN (CttEATE\*))

in the Interlisp-D Executive window. Sweep out the window. To print something in the default font, type: (PRINT 'HELLO N'f.FO\*T.WIN~)

in the Interlisp-D Executive window. Your window, MY.FONT.WINDOW, will look something like this:

HELL

Figure 31.5. HELLO, printed with the default font in MY.FONT.WINDOW. Now change the font in the window. Type: (DSPFONT (FONTCREATE 'HELVETICA 12 'SOLD) \*T.FONT.WINDaN) in the Interlisp-D Executive window. The arguments to FONTCREATE can be changed to create any desired font. Now retype the PRINT statement, and your window will look something like this:

HEL

```
.q.'~;, PSPFnNT (FnNTrRE~TE 'HEL";'ET1L~
1:'øBnLPt
```

```
M'tø.FnNT.vINPnWj
```

```
l:FnNTPE~1'RIpTnfl~#?~.,. 1-ø 14 "4
3~~iPR[NT 'HELLO MY.fnflr.l]INoniff)
HELLO
```

Figure 31.6 The font in MY FONT WINDOW, changed. Notice the font has been changed.

J.

## 31.6 FONTS

### FUNCTIONS FOR USING FONTS

#### 31.5.4 \_ Globally Changing \_ Fonts

There is a library package to globally change the fonts in all the windows. To use it, first load BIG.DCOM. (See Section 8.6, Page 8.4 for how to load a file.)

To change fonts in all windows using the package BIG.DCOM, type

```
(KE*Fo*T <ke~o~>~
```

There are four keywords for size of fonts to specify. They are

HUGE, BIG, STANDARD, and MEDIUM. For example:  
(\*E\*FKT 'BIG)

sets the fonts in ALL the windows to be a larger size. Note: this package changes the fonts everywhere, including the editor window and system merius It is particularly useful to change the size of the font for demos.

### 31.5.5 Personalizing Your Font Profile

Interlisp-D keeps a list of default font specifications. This list is used to set the font in all windows where the font is not specifically set by the user (Section 31.5.3). The value of the atom FONTPROFILE is this list. (See Figure 31.7.)

A FONTPROFILE is a list of font descriptions that certain system functions access when printing output. It contains specifications for big fonts (used when pretty printing a function to type the function name), small fonts (used for printing comments in the editor), and various other fonts.

FOONTS 317

I

### FUNCTIONS FOR USING FONTS

~...~

43-FJtltPRUF[LE

!P P—F"ULTFCIFIT i ,.' ,',cH4 LLT;  
;ø',~LHk aj  
t'TEPMINK'L Sij

'BILPF'INT : ' (H—LIETIL='n' Jo E,PP,;1  
'IH—L':—TIC" L=' BPP,i  
'IJPEPtl in' ~FF)

'LITTLL—FC'NT 3 ;ttEL'ø?ErIC" ,3,'  
iHE - c 1,1p;

i.'BIC-FCNT ~ llnoF - hIIP"i  
'HE 1.= ' BpP.i  
'HEL'ø'ET .- it' epp:'  
'IrtoPEPrl -  
(J\EPFONT 6oLOFEINT  
(C.IMIIENTFANT LITTLL—Fi)r'T  
'L"MOF~"FL1fIT 61 eFol~! T  
i.'='r'3TEMFENT',i

'CLI~T~PFUNT BC'LOF')1'lf  
i.' CH,,N'3'EF')HT

i,'PPETT\1?.Cit1F(ji~T BLILC'FilltIT  
i.'FCPIITL DEf""ULTFEitiT"  
,Fel'JT"ø 6cLDFcllIT;  
t.'FCT'1T3 LITTLEFciføIT;  
'f1tITJ BICF(.r'lT',i

i.'FEINT~ S ',HEL', "ETL,, ' 10 81P'.  
'HEL'y'ETlc,, '3 61A)  
CfillDEPN a 81P,;

t.'FilNTB 6 'HEL';ET16~ 10 8RP',i  
'HEL'. "ET1C~ L'~ BAA"  
llPEPN 3 BAA]

```
Fi)NT7 ? c"'i'.H~ 1:ø"
:e-"Ln~ 1:ø!"

'.TERMI tL'L 1;','.,!,
5a,
```

Figure 31.7. The value of the atom FONTPROFILE

The list is in the form of an association list. The font class names, (e.g. DEFAULTFONT, Or SOLDFONT) are the keywords of the association list. When a number follows the keyword, it is the font number for that font class.

The lists following the font class name or number are the font specifications, in a form that the function FONTCREATE can use. The first font specification list after a keyword is the specification for printing to windows. The list, (GACHA 10), in the figure above is an example of the default specification for the printing to windows. The last two font specification lists are for Press and Interpress file printing, respectively. For more information, see the Interlis~D Reference Manual, Volume 3, Chapter 27.

Now, to change your default font settings, change the value of the variable FONTPROFIL—. Interlis~D has a list of profiles stored as the value of the atom FONTDEFS. Choose the profile to use, then install it as the default FONTPROFILE. Evaluate the atom FONTDEFS and notice that each profile list begins with a keyword. (See Figure 31.8.) This keyword corresponds to the size of the fonts included. BIG, SMALL, and STANDARD are some of the keywords for profiles on this list - SMALL and STANDARD appear in Figure 31.8.

31.8 F0Htt  
1

```
FUNCTIONS FOR USING FONTS
[[SMALL cFONTPRQFILE
(DEFAULTFONT 1 (TERMINAL
8)
```

```
tøUaCHA 8)
'TERmIHAL 8))
```

```
(8OLPFL~NT (Mi!OERtt 3 BRR)
\HELY'FTIL" 6 BRR)
ltl\flEfiH 8 BRfi))
1 LITTLEFCNT ~'ø
(hllCiERN 8 MIR)
lHEL'v'ETIu"" 8 MIR)
iMCiPERN ,q, MIR))
(TIN\FONT a IhllOERN a)
to,'F..H" ~)
hll!nEr.H 6
```

```
iBIrFnNT j (;,nPF~N 1P BFR)
'!HE".LETIcA lG BRf)
hlrPEF;11 16 ~RP)
```

```
iTE.\TFrNT r ',.6LM"~'.lC 13)
'iTIhlE:'Pnn,, "N In)
i.LL~.~:lC lot)
!'TE\TBnLPFnNT
tCL~~CIC 16 Bfifi.,'
~TIME.;RL1MAN
```



1P BfiR)

```
tP:LAc.~,Ir 16 BRR]
[cT~NPARP (FDNTPrnPiLE
(PEF"ULTFnNT 1
```

Figure 31.8. Part of the value of the atom FONTDEFS  
To install a new profile from this list, follow the following example, but insert any keyword for BIG.

To use the profile with the keyword BIG instead of the standard one, evaluate the following expression  
(FOMTSET 'BIG))

Now the fonts are permanently replaced. (That is, until another profile is installed.)

FONIS 319

1

r.

FUNCTIONS FOR USING FONTS

```
[[SMALL cFONTPROFILE
(OEFALLTPONT i (TERMINAL
6)
```

```
*U'acHA 6)
tøTERmIHAL 6))
```

```
(SOLPFL~NT (M'1.OERN 6 BRR)
tHELY'FTIL"" 6 BRR)
Ihll!OER'H qL BRR))
i LITTLEFCNT ~"
(MlcERN 6 MIR)
IHEL'v'ETIu"" 6 MIR)
iMCiOERN ,qø MIR))
(TIN\FONT a IhllOERN a)
U,,F.,H" aj
hll!nEr.N 6
```

```
iBIrFnNT J ';;1nPF~N 1P BFR)
'!HE".!ETICA 16 BRF)
hlrPEF;i1 16 ~fiP) !'
```

```
i TE.\TFrNT r 6L"~.'lc 1'~)
liTIhlE;;pnMN In)
i.LL~.>:Ic In:)
!'TE\TBnLPFnNT
t CLA~C 1 16 Bfifi
jTIME.;ROMAN
1P BfiR)
```

```
\P:LAc.~,Ir 16 BRR]
[<~T~NPARP (FlNTPRnPILE
(PEF"ULTFnNT 1
```

Figure 31.8. Part of the value of the atom FONTDEFS  
To install a new profile from this list, follow the following example, but insert any keyword for BIG.

To use the profile with the keyword BIG instead of the standard one, evaluate the following expression

(FITSET 'BIG))

Now the fonts are permanently replaced. (That is, until another profile is installed.)

FONTS 319

1

----- Next Message -----

Date: 19 Dec 91 16:35 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.163540pst.43009@origami.parc.xerox.com>.:>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11681>;  
 Thu, 19 Dec 1991 16:35:49 PST

Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:35:40 -0800

From: John Sybalsky <sybalsky.PARC@xerox.com>

-----RFC822 headers----->

## 12. YOUR INIT FILE

Interlisp-D has a number of global variables that control the environment of your 1108 or 1186. Global variables make it easy to customize the environment to fit your needs. One way to do this is to develop an "INIT" file. This is a file that is loaded when you log on to your machine. You can use it to set variables, load files, define functions, and any other things that you want to do to make the Interlisp-D environment suit you.

Your init file could be called INIT, INIT.LISP, INIT.USER, or whatever the convention is at your site. There is no default name preferred by the system, it just looks for the files listed in the variable USERGREETFILES, (see below). Check to see what the preference is at your site. Put this file in your directory. Your directory name should be the same as your login name. The INIT file is loaded by the function GREET. GREET is normally run when Interlisp-D is started. If this is not the case at your site, or you want to use the machine and Interlisp-D has already been started, you can run the function GREET yourself. If your user name was, for example, TURING, then you would type:  
 (GREET 'TURIK)

This does a number of things, including undoing any previous greeting operation, loading the site init file, and loading your init file. Where GREET looks for your INIT file depends on the value of the variable USERGREETFILES. The value of this variable is set when the system's SYSOUT file is made, so check its value at your site! For example, its value could be:

- . - 11

NIL

3'USERGREETFILES

iiiFD5hl,(LI5PFILES~ USER ;INIT.LISPJ  
 t1rD5h','.LI5PFILE.>~INIT.LISPJ  
 t',rFLoPPY',INIT.L15—J

i,rosh',øLI5PFILES\ USER .'INIT.U5ERJ  
 ((O.h L FILE.' .INIT.U.'ER'øj

```
i(D. . FIL SER INIT;
i(FLUPP';j I
```

Figure 12.1. ApcsstblevalueofUSERGREETFILES.

In each place you see, "> USER >", the argument passed to GREET is substituted into the path. This is your login name if you are just starting Interlisp-D. For example, the first value in the list would have the system check to see whether there was a file, [DSX]<L'SPFILES>TURING>INIT.LISP. No error is generated if you do not have an INIT file, and none of the files in USERGREETFILES are found.

YOUR INIT FILE 12.1

## MAKING AN INIT FILE

### 12.1 Making an Init File

As described in Section 11.5, Page 11.7, each Interlisp-D program file has a global variable associated with it, whose name is formed by appending "COMS" to the end of the root filename. For any of the standard INIT file names, the variable INITCOMS is used. To set up an init file, begin by editing this variable. First, type:

```
(SETQ INITCOMS '(VAN))
Now, to edit the variable, type:
(lz:edit 'INITCOMS)
```

A DEdit window will appear. This DEdit window is the same as the one called with the function OF, and described in Section 11.3, Page 11.4. This chapter will assume that you know how to use the structure editor, DEdit.

The CONS variable is a list of lists. The first atom in each internal list specifies for the file package what types of items are in the list, and what it is to do with them. This section will deal with three types of lists: VARS, FILES, and P. Please read about others in the Interlisp-D Reference Manual, Volume II, Chapter 17. The list that begins with "VAR5" allows you to set the values of variables. For example, one global variable is called DEditLinger. Its default value is T, and means that the DEdit window won't close after you exit DEdit. If it is set to NIL, then the DEdit window will be closed when you exit DEdit. To set it to NIL in your INIT file, edit the VARS list so that it looks like this:

```
0 . . 1 1 0 1
((,4R. 'ioEdirLinger NIL))
B~,are
G~lete
Replace
'yvitch
()
```

```
(out
Undo
Find
5'rtap
Rcprint
Edit
```

```
EditCam
Break
Eva
```

Exit

Figure 12J. Setting the variable `DEFINTEGR` in `INITCONS`. Notice that inside the vars list, there is yet another list. The first item in the list is the name of the variable. It is bound to the value of the second item. There are many other variables that you can set by adding them to the VARS list. Some of these variables are described in Chapter 43, and many others can be found in the Interlis-D Reference Manual.

If you want to automatically load files, that can be done in your init file also. For example, if you always want to load the Library file `SPY.DCOM`, you can load it by editing the `INITCOMS` variable to list the appropriate file in the list starting with `FILES`:

## 12.1 YOUR INIT FILE

### MAKING AN INIT FILE

```
((VAR (DEFINTEGR NIL)) After
 (FILES SPY) Betone
 Delete
 Replace
 Switch
```

```
Quit
Undo
Find
Swap
Reprint
Edit
```

```
EddCom
Break
Eval
Exit
```

Figure 12.3. `INITCOMS` changed to load the file `SPY.DCOM`. Other files can also be added by simply adding their names to this `FILES` list.

Another list that can appear in a `COMS` list begins with "P". This list contains Interlis-D expressions that are evaluated when the file is loaded. Do not put `DEFINEQ` expressions in this list.

Define the function in the environment, and then save it on the file in the usual way (see Section 11.6, Page 11.7).

One type of expression you might want to see here, however, is a `FONTCREATE` function (see Section 31.2, Page 31.2). For example, if you want to use a Helvetica 12 BOLD font, and there is not a fontdescriptor for it normally in your environment, the appropriate call to `FONTCREATE` should be in the 'P' list. The `INITCOMS` would look like this:

```
((VAR (DEFINTEGR NIL)) After
 (FILES SPY) Betone
 (~ FONTCREATE (QUOTE Helvetica
 Helvetica 12 BOLD) Repace
 ~vyitch
```

```
1-
 (~juate _ SOL") _ .1)) ('out
 Undo
 Find
```

Swap  
Reprint  
Edit

EdiKom  
Break  
Eva  
Exit

Figure 12.4. ItulTcOfI5editedtoincludeacalltofOffTCfIEATE. The form will be evaluated when theINIT file is loaded.

To quit, exit from DEdit in the usual way. When you run the function NAKEFILES (See Section 11.6, Page 11.7.), be sure that you are connected to the directory (see Section 8.7, Page 8.4) where the INIT file should appear. Now when GREET is run, your init file will be loaded.

YOUR INIT FILE 123

----- Next Message -----

Date: 19 Dec 91 16:48 PST  
From: sybalsky:PARC:Xerox  
To: sybalsky  
Message-ID: <<91Dec19.164812pst.43009@origami.parc.xerox.com>.?::>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11686>;  
Thu, 19 Dec 1991 16:48:22 PST  
Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:48:12 -0800  
From: John Sybalsky <sybalsky.PARC@xerox.com>  
-----RFC822 headers----->

r 33. MASTERSCOPE

Masterscope is a tool that allows you to quickly examine the structure of complex programs. As your programs enlarge, you may forget what variables are global, what functions call other functions, and so forth. Masterscope keeps track of this for you.

Suppose that JVTO is the name of a file that contains many of the functions involved in a complex system and that LINTRANS is the file containing the remaining functions. The first step is to ask Masterscope to analyze these files. These files must be loaded. All Masterscope queries and commands begin with a period followed by a space, as in  
Ø AliLYZE FKS a Jvro

The ANALYZE process takes a while, so the System prints a period on the screen for each function it has analyzed. (See Figure 33.1)

82&. ANALYZE FNS ON 3VTO  
. d.,ne  
D3~. aNALY?E FNS ON LIH1R'N~  
. 1a,lA.l

Figure 33.1. The Interlisp-D Executive Window after analyzing the files. If you are not quite sure what functions were just analyzed, type the file's CONS variable (See Section 11.5, Page 11.7.) into the Interlisp-D Executive Window. The names of the functions

stored on the file will be a part of the value of this variable.

A variety of commands are now possible, all referring to individual functions within the analyzed files. Substantial variation in exact wording is permitted. Some commands are:

```
ø SHoN PATHS FRDN ANY T0 ANY
ø EDIT WERE ANY CALLS functionname
ø EDIT WERE ANY USES variablename
ø Wo CALLS WDN
```

```
ø Wo CALLS functionname
ø BY WoN IS functionname CALLED
ø WD USES variablename AS FIELD
```

Note that the function is being called to invoke each command. Refer to the *Interlis-D Reference Manual* for commands not listed here.

Figure 33.2 shows the Interlis~D Executive Window after the commands `ø wno CALLS GobbleDunp` and `ø vffo DOES JVLinScan CALL`.

```
MASTb'R'j~OPE 331
```

```
MASTEH,COPE
```

```
NIL
```

```
7,,'. 1,,.lillj O~LL;==: ,1)~8 iD.B~imp
```

```
("c.h.~t;:r~i'TJ .J;/j~J,J .J'.t'r'Jet'TJ J;,'~~ 1Tij Gi>"ri'.p~" ,,)bbl,,Ffu:h ,,"jbb1~'Srririll
l/dump Fiji
```

```
'...9j', "Ho clclE.. .J"i'L i '-. r, 1""LL
(Liri.'ci-ri 1'ø.'Cfr.3b1A 3 -h1~J
'9'A
```

Figure 33.2. Sample Masterscope Output

33.t The SHOW DATA command and GRAPNER  
When the library package GRAPHER is loaded, (to load this package, type (FILESLOAD GRAPHER).) Masterscope's SHOWPATHS command is modified. The command will be changed to generate a tree structure showing how the program's functions interact instead of a tabular printout into the Interlis~D Executive window. For example, typing:  
`ø ~ PATHS FW Proce:s—E.`  
produced the display shown in Figure 33.3.

```
.GtB.,31nT~, T L:~n;:lLl:'ø.Utø:n:P,.=
~""-,Jt",,,r,.pj
```

```
r~infr.:p it'~ø;r.!rPr:p
r.>~>(Li;'7Uiø~p..
..~..JtL,;.l. Of'.JtLl;l.
'r:L,st lET _ p-:
—:J8:~inEnJ ,*.l.Tø.'r:.
```

```
ø .Err.'.r Pfl!l Pr'ni.~n,;
:p~1y
~Int~nlno
```

Figure 33.3. SHOW PATHS Display Example

All the functions in the display are part of this analyzed file or a previously analyzed file. Boxed functions indicate that the function name has been duplicated in another place on the display.

Selecting any function name on the display will pretty print the function in a window. (See Figure 33.4.)

ij.J MAsTERscopa

#### THE SHOW DATA COMMAND AND GRAPHER

```
~&lLir1wilhS hØfi
.Ø~TlØo1nTwTr1no~
~9i"~
```

```
Ø 1n,fl~ ~i~or9~~~
Ø ~~&tl1r?inith. "(s
.
```

```
to~~tLisi_ ~otLirt
~Ø~.~.~rØ,Ø; _____
Pw:ØLlrt ~(LØTTØ.'
```

```
Ø .'.d.~~~;Ø ~qLT~' f
```

```
Ø ØØ..'PøintError PTL.I Frint~ninØi
Ø
```

```
Ø .- upv
~inlWr,r.
```

```
[LAnaPA ~propnaaØØ'i (' cdttd: ØØ16ØMAAØØ3Ø' L'6"dØØ
Ø ØleCAN~9SCorProp prcpneae (suR 1012C8L'Øk])
```

Figutt 33.4. Browser Printout Example.

Selecting it again with the leff mOuse button will produce a dexcription of the function's role in the overall system (See Figure 33.4)

```
~r.Ø;l.1,'?U1ØnØ;Øf.:Ø
Øt1BfgØinTW:Øl.riny~ _____
```

```
,p'~Ø~ 1"~Ø:t,-'ØØØ.Pr'~
.~c.3v.LiØiniih~t
```

```
Proc&;:Eli<. _____
Ø .. . Por;Ø~Ør' Ø~:l=T7ØØØ.
```

```
Ønf&rØ.ØØr ~T=i Prir,t.~nir,Ø
Ø ØPØntWr,1fl4Ø
```

```
Ø GerryProp i,, -
Ø L-Qll:! inetAnC .rorPrtihØ
1n—N—l,~ lrirFiØJ.,rningØ
Din" 'pe~ØbØØ'c8cJ1nT;.,='Øtr1n0.
i=~rMLØ,Ø 1rØ'"ØØØT=~',FØr=CeØ'ØØEtld
Ø u~' f.-ccØ TO cocl
```

FlØUrf 33.5. Browser Description Example.

33.2 Databasefns: Automatic Construction and Upkeep of a Masterscope Database

DataBaseFns is a separate library package that allows you to automatically construct and maintain Masterscope databases of your files. The package is contained in the DATABASEFNS.DCOM file.

When DATABASEFNS.DCOM is loaded, a Masterscope database will be automatically maintained for every file whose DATABASE

MAST'RS~OPE 333

DATABASEFNS: \_ AUTOMATIC CONSTRUCTION AND \_ UPKEEP OF A MASTERSCOPE \_ DATABASE

property has the value YES. If this property's value is not set, you will be asked when you save the file "Do you want a Masterscope Database for this file?". Saying YES enables the DabaBaseFns to construct a Masterscope database of the file you are saving. Each time the function \*AKEFILE is used on a file whose DATABASE property has a value YES, Masterscope will analyze your file and update its own database. Each file's masterscop database is kept in a separate file whose name has the form FILE.DATABASE. Whenever you load a file with a YES value for its DATABASE property, you will be asked whether you also want the database file loaded.

### 33.4 N~TERSCOPE

1

----- Next Message -----

Date: 19 Dec 91 16:50 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.165058pst.43009@origami.parc.xerox.com>.:>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11688>; Thu, 19 Dec 1991 16:51:02 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:50:58 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 -----RFC822 headers----->

rø 34.WHERE DOES ALL THE TIME GO?

sPY

SPY is an Interlisp-D library package that shows you where you spend your time when you run your system. It is easy to learn, and very useful when trying to make programs run faster.

#### 34.1 Now to use Spy with the SPY Window

The function SPY. BUTTON brings up a small window which you will be prompted to position. Using the mouse buttons in this window controls the action of the SPY program. When you are not using SPY, the window appears as in Figure 34.1.

Figure 34.1. The SPY window when SPY is not being rnsed. To use SPY, click either the left or middle mouse button with the mouse cursor in the SPY window. The window will appear as in Figure 34.2, and means that SPY is accumulating data about your program.



Figure 34.2. The SPY window when SPY is being used

To turn off SPY after the program has run, again click a mouse button in the SPY window. The eye closes, and you are asked to position another window. This window contains SPY's results. An example of result window is shown in Figure 34.3.

```
WHERE DOES ALL THE TIME GO? SPY 341
1
```

HOW TO USE SPY WITH THE SPY WINDOW

- TREE.

```
1 _ '3~'"H[P _ J~. _ [WIT. _ IN~&F.-1!~
```

```
17 _ '..TirtP. PplJl'.E,-'..'
- a ~i~::~~pT~—&-
```

```
. REPE,,TE&L'.EV~rn EJ~rn 01 EF.—UFE
```

```
7 _ ---RR.. h.JPillU~. _ fPILE-0 -. 4 f, IPP,9R.h 'F..n 4
```

Figure 34.3. The window produced after running SPY

This window is scrollable in two directions—horizontally, and vertically. This is useful, since the whole tree does not fit in the window. If a part that you want to see is not shown, then you can scroll the window to show the part you want to see.

### 34.2 How to use SPY from the Lisp Top Level

SPY can also be run while a specific function or system is being used. To do this, type the function WITH. SPY:  
(WITH.SPY form)

The expression used for form should be the call to begin running the function or system that SPY is to watch. If you watch the SPY window, the eye will blink! To see your results, run the function SPY. TREE. To do this, type:

```
(SPY.TREE)
```

The results of the last running of SPY will be displayed. If you do this, and SPY.TREE returns (no SPY samples have been gathered), your function ran too fast for SPY to follow.

### 34.3 Interpreting SPY's Results

Each node in the tree is a box that contains, first, the percentage of time spent running that particular function, and second, the function name. There are two modes that can be used to display this tree.

The default mode is cumulative. In this mode, each percentage is the amount of time that function spent on top of the stack, plus the amount of time spent by the functions it calls.

The second mode is individual. To change the mode to individual, point to the title bar of the window, and press the middle mouse button. Choose Individual from the menu that appears. In this mode, the percentage shown is the amount of time that the function spent on the top of the stack.

```
34.2 WHERE DOES ALL THE TIME GO? SPY
```

```
1
```

INTERPREn~ SPY'S RESuLtt

----- Next Message -----

Date: 19 Dec 91 16:40 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.164041pst.43009@origami.parc.xerox.com>.:>

<----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11685>;  
 Thu, 19 Dec 1991 16:40:51 PST

Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:40:41 -0800

From: John Sybalsky <sybalsky.PARC@xerox.com>

-----RFC822 headers----->

## 32. THE INSPECTOR

The Inspector is a window-oriented tool designed to examine data structures. Because Interlisp-D is such a powerful programming environment, many types of data structures would be difficult to see in any other way.

### 32.1 Calling the Inspector

Take as an example an object defined through a sequence of pointers (i.e. a bitmap on the property list of a window on the property list of an atom in a program.)

To inspect an object named NAME, type:  
 (INSPECT '~)

If NAME has many possible interpretations, an option menu will appear. For example, in Interlisp-D, a litatom can refer to both an atom and a function. For example, if NAME was a record, had a function definition, and had properties on its property list, then the menu would appear as in Figure 32.1.

PRG'PS  
 FkIS

FIELD;=~

Figure 32.1. Option Window For Inspection of NAME

If NAME were a list, then the option menu shown in Figure 32.2 would appear. The options include:

Ø calling the display editor on the list;

Ø calling the ~ editor (the "Typing Shortcuts", Chapter 6);

Ø seeing the list's elements in a display window. If you choose this option, each element in the list will appear in the right column of the Inspector window. The left column of the Inspector window will be made up of numbers. (See Figure 32.3.)

Ø inspecting the list as a record type (this last option would produce a menu of known record types). If you choose a record type, the items in the list will appear in the right column of the Inspector window. The left column of the Inspector window will be made up of the field names of the record.

P~rI~.lErtr  
 Tr:rE'1ir.

Inip~rr

A~are'iJrd

Figure 32.2. Option Window For Inspection of Lirt

## THE INSPECTOR 321

### USING THE INSPECTOR

#### 32.2 Using the Inspector

If you choose to display your data structure in an edit window, simply edit the structure and exit in the normal manner when done. If you choose to display the data structure in an inspect window, then follow these instructions:

Ø To select an item, point the mouse cursor at it and press the left mouse button.

Ø Items in the right column of an Inspector window can themselves be inspected. To do this, choose the item, and press the center mouse button.

Ø Items in the right column of an Inspector window can be changed. To do this, choose the corresponding item in the left column, and press the center mouse button. You will be prompted for the new value, and the item will be changed. The sequence of steps is shown in Figure 32.3.

Ø .

```
1 INPEu'T-ME-TOOI
1ie-,PErT-hl—TQi32
```

a IN.u'FErT-11E-TQO3 The item in the left column is selected, and the middle mouse button pressed. Select the SET option from the menu that pops up.

The ev..pre:1Un re3J will be E  
'/~LuQred.

cHaflGE&-'.:~"LIJ4

```
1]N.=~Pfi=.T-rrtE-Tc~i
2 1H".~pEcT-ttE-TI:i12
```

a Il You will then be prompted for the new value. Type it in.

Ø6

```
1 [flPEQT-ME-TOOI
2 [Y.~PECT-1'1E-TOfl2
```

a CH~Pl,'ED-'.;~LUE The item in the right column is updated to the value of what you typed in.

Figure 32.3. The sequence of steps involved in changing a value in the right column of an Inspector window.

#### 32.3 Inspector Example

This example will use ideas discussed in Section 37.1. An example, ANIMALGItAPH, is created in that section. You do not need to know the details of how it was created, but the structure

will be examined in this chapter.  
If you type

```
(IKSPECT II~.6liPN)
```

and then choose the Inspect option from the menu, a display appears as shown in Figure 32.4. ANIMAL.G~PH is being

J

```
33.J TkeINSPECT~
```

```
lff5PE~0R EXAMPLE
```

inspected as a list. Note the numbers in the left column of the inspector window.

```
1 i't'fl.~H ~ NIL NIL --j 'BIRD .~ NIL NIL
Ø T
.i, NIL
4 NIL
5 NIL
6 NIL
? NIL
,qØ NIL
9. NIL
1A. NIL
11 NIL
1~" NIL
```

Figure 32.4. Inspector Window For ANIMAL GRAPH, inspected as a list.  
If you choose the "As A Record" option, and choose "GRAPH" from the menu that appears, the inspector window looks like Figure 32.5. Note the fieldnames in the left column of the inspector window.

```
UP"PH.CH"NCEL"eELFfl NIL
CR"PH. INVEP.TL~BELFN NIL
CR"PH. IFlvEp.TBCiROERFN NIL
CR"PH.FONTch"NOEFN NIL
bRaPH.&ELETELINKFN NIL
CRaPHØ~D&LINKFN NIL
URAPH.c—LETENC~UEFN HIL
bRAPH..oo&NUGEFN NIL
oRoPH.Mo□ENUoEFN NIL
DIREcTEDfLG NIL
o"IDE~FLo T
```

```
C.RuPHNi:DE.~ (i.'fl:H & NIL NIL --! 'BIPP & NIL GIL
```

Figure 32.5. Inspector Window For ANIMAL.GRAPH, inspected as an instance of a "GRAPH" record.

The remaining examples will use ANIMAL.GRAPH inspected as a list. When the first item in the Inspector window is chosen with the left mouse button, the Inspector window looks like Figure 32.6.

```
1 ' _ Ø1
T
3 NIL
4 NIL
5 NIL
r~ NIL
```

```

NIL
NIL
9 NIL
1H NIL
11 NIL
1- NIL

```

Figure 32.6. Inspector Window For ANIMAL.GRAPH With First Element Selected  
When you use the middle mouse button to inspect the selected list element, the display looks like Figure 32.7.

THE INSPECTOR 32 j

INSPECTOR EXAMPLE

```

1 01
T

3 NIL =
4 PIIL

5 NIL 1 iFifl 1.19:' 44) PIIL NIL HIL --!
'BIRD (.192 29) NIL PIIL NIL --
b' NIL 3 (CAT (.is ,J NIL NIL NIL

PIIL j. (&UU i"1;39 7) PIIL PJIL NIL
NIL ~ ((rh,,"trtffi,,"L GJU c~T) 199 14j fiL J.IIL
9 PIIL 6 ((,,"PIIMAL ; BIRD FI.Jh) .'~ C9. IIL
19 PIIL
11 NIL
1'.' NIL

```

Figure 32.7. Inspector Window For ANIMAL.GRAPH and For the First Element of ANIMAL0GRAPH

How you can see that 5ix items make up the list, and you can further choose to inspect one of these items. Notice that this is also inspected as a list. As usual, it could also have been inspected as a record.

Select item 5 - MAMMAL DOG CAT - with the left mouse button. Press the middle mouse button. Choose "Inspect" to inspect your choice as a list. The Inspector now displays the values of the structure that makes up MAMMAL DOG CAT. (See Figure 32.8.)

```

1 (h1~~MMkL GJ, lIT)
2 ilvjy' lJ)
NIL
4 NIL
5 NIL
6 45
7

is

o i',Do': ClIT',i

l) (c"NIMIL .~ BIRP FI."3Hj
iR, (Fi=1NTCLn",~r'j7Rli?e..764
ii hllIPtMIL
12 NIL

```

Figure 32.8. Inspector Window for Element 5 From Figure 32.7 That Begins ((MAMMAL DOG CAT).

## 32.A THE INSPECTOR

----- Next Message -----

Date: 19 Dec 91 16:54 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.165444pst.43009@origami.parc.xerox.com>.:>

<-----RFC822 headers-----  
 Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11690>;  
 Thu, 19 Dec 1991 16:54:53 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:54:44 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 -----RFC822 headers----->

.....

rø 34.WMERE DOES ALL THE TiME GO?

sPY

SPY is an InterlispøD library package that shows you where you spend your time when you run your system. It is easy to learn, and very useful when trying to make programs run faster.

#### 34.1 How to use Spy with the SPY Window

The function SPY. BUTTON brings up a small window which you will be prompted to position. Using the mouse buttons in this window controls the action of the SPY program. When you are not using SPY, the window appears as in Figure 34.1.

Figure 34.1. The SPY window when SPY is not being used.  
 To use SPY, click either the left or middle mouse button with the mouse cursor in the SPY window. The window will appear as in Figure 34.2, and means that SPY is accumulating data about your program.

sPY

Figure 34.2. The SPY window when SPY is being used

To turn off SPY after the program has run, again click a mouse button in the SPY window. The eye closes, and you are asked to position another window. This window contains SPY's results. An example of result window is shown in Figure 34.3.

WHERE DOES ALL THE TIME Go' SPY 341  
 l

How TO USE SPY WITH THE SPY WINDOW

rp.i)rE?  
 ~L,;,.\*~.

IrtP.rpji=.E;";. .

øU TI~REhpYfi—&.

J!!li .EV~fi)f. '. ø 01 FEPEA~OL.EU~rn -'1 EJ~J .l ER.GURE

```
7 _ 000.BN.0.F i;f;i0.iU~. _ Fpi'cf00:011 .. j IPP,fl~.0hJri.0ii.iN
4
```

Figure 34.3. The window produced after running SPY

This window is scrollable in two directions, horizontally, and vertically. This is useful, since the whole tree does not fit in the window. If a part that you want to see is not shown, then you can scroll the window to show the part you want to see.

### 34.2 How to use SPY from the Lisp Top Level

SPY can also be run while a specific function or system is being used. To do this, type the function WITH SPY:  
(WITH.sPY form)

The expression used for form should be the call to begin running the function or system that SPY is to watch. If you watch the SPY window, the eye will blink! To see your results, run the function SPY.TREE. To do this, type:

```
(SPY.TREE)
```

The results of the last running of SPY will be displayed. If you do this, and SPY.TREE returns (no SPY samples have been gathered), your function ran too fast for SPY to follow.

### 34.3 Interpreting SPY's Results

Each node in the tree is a box that contains, first, the percentage of time spent running that particular function, and second, the function name. There are two modes that can be used to display this tree.

The default mode is cumulative. In this mode, each percentage is the amount of time that function spent on top of the stack, plus the amount of time spent by the functions it calls.

The second mode is individual. To change the mode to individual, point to the title bar of the window, and press the middle mouse button. Choose Individual from the menu that appears. In this mode, the percentage shown is the amount of time that the function spent on the top of the stack.

### 34.2 WHERE DOES ALL THE TIME GO? SPY

1

#### 1 INTERPRETING SPY'S RESULTS

To look at a single branch of the tree, point with the mouse cursor at one of the nodes of the tree, and press the right mouse button. From the menu that appears, choose the option SubTree. Another SPY window will appear, with just this branch of the tree in it.

Another way to focus within the tree is to remove branches from the tree. To do this, point to the node at the top of the branch you would like to delete. Press the middle mouse button, and choose Delete from the menu that appears.

There are also different amounts of "merging" of functions that can be done in the window. A function can be called by another function more than once. The amount of merging determines where the subfunction, and the functions that it calls, appear in the tree, and how often. (For a detailed explanation of merging, see the Lisp Library Packages Manual.)

### WHERE DOES ALL THE TIME GO? SPY 343

----- Next Message -----

Date: 19 Dec 91 16:59 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.165929pst.43009@origami.parc.xerox.com>.:>

<-----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11691>;  
 Thu, 19 Dec 1991 16:59:33 PST

Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 16:59:29 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>

-----RFC822 headers----->

till.. 36. FREE MENUS

Free Menu is a library package that is even more flexible than the regular menu package. It allows you to create menus with different types of items in them, and will format them as you would like. Free menus are particularly useful when you want a "fill in the form" type interaction with the user.

Each menu item is described with a list of properties and values. The following example will give you an idea of the structure of the description list, and some of your options. The most commonly used properties, and each type of menu item will be described in Section 36.2 and Section 36.3.

### 36.1 An Example Free Menu

Free menus can be created and formatted automatically! It is done with the function FN.FORMATMENU This function takes one argument, a description of the menu. The description is a list of lists; each internal list describes one row of the free menu. A free menu row can have more than one item in it, so there are really lists of lists of lists! It really isn't hard, though, as you can see from the following example:

```
(SETQ Ex~1e*anu
(F*.FORliT*EMu
```

```
'((TYPE TITLE LABEL TitlesDonothing)
TYPE 3STATE LABEL Ex~1e3State))
```

```
(TYPE EDITSTART LABEL PressToStartEd;ting
ITEMS (EDITE*))
```

```
(TYPE EDIT ID EDITEN LABEL øø))
(*IKDDMPRDPS TITLE øEx~1e Dris Nothing))))
```

The first row has 2 items in it; one is a TITLE, and the second is a 3STATE item. The second row also has 2 items. The second, the EDIT item, is invisible, because its label is an empty string. The caret will appear for editing, however, if the EDITSTART item is chosen. Windowprops can appear as part of the description of the menu, because a menu is, after all, just a special window. You can specify not only the title with WINDOWPROPS, but also the position of the free menu, using the "left" and "bottom" properties, and the width of the border in pixels, with the "border" property. Evaluating this expression will return a window. You can see the menu by using the function OPENW. The following example illustrates this:



FREE MENUS 361

1

AN EXAMPLE FREE MENU

6i,'~T~ E;mD.1c1d~nij

.;F(,7,fJp,[4,,~]~fJ.J\J,,ø. T' Gf TITLE LBEL T ir1~,flN~rr T  
 .T"rE =-ø'T."TE L"bEL E:..Jm"1c5tJcs'!'.  
 . 'FE =,IT:..THF:T

LEL =r~'=Tu""r~t'tEditing  
 lTEfl= cOlTEN'

T"E 'IT ID EDITE m L~8EL ,  
 ""..ililci=,.."=cipT=..

TITLE ' ,,,1";c Din' 1'luthlnj.'?

.TT9'i i)pf)liff molMertiJ'i  
 f.hi1ltlDu',V'r#' j64

Figure 36.1. An example free menu

The next example shows you what the menu looks like after the EDITSTART item, PressToStartEditing, has been chosen.

T,r f~"=.Oi=i1'1,=irhin3 E';,mp1~,='.=r.,r~  
 P~'~="""TJT.,rTEJ1r1r1.j A

Figure 36.2. Free menu after the EDITSTART item has been chosen  
 The following example shows the menu with the 3STATE item in its T state, with the item highlighted (In the previous bitmaps, it was in its neutral state.)

.  
 ø c l l 1

.1-=':OiJ-tljrhini=!  
 ,:'T'='Ot.;rrE'lririj,

Figure 36.3. Free menu with the 3STATE item in its T state  
 Finally, Figure 36.4 shows the 3STATE item in its NIL state, with a diagonal line through the item

T1r le."OcNorhing E...:r'~ \_ 1 \_ = \_ ø'...'i.~  
 Rrn. ;", T,St.arrEdir,iri,

.....

Figure 36.4. Free menu with the 3STATE item in its NIL state  
 If you would like to specify the layout yourself, you can do that too. See the Lisp Library Packages Manual for more information.

### 36.2 Parts of a Free Menu Item

There are 8 different types of items that you can use in a free menu. No matter what type, the menu item is easily described by a list of properties, and values. Some of the properties you will use most often are:

### 36.2 FREE MENUS

1

### PARTS OF A FREE MENU ITEM

**LABEL** Required for every type of menu item. It is the atom, string, or bitmap that appears as a menu selection.

**TYPE** One of eight types of menu items. Each of these are described below.

**MESSAGE** The message that will appear in the prompt window if a mouse button is held down over the item.

**ID** An item's unique identifier. An ID is needed for certain types of menu items.

**ITEMS** Used to list a series of choices for an NCHOOSE item, and to list the ID's of the editable items for an EDITSTART item.

**SELECTEDFN** The name of the function to be called if the item is chosen

### 36.3 Types of Free Menu Items

Each type of menu item is described in the following list, including an example description list for each one.

**Momentary** This is the familiar sort of menu item. When it is selected, the function stored with it is called. A description for the function that creates and formats the menu looks like this:

```
(TYPE WEKTARY
 LABEL Blink-K-Rin9
```

```
*ES~6E øBlinks the screen and rings bellsø
s—LEcTEDFK RIKBELLS)
```

**TOGGL—** This menu item has two states, T and NIL. The default state is NIL, but choosing the item toggles its state. The following is an example description list, without code for the SELECTEDFN function, for this type of item:

```
(TYPE T~6LE
 LABEL hi~isab1e
```

```
sELEcTEDFN changeIl*State)
```

**3STATE** This type of menu item has 3 states, NUIETRAL, T, AND NIL. Neutral is the default state. T is shown by highlighting the item, and NIL is shown with diagonal lines. The following is an example description list, without code for the SELECTEDFN function, for this type of item:

```
(TYPE 3STATE
```

```
LABEL correctprograøAllofflospelling
sELEcTEDFli ToggleSpellingcorrection)
```

**TITLE** This menu item appears on the menu as dummy text. It does nothing when chosen. An example of its description:

```
(TYPE TITLE LABEL øChoices:")
```

**NWAY** A group of items, nnly one of which can be chosen at a time. The items in the NWAøY group should all have an ID field, and the ID's should be the same. For exan1Fle, to set up a menu that would allow the user to chose betvvvei Helvetica, Gacha, Modern, and Classic fonts, the descriptions might look like this (Once again, without the code for the SELECTEDFN):

```
(TYPE lAY ID F~Tc~Ic'
 LABEL blvetica
sELEcTEDFN changeFont)
```

FREE MENUS 36)

I

## TYPES OF FREE MENU ITEMS

(TYPE NWAY ID FOQTCKICE  
 LABEL Gacha  
 SELECTEDF)

(TYPE IAY ID F05TliCriC0ha,~n8efont)  
 LABEL Modern

SELECTEDFli Chan2eFont)  
 (TYPE KAY ID fONTCHOIC  
 LABEL Classic

SELECTEDFN Changefont)

NCHOOSE This type of menu item is like NWAY except that the choices are given to the user in a submenu. The list to specify an NCHOOSE menu item that is analogous to the NWAY item above might look like this:

(TYPE MC~SF  
 LABEL FontChoices

ITEMS Helvetica Gacha Modern Classic)  
 SELECT DfK Changefont)

EDITSTART When this type of menu item is chosen, it activates another type of item, an EDIT item. The EDIT item or items associated with an EDITSTART item have their ID's listed on the EDITSTART's ITEMS property. An example description list is:

(TYPE EDITSTART LABEL øFunction to add? ITEMS (Fn))

EDIT This type of menu item can actually be edited by you. It is often associated with an EDITSTART item (see above), but the caret that prompts for input will also appear if the item itself is chosen. An EDIT item follows the same editing conventions as editing in Interlisp-D Executive window:

Add Characters by typing them at the caret.

Move the caret by pointing the mouse at the new position, and clicking the left button.

Delete Characters from the caret to the mouse by pressing the right button of the mouse. Delete a character behind the caret by pressing the back space key.

Stop editing by typing a carriage return, a Control-X, or by choosing another item from the menu.

An example description list for this type of item is:

(TYPE EDIT ID Fn LABEL øø)

## 36.4 FREENEMus

1

----- Next Message -----

Date: 19 Dec 91 17:05 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky

Message-ID: <<91Dec19.170545pst.43009@origami.parc.xerox.com>.:>

<-----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11694>;

Thu, 19 Dec 1991 17:05:54 PST

Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:05:45 -0800

From: John Sybalsky <sybalsky.PARC@xerox.com>

-----RFC822 headers----->

### 37. THEGRAPHER

#### 37.1 Say it with Graphs

Grapher is a collection of functions for creating and displaying graphs, networks of nodes and links. Grapher also allows you to associate program behavior with mouse selection of graph nodes. To load this package, type  
(FILES~ GliPHER)

Figure 37.1 shows a simple graph.

```
i 'iLk w.F."PH 'N M'L.L;R"PH 'NlM'ØL r;P"Pffø'
,(h,,lINGUY!;tw',1513.y1
14'
```

-FIH

. NIM"L, BIRO

Figure 37.1. A Simple Graph

In Figure 37.1 there are six nodes (ANIMAL, MAMMAL, DOG, LAT, FISH, and BIRD) connected by five links. A GRAPH is a record containing several fields. Perhaps the most important field is GRAPHNOD—S - which is itself a list of GRAPHNODE records. Figure 37.2 illustrates these data structures. The window on top contains the fields from the simple graph. The window on the bottom an inspection of the node, DOG.

#### THEGRAPHER 371

##### SAY ITWITH GRAPHS

```
i9'1, I ET Ø'NI1'1,,L.CR~PH'.i
Ilvl,l = '#'=9,1j~'j',3
```

```
GPPH.cr"iLEL,,Ø'BELFN 'IL
Ø 1'R"pH. lli'Ø!ERTLBELFN 1.IIL
Ø H. IflØØ.ØØERTBDPDEPFN till
H.FGtTi'.HNoEFPl 1ØIL
```

```
, rPØ'PH,t.IL/ElØllDEFbl IL
. OIRECTECFLi, (ilL
```

```
Ø rp..'Pflbll) ~ I.F = , till III '.ØB.IP.D NIL ff IL
```

```
. NOOEBPOER 'ilL
tiODEL,, 'BEL loo
, , 'tODEFONT ~FOliT '
. .OtlffO&EO- It'lL -
```

```
. t,iODE'.~,lOTH '.4,
, liUOEL6EL.'-H~OE ØØIIL
, NODELfiELBlTirt,iØP ØIIL
, I,iUDEPUITICII.l in
NODE ID 300
```

Figure 37.2. Inspefling a Graph and a Node

The GRAPHNODE data structure is described by its text (NODEID), what goes into it (FROMNODES), what leaves it (TONODES), and other fields that specify its looks. The basic model of graph building is to create a bunch of nodes, then layout the nodes into a graph, and finally display the resultant graph. This can be done in a number of ways. One is to use the function NODECREATE to create the nodes, LAYOUTGRAPH to lay out the nodes, and SHOWGRAPH to display the graph. The primer shows you two simpler ways, but please see the Library Packages Manual for more information about these other functions. The primer's first method is to use SHOWGRAPH to display a graph with no nodes or links, then interactively add them. The second is to use the function LAYOUT5EXPR, which does the appropriate NODECREATES and a LAYOUTGRAPH, with a list. The function SHOWGRAPH displays graphs and allows you to edit them. The syntax of SHOWGRAPH is

```
(~liPH graph window leftbuttonfn middlebuttonfn
topjustifflg alloweditflg copybuttoneventfn)
Obviously the graph structure is very complex. Here's the easiest
way to create a graph.
~.6liPN III)
```

```
IS5~liPN P.6liPH 05Y Sraph0 KIL NIL NIL T)
```

Figure 37.3. My Graph

## 37.2 THEGRAPHER .J

### SAY IT WITH GRAPHS

You will be prompted to create a small window as in Figure 37.3. This graph has the title My Graph. Hold down the right mouse button in the window. A menu of graph editing operations will appear as in Figure 37.4.

```
D;Ier~ Link
&=h~n9e ib P.I
ljbel g,nill~.r
l&'bel l~.ro0~.r
Dir~..ct~.i1
SIdPg
~ Boi0dP.r
```

```
'h;~d"
"Tr"P
```

Figure 37.4. A Menu of Graph Editing Operations  
Here's how to use this menu to:

Add a Node Start by selecting Add Node. Grapher will prompt you for the name of the node (See Figure 37.5.) and then its position.

Figure 37.5. Grapher prompts for the name of the node to add after Add Node is chosen from the graph editing menu.

Position the node by moving the mouse cursor to the desired location and clicking a mouse button. Figure 37.6 shows the graph with two nodes added using this menu.

```
~ir0r-ri~tle
s~~'ondnod~
```

Figure 37.6. Two nodes added to MY GRAPH using the graph edit.q.g menu AddALink Select Add Link from the graph editing menu The Prompt window will prompt you to select the two nodes to be linked. (See Figure 37.7.) Do this, and the link will be added.

o .

```
first-node
,second-node
```

Figure 37.7. The Prompt window will prompt you to select the two nodes to link.

THEGRAPHER 37.3

SAY IT WITH GRAPHS

DeleteALink Select Delete Link from the graph editing menu. The Prompt window will prompt you to select the two nodes that should no longer be linked. (See Figure 37.8.) Do this, and the link will be deleted.

```
r_ 'rr-n>';1~
; '~"or,'j-nod;
```

Figure 37.8. The Prompt window will prompt you to select two nodes that should no longer be linked.

Delete A Node Select Delete Node from the graph editing menu. The Prompt window will prompt you to select the node to be deleted. (See Figure 37.9.) Do this, and the node will be deleted.

```
firs. r-nod"
,L'0fl'S1-fl0d~
```

Figure 37.9. The prompt to delete a node

Moving a Node Select "Delete Node" from the graph editing menu. Choose a node pointing to the it with the mouse cursor, and pressing and holding the left mouse button. When you move the mouse cursor, the node will be dragged along. When the node is at the new position, release the mouse button to deposit the node.

The commands in this menu are easy to learn. Experiment with them!

### 37.2 Making a Graph from a List

Typically, a graph is used to display one of your program's data structures. Here is how that is done.

LATOUTSEXPR takes a list and returns a GRAPH record. The syntax of the function is

```
(UYWTSEXPR sexpr format ~xing font motberd
penonald fam;lyd)
For example:
```

```
(u~T10Q AKiliL.TREE '(MILL (l'~ ~ CAT) Bili FISH))
AaIliL.6liN
```

### 37.4 THEGRAPHER

#### MAKING A GRAPH FROM A LIST

```
b~YouTSE*PR AKiliL .TREE øHoRIZ0NTALi~)
```

(Eli N AHiliL.GliPN øNj Grøpbø NIL KIL a T)

This is how Figure 37.1 was produced.

### 37.3 Incorporating Grapher into Your Program

The Grapher is designed to be built into other programs. It can call functions when, for example, a mouse button is clicked on a node. The function SHOWGRAPH does this:

```
(~liPH graph window leflbuttonfn middlebuttonfn
topjusti~Rg alloweditflg copybuttoneventfn)
```

For example, the third argument to SHOWGRAPH, leftbuttonfn, is a function that is called when the left mouse button is pressed in the graph window. Try this:

```
(DEFIK—Q (~.LEFT.BUTTON.FUNCTION
(TNE.6liPHNooE THE.GliPH.wIN~)
(INSPECT TNE.6liPNNooE)))
```

```
(~liPH FIILY.61PN øInspoct&bla fiilyø
(F~TIK N".LEFT.BUTTa.FuNCTIo*)
liIL NIL T)
```

In the example above, liT.LEFT.BUTTON. FUNCTION simply calls the inspector. Note that the function should be written assuming it will be passed a graphnode and the window that holds the graph. Try adding a function of your own.

### 37.4 More of Grapher

Some other Library packages make use of the Grapher. (Note: Grapher needs to be loaded with the packages to use these functions.)

ø NASTERSCOPE: The Browser package modifies the Masterscope command, . SHOW PATHS, so that its output is displayed as a graph (using Grapher) instead of simply printed.

ø GRAPHZOOM: allows a graph to be redisplayed larger or smaller automatically.

THEGRAPHER 375

----- Next Message -----

Date: 19 Dec 91 17:11 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.171147pst.43009@origami.parc.xerox.com>?::>

<-----RFC822 headers----->

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11697>;  
 Thu, 19 Dec 1991 17:11:55 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:11:47 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 <-----RFC822 headers----->

## 41. RESOURCE MANAGEMENT

### 41.1 Naming Variables and Records

You will find times when one environment simultaneously hosts a number of different programs. Running a demo of several programs, or reloading the entire Interlisp-D environment from

floppies when it contains several different programs, are two examples that could, if you aren't careful, provide a few problems. Here are a few tips on how to prevent problems:

- Ø If you change the value of a system variable, `ffENUHELDVAIT` for example, or connect to a directory other than `(DsK)<LlsPFILes>`, write a function to reset the variable or directory to its original value. Run this function when you are finished working. This is especially important if you change any of the system menus.

- Ø Don't redefine Interlisp-D functions or `CLISP` words. Remember, if you reset an atom's value or function definition at the top level (in the Interlisp-D Executive Window), the message (Some.Crucial.Function. Or. Variable redefined), appears. If this is not what you wanted, type `UNDO` immediately!

If, however, you reset the value or function definition of an atom inside your program, a warning message will not be printed.

- Ø Make the atom names in your programs as unique as possible. To do this without filling your program with unreadable names that noone, including you, can remember, prefix your variable names with the initials of your program. Even then, check to see that they are not already being used with the function `BOUNDP`. For example, type:  
(~P ØB&ckgroundhnu)

This atom is bound to the menu that appears when you press the left mouse button when the mouse cursor is not in any window. `BOUKDP` returns T. `BOUNDP` returns NIL if its argument does not currently have a value.

- Ø Make your function names as unique as possible. Once again, prefixing function names with the initials of your program can be helpful in making them unique, but even so, check to see that they are not already being used. `GETD` is the Interlisp-D function that returns the function definition of an atom, if it has one. If an atom has no function definition, `GETO` returns NIL. For example, type:  
(GEffl 'CAR)

## RESOURCE MANAGEMENT 411

### NAMING VARIABLES AND RECORDS

A non-NIL value is returned. The atom `CAR` already has a function definition.

- Ø Use complete record field names in record `FETCHes` and `REPLACEs` when your code is not compiled. A Complete record field name is a list Consisting of the record declaration name and the field name. Consider the following example:  
`RECORD N~ FIRST LAST`)

```
SETQ Nyfflrn create Nl— FIRST 'John LAST '~ith))
FETCH (~ FIRST) OF Mylrn)
```

- Ø Avoid reusing names that are field names of Interlisp-D System records. A few examples of system records follow. Do not reuse these names.

```
RECORD RE6IOl (LEFT RoTTOl WIDTH NEIGHT))
RECORD POSITIK xC00RD
RECORD Ili6E~ BITliYCP00RD)))
```



ø When you select a record name and field names for a new record, check to see whether those names have already been used.

Call the function RECLOOK, with your record name as an argument, in the Interlis~D Executive Window. (See Figure 41.1.) If your record name is already a record, the record definition will be returned; otherwise the function will return NIL.

- . 11

```
4..Oø:ø(fi.ECL)OY~ FB;~ITiON)
!'P\ECCPO
PO~1TI)N
```

```
[øT\L~'.lp:E)F;,P "So'1P~D!
(8Nfl (LlSTP O~TUM\
(NUl18—P~P !',C4l'~ D~TUfi1:))
```

```
(i\"('~T—h1\\j (NUMBER— (CDR OurOIl]
5ik(P~ECLOUff N~,~P~~)
NIL
5~'~E
```

Figure 41.1. RECLOOK returns the record definition if its argument is already declared as a record, NIL otherwise.

Call the function FIELDLOOK with your new field name in the Interlis~D Executive Window. (See Figure 41.2.) If your field name is already a field name in another record, the record definition will be returned; otherwise the function will return NIL.

## 412 RESOURCE MANAGEMENT ø1

### NAMING VARIABLES AND RECORDS

```
, 1
~.4'+(fiELOLOOf Y96COORD)
((RECORD
pO'e.ITION

(:~'COORO \COORD)
['TY'PE"ø (~NP (Llvt~TP O4TUbl!
(NUblBERP (CAR D,iTUhl\
I:.NUMBERP ("OR D~TUftt]
(S\ø'~~.TEb1)\)
```

```
55~(FIELPLOPh .ip~;\
NIL
58-
```

Figure 41.2. FIELDLOOK returns the record definition if its argument is already the field of a record. NIL otherwise.

## 41.2 Some Space and Time Considerations

In order for your program to run at maximum speed, you must efficiently use the space available on the system. The following section points out areas that you may not know are wasting valuable space, and tips on how to prevent this waste.

Often programs are written so that new data structures are created each time the program is run. This is wasteful. Write your programs so that they only create new variables and other data structures conditionally. If a structure has already been

created, use it instead of creating a new one.

Some time and space can be saved by changing your RECORD and TYPERECORD declarations to DATATYPE. DATATYPE is used the same way as the functions RECORD and TYPERECORD. (See Chapter 24.) In addition, the same FETCH and REPLACE commands can be used with the data structure DATATYPE creates. The difference is that the data structure DATATYPE creates cannot be treated as a list the way RECORDs and TYPERECORDs can.

#### 41.2.1 Global Variables

Once defined, global variables remain until Interlisp-D is reloaded. Avoid using global variables if at all possible!

One specific problem arises when programs use the function 6ENSYff. In program development, many atoms are created that may no longer be useful. Hints:

Ø Use

```
(DELDEF atomname 'PKP)
to delete property lists, and
(DELDEF atomname 'vARS)
```

RESOURCE MANAGEMENT 413

----- Next Message -----

Date: 19 Dec 91 17:15 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.171603pst.43009@origami.parc.xerox.com>.:>

<-----RFC822 headers-----

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11560>;  
 Thu, 19 Dec 1991 17:16:06 PST

Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:16:03 -0800

From: John Sybalsky <sybalsky.PARC@xerox.com>

-----RFC822 headers----->

#### SOME SPACE AND TIME CONSIDERATIONS

to have the atom act like it is not defined.

These not only remove the definition from memory, but also change the appropriate f 11 eCoffS that the deleted object was associated with so that the file package will not attempt to save the object (function, variable, record definition, and so forth) the next time the file is made. Just doing something like (SETQ (arg at~nm) '~IE)

looks like it will have the same effect as the second DELDEF above, but the SETQ doesn't update the file package.

Ø If you are generating atom names with GENSYN, try to keep a list of the atom names that are no longer needed. Reuse these atom names, before generating new ones. There is a (fairly large) maximum to the number of atoms you can have, but things slow down considerably when you create lots of atoms.

Ø When possible, use a data structure such as a list or an array, instead of many individual atoms. Such a structure has only one pointer to it. Once this pointer is removed, the whole Structure will be garbage collected and space reclaimed.

### 41.2.2 Circular Lists

If your program is creating circular lists, a lot of space may be wasted. (Note that many cross linked data structures end up having circularities.) Hints when using circular lists:  
 ø Write a function to remove pointers that make lists circular when you are through with the circular list.

ø If you are working with circular lists of windows, bind your main window to a unique global variable. Write window creation conditionally so that if the binding of that variable is already a window, use it, and only create a new window if that variable is unbound or NIL.

Here is an example that illustrates the problem. When several auxiliary windows are built, pointers to these windows are usually kept on the main window's property list. Each auxiliary window also typically keeps a pointer to the main window on its property list. If the top level function creates windows rather than reusing existing ones, there will be many lists of useless windows cluttering the work space. Or, if such a main window is closed and will not be used again, you will have to break the links by deleting the relevant properties from both the main window and all of the auxiliary windows first. This is usually done by putting a special CLOSEFLI on the main window and all of its auxiliary windows.

### 41.2.3 When You Run Out Of Space

Typically, if you generate a lot of structure that won't get garbage collected, you will eventually run out of space. The important part is being able to track down those structures and

## 41.4 RESOURCE MANAGEMENT

### SOME SPACE AND TIME CONSIDERATIONS

the code that generates them in order to become more space efficient.

The Lisp Library Package GCHAX.DCOM can be used to track down pointers to data structures. The basic idea is that GCHAX will return the number of references to a particular data structure.

A special function exists that allows you to get a little extra space so that you can try to save your work when you get toward the edge (usually noted by a message indicating that you should save your work and sysin a fresh Lisp). The GAINSPACE function allows you to delete non-essential data structures. To use it, type:

```
(LIKSPACE)
```

into the Interlisp-D Executive Window. Answer "N" to all questions except the following.

ø Delete edit history  
 ø Delete history list.

ø Delete values of old variables.  
 ø Delete your MASTERSCOPE database  
 ø Delete information for undoing your greeting.  
 Save your work and reload Lisp as soon as possible.

## RESOURCE MANAGEMENT 41 S

----- Next Message -----

Date: 19 Dec 91 17:23 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.172334pst.43009@origami.parc.xerox.com>.:>

<-----RFC822 headers----->  
 Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11625>;  
 Thu, 19 Dec 1991 17:23:37 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:23:34 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 <-----RFC822 headers----->

## 42. SIMPLE INTERACTIONS WITH THE CURSOR, A BITMAP, AND A WINDOW

The purpose of this chapter is to show you how to build a moderately tricky interactive interface with the various Interlis~D display facilities. In particular how to move a large bitmap (larger than 16 x 16 pixels) around inside a window To do this, you will change the CURSORINFN and CURSOROUTFN properties of the window. If you would also like to then set the bitmap in place in the window, you must reset the BUTTOKEVENTFN. This chapter explains how to create the mobile bitmap.

### 42.1 An Example Function Using GETMOUSESTATE

One function that you will use to "trace the cursor" (have a bitmap follow the cursor around in a window) is GETNOUSESTATE. This function finds the current state of the mouse, and resets global system variables, such as LASTMOUSEX and LASTMOUSEY.

As an example of how this function works, create a window by typing

```
(SETQ EzMPLE.wIN~ (CREATEI))
```

into the Interlisp-D Executive window, and sweeping out a window. Now, type in the function  
 (DEFIKEY (PRIKTC00RDS (V)

```
P~TPRI*T Ø(Ø ~TuouSEx ., . US~EY Ø)Ø)
BLocK)
```

```
6E~SESTATE)))
```

This function calls GETNOUSESTATE and then prints the new values of LASTNOUSEX and LASTMOUSET in the promptwindow. To use it, type

```
(WIKraoPRoP EXIPLE .ilI*~ 'CURSD~EDFK 'PRIaTC00RDS)
The window property CURSORffOVEDFN, used in this example,
will evaluate the function PRINTCOORDS each time the cursor is
moved when it is inside the window. The position coordinates of
the mouse cursor will appear in the prompt window. (See Figure
42.1.)
```

## SIMPLE INTERACTIONS WITH THE CURSOR, A BITMAP, AND A WINDOW 42 f

### AN EXAMPLE FUNCTION USING GETMOUSESTATE

Figure 42.1. The current position coordinates of the mouse cursor are shown in the prompt window

#### 42.2 Advising GETMOUSESTATE

For the bitmap to follow the moving mouse cursor, the function GETMOUSESTATE is advised. When you advise a function, you can add new commands to the function without knowing how it is actually implemented. The syntax for advise is

(RISE fn when where what)

fn is the name of the function to be augmented.  
when and where are optional arguments. when specifies whether the change should be made before, after, or around the body of the function. The values expected are BEFORE, AFTER, or AROUND.

what specifies the additional code.

In the example, the additional code, what, moves the bitmap to the position of the mouse cursor. The function GETMOUSESTATE will be ADVISED when the mouse moves into the window. This will cause the bitmap to follow the mouse cursor. ADVISE will be undone when the mouse leaves the window or when a mouse button is pushed. The ADVISING will be done and undone by changing the CURSORINFK, CURSOROUTFN, and BUTTONEVENTFK for the window.

#### 42.3 Changing the Cursor

Of the last part of the example, to give the impression that a bitmap is dragged around a window, the original cursor should disappear. Try typing:

```
(CURSOR (CURSORCR—Rrt (6I~PCREAtt 1 l) 1 11
```

## 42.2 INPUT EVENTS WITH THE CURSOR, A STATEMAP, AND A WINDOW

### CHANGING THE CURSOR

into the Interlisp-D Executive Window. This causes the original cursor to disappear. It reappears when you type (CURSOR T)

When the cursor is invisible, and the bitmap moves as the cursor moves, the illusion is given that the bitmap is dragged around the window.

#### 42.4 Functions for "Tracing the cursor"

To actually have a bitmap trace (follow) the cursor, the environment must be set up so that when the cursor enters the tracing region the trace is turned on, and when the cursor leaves the tracing region the trace is turned off. The function

Establish/Trace/Data will do this. Type it in as it appears (note: including the comments will help you remember what the function does later).

```
(DEFIKEQ Establish/Trace/rata
```

[LANR0 (ønd tracebiteap cursor/rightoffse~ cursor/heighteffse~ GCGA6P)

ø This functlri is collød to oestablish tha døti to tracø  
the døsirod bitaapø øøndø is the øind~ in øhich the tracing  
is to take place, {tracebitøap" is the

øcursor/rightoffsetø and ø~~~50~'~~~g~~0t~~~~~5~i~~1 b~~i~~t~~g~~5  
øhich dete~ine the hotspot of the tracing biteap.  
As "cursor/heightof'set and øcursor/rightoffsetø increase  
the cursor hotspot :ves up and to the right.  
If GCGAGP is non-NIL, GcGAC øill be disabled.)

(PRoG NIL

(if (OR NULL ønd)  
(NULL tracebitaap))

then (PLAYTUN— (LIST (CONS 1000 4000)))

(if ~&~&pRET~RN))  
then (GC6A6))

ø Create a blank cursor.)

(SSEETTQQ  
~8BrnUNNKKCTliURCS0ECRtiR(soBIRliNAø(CPCURRsoEARTcEREI,eT~loø)jwxc~~2~øj~  
ø ø Set the CURSOR IN and OUT FNS for ønd to the  
Jolloeing:)

(\*INroNPRoP ønd UTE CURSORINF  
(FU TIrn SETUP/Tlic

(WINDoNPooP ønd~~TE CURSoRoUTFENNJJ  
(FU TIoN UNTlic—/CURSOR))

(O ø To all", ' the bita,ap to be set den in the øindw bY  
pressing a "ouse button, include this line.  
Øther', 'ise, it is not needed)

(WINnoNPRoP ønd (UTE RUTToNEVENTFN)  
(FUNCTIoN PLACEIBITNAPIINIwINrGN))

Set up Global Variables for the tracing 9eration)  
(SETQ øTRAcEIITNApø tracebiteap)  
TQ øRIGHTTliCE'oFFSETø(øR cursor/rightoffse~ 0)

5>sfE~TQ øHEIGHTTRACEIoFFSETø(ØR cursor/hei htoffset ))  
TQ ø0LQBIT~PPosITIoNø(BiffIPCREATE llnArnIOTN tracebitøap)

(SETQ øTliCCwfNDoNø rndj)) BITNApHEI6hT tracebiteap)))

SIMPLE INTERAcT'ons WITH THE cuRsoR. A BJTMAP. AND A WINDOW 423  
1

FUNCTIONS FOR "TRACING THE CURSOR'

When the function Establish/Trace/Data is called, the  
functions SETUP/TRACE and UNTRACE/CURSOR will be installed  
as the values of the window's WINDOWPROP□, and will be used  
to turn the trace on and off. Those functions should be typed in,  
then:

(DEFINEQ SETUP/TRACE

[ADA end)

(O This function is and's cuRSORIKFK.  
It siepiy resoti thø last trace position and the current  
tracing region. It also raadviseS fiETNouSESTATE to perform  
the trace function after each call.)

(if øTRAcEBITNAPø

then SETQ hLAST-IPACE-XPO5ø -zOo  
SETQ øLAST-TRACE-YPOSø -zooøt)

SETQ øvNDREGIaø (WINOaNPROP and (ATE REGION)))  
WIN~fIROP and (ATE TliCIK)  
T)

ø :ake the cursor disappear)

CURSOR hBLANKTRACECURSORø)  
ANISE QUOTE GEThOUSESTATE)  
QUOTE AFTER)  
IL

(QUOTE (TPACE/CURSOR)))

(Dt~EQ(UaNTRACE/CURSOR

ø Th1s function is ønd's CURsOROUTFN.  
The function first checks if the cursor is currently being  
traced; if so, it replaces thø tracing biiap aith ahat is  
under it and then turns tracing off by unadvlsng  
6ETNOUSESTATE and setting thø TliCIK aindä propertj of  
hTRACEWINOoOø TO NIL.)

(if (VIN~PnOP øTRACEWINOONh(QUOTE TliCIK))  
then (BITBLT ø0LOBITNAPPOSITIONø o o (scREENBIIINAP)  
IPLUS CAR hfdRE6IONø)øLAST-IRACE-xPOSø)  
(IPLUS 1CADR ø:DREGIOffo )øusT-TlicE-YPosø))  
(WINOoePRoP ølliCEMINOONø(QUOTE TRACIK)  
NIL))

replace the original cursor shape)  
(CURSOR T)

unadvise 6E~sESTATE)  
(U~"ISE (QUOTE 6ETNOUSESTAIE)))

The function SETUP/TRACE has a helper function that you must  
also type in. It is TRACE/CURSOR:

(DEFINEQ (TlicE/CURSoR  
[LANRli NIL

ø This functi: does thø actual BITBLTln of thø tracing  
blteap. This functla Is cølled after a GE f TATE, abl ø  
tracing.)

(PRoG ( xpos IDIFFERENCE LAsTNOUsEZ øTRACEWINnoNø øRIGHTTRACE/OFF  
[ypoo IDIFFERENCE LAsTNOUsEY øTRACE\*INr~t øNEIGNtTRACE/OfFSsEETiJ)))  
ø If there Is an ørror In thø function, resS thø riKbt  
button to unodvlsø thø function. This øill eep thø ac inø  
fr: locking up.)

(If (LASTNOUSESTATE RIGiit)

```
(if ~t1h~~ (NUNAPuISE (QUOTE 6EIS~5ESIATE)))
Q xpoa ØLAST-TRACE-XP0sØ
(KEO ,pea ØLAsl-TeACE-YPOSØj)
thØn
```

```
Ø Restoro ahØt ~s undØr the eld pooltla of thØ trØcØ
OilUp)
```

```
(SITGLY ØOLlillnApposITIGØØ o o (IREE5illiA~)
```

```
Ø2A SIMPLE INTE~CJ\OHS WITH THE CURSOR. A BtTMAP. AfIO A WINDOW
1
```

```
F,
FUNCTIONS FOR "TRACING THE CURSOR"
```

```
IPLUS CAR Øil
```

```
IPLUS CADR Ø:DDRRESEGISIrØ2ØLASTTRAC
```

```
Ø søvØ Øb&t Ø111 bØ undØr thØ position of thØ nee trØcØ
biteap)
```

```
(51 TILT SCREENBITNAP)
[IPLUS (CAR ØaDREGIaØ)
xpos)
```

```
(IPLUS ØvNDREGIoaØ O O)
```

```
BiffILT the tracG blt:p onto thØ nØØ position of thØ
eouse.)
```

```
(8ITBLT ØTRACEBITNAP† O O ~5SCREENBITNAP)
(IPLUS (CAR ØilDRE ION"))
```

```
(fPLUS (CADS ØffOREGIONØ)
ypos
```

```
NIL NIL YE INPUT)
```

```
(ONDTE P liT))
```

```
Ø Savu the current position as the last trace position.)
```

```
(SETQ ØLAST-TRACE-xPDSØ xpos
(SETQ "LAST-TRACE-YPOS† ypos
```

```
The helper function for UHTRACE/CURSOR, called
UNDO/TRACE/DATA, must also be added to the environment:
(DEFINEQ (UNDO/TRACE/DATA
[LISA NIL
```

```
Ø The purpose of this function is to turn tracing off and
to free up the global variables used to trace the bitaap, so
that thej can be garbage collected.)
```

```
Check if the cursor is currently being traced.
It so, turn it off.)
```

```
UiTRACE/CURSoR)
```

```
WINDoNPRDP †TliCE*IN~†(uTE CURSDRINFN)
NIL)
```



```
(WINDo*PRDP tTRACEwINDDNt(uTE CUR~R0UTFN)
NIL)
```

```
SETQ "TRACEBITsAPI NIL)
SETQ 0RIGNTTlicE/oFFsET0 NIL)
SETQ 0HEIGHTTRACE/OFFSET0 NIL)
SETQ 0OLDBITliPP0SITIDN0 NIL)
SETQ tTRACE*I~t NIL)
```

0 Turn GCGAG on)

(6C6A6 TJ))

Finally, if you inCluded the WINDOWPROP to allow the user to place the bitmap in the window by pressing a mouse button, you must also type this function:

```
(D[E~DEAQ, 0nd)
UNADVISE (SETNDUSESTATE))

fBITBLT 0TliCEBITNAP0 O O SCREENBIINAP)
(IPLUS (CA 0N0)
xpo

(IPLUS (CADR 0iDREGION0)
ypos)

NIL NIL (UTE INPUT)
(ATE PAINT]
```

That's all the functions!

SIMPLE INTERAcTioNs NITH THE cuRsoR, A BITMAP, AND A WINDOW 42 S

p

RUNNING THE FuNcTIgh5

## 42.5 Running the Functions

To run the functions you just typed in, first set a variable to a window by typing something like  
(SETQ EXMPLE.wIN~ (CR—ATEI))

into the Interlisp-D Executive window, and sweeping out a new window. No'rv, set a variable to a bitmap, by typing, perhaps,  
(SETQ EXlPLE.BTn (—DITl))  
Type

```
(Etab1isfl'Trsc'e'Oo~ EXIPL.E.WIN~ EXIPL.E.BTK))
When you move the cursor into the window, the cursor will drag
the bitmap.
```

(Note: If you want to be able to make menu selections while tracing the cursor, make sure that the hotspot of the cursor is set to the extreme right of the bitmap. Otherwise, the menu will be destroyed by the BITSLTs of the trace functions.)  
To stop tracing, either

- 0 move the mouse cursor out of the window;
- 0 press the right mouse button;
- 0 call the function UNTRACE/CURS0R.

u.6 SIMPLE INTEPACT~NS WITH THE CURSOR. A SITMAP. AND A WIN00W

----- Next Message -----

Date: 19 Dec 91 17:30 PST  
 From: sybalsky:PARC:Xerox  
 To: sybalsky  
 Message-ID: <<91Dec19.173105pst.43009@origami.parc.xerox.com>.?::>

<-----RFC822 headers----->

Received: from origami.parc.xerox.com ([13.1.100.224]) by alpha.xerox.com with SMTP id <11702>;  
 Thu, 19 Dec 1991 17:31:18 PST  
 Received: by origami.parc.xerox.com id <43009>; Thu, 19 Dec 1991 17:31:05 -0800  
 From: John Sybalsky <sybalsky.PARC@xerox.com>  
 <-----RFC822 headers----->

re. REFERENCES THAT WILL  
 BE USEFUL TO YOU

Here are some references to works that will be useful to you in addition to this primer. Some of these you have already been referred to, such as:

- Ø The Interlisp-D Reference Manual
- Ø The Library Packages Manual
- Ø The User's Guide to SKETCH
- Ø The Interlisp-D User's Guide

In addition, you can learn more about LISP with the books:

- Ø Interlisp-D: The language and its usage by Steven H. Kaiser. This book was published in 1986 by John Wiley and Sons, NY.

- Ø Essential LISP by John Anderson, Albert Corbett, and Brian Reiser. This book was published in 1986 by Addison Wesley Publishing Company, Reading, MA. It was informed by research on how beginners learn LISP.

- Ø The Little Lisper by Daniel P. Friedman and Matthias Felleisen. The second edition of this book was published in 1986 by SRA Associates, Chicago. This book is a deceptively simple introduction to recursive programming and the flexible data structures provided by LISP.

- Ø LISP by Patrick Winston and Berthold Horn. The second edition of this book was published in 1985 by the Addison Wesley Publishing Company, Reading, MA.

- Ø LISP: A Gentle Introduction to Symbolic Computation by David S. Touretzky. This book was published in 1984 by the Harper and Row Publishing Company, NY.

Finally, there are three articles about the Interlisp Programming environment:

- Ø Power Tools for Programmers by Beau Sheil. It appeared in Datamation in February, 1983, Pages 131 - 144.
- Ø The Interlisp Programming Environment by Warren Teitelman and Larry Masinter. It appeared in April, 1981, in IEEE Computer, Volume 14:1, Pages 25 - 34.

- Ø Programming in an Interactive Environment: the LISP Experience by Erik Sandewall. It appeared in March,

1978, in the ACM Computing Surveys, Volume 10:1, pages 35 - 71.

Each of these articles was reprinted in the book Interactive Programming Environments by David R. Barstow, Howard E.

OTHER REFERENCES THAT WILL BE USEFUL TO YOU 441  
I

OTHER REFERENCES THAT WILL BE USEFUL TO YOU

Shrobe, and Erik Sandewail. This book was published in 1984 by McGraw Hill, NY. The first article can be found on pages 19 - 30, the second on pages 83 - 96, and the third on pages 31 - 80.

J:'

ill OTHER REFERENCES THAT WILL BE USEFUL TO YOU

I

----- End Forwarded Messages -----  
—End of message—

## APPENDIX A. THE EXEC

In most Common Lisp implementations, there is a "top-level **read-eval-print** loop," which reads an expression, evaluates it, and prints the results. In Xerox Common Lisp, the Exec acts as the top-level loop, but in addition to **read-eval-print**, it also performs a number of other tasks, and allows a much greater range of inputs. This appendix contains information from the Lyric and Medley releases. Medley additions are indicated with revision bars in the right margin.

The Exec is based on concepts from the Interlisp Programmer's Assistant (see the *Interlisp-D Reference Manual*).

The Exec traps all throws, and recovers gracefully. It prints all values resulting from evaluation, on separate lines. When zero values are returned, nothing is printed.

The Exec keeps track of your previous input, in a structure called the history list. A history list is a list of the information associated with each of the individual events that have occurred, where each event corresponds to one input. Associated with each event on the history list is the input, its values, plus other optional information such as side-effects, formatting information, etc.

The following dialogue contains illustrative examples and gives the flavor of the use of the Exec. Be sure to type these examples to an Exec whose **\*PACKAGE\*** is set to the **XCL-USER** package. The Exec that Lisp starts up with is set to the **XCL-USER** package. Each prompt consists of an event number and a prompt character ("**>**").

```
12>(setq foo 5)
5
13>(setq foo 10)
10
14>undocr
SETQ undone.
15>foocr
5
```

*This is an example of direct communication with the Exec. You have instructed the Exec to **undo** the previous event.*

...

```
25>set(lst1 (a b c))
(A B C)
26>(setq lst2 '(d e f))
(D E F)
27>(mapc #'(lambda (x) (setf (get x 'myprop) t)) lst1)
(A B C)
```

*The Exec accepts input both in APPLY format (the **SET**) and EVAL format (the **SETQ**.) In event 27, the user adds a property MYPROP to the symbols A, B, and C.*

```
28>use lst2 for lst1 in 27cr
```

NIL

*You just instructed the Exec to go back to event number 27, substitute **LST2** for **LST1**, and then re-execute the expression. You could have also used -2 instead of 27, specifying a relative address.*

.  
.  
.

46>(setf my-hash-table (make-hash-table))

#<Hash-Table @ 66,114034>

47>(setf (gethash 'foo my-hash-table) (string 'foo))  
"FOO"

*If **STRING** were computationally expensive (which it is not), then you might be caching its value for later use.*

48>use fie for foo in string<sup>cr</sup>  
"FIE"

*You now decide you would like to redo the **SETF** with a different value. You specify the event using "**IN STRING**" rather than **SETF**.*

49>?? use<sup>cr</sup>

**USE FIE FOR FOO IN STRING**  
48> (SETF (GETHASH 'FIE MY-HASH-TABLE)  
(STRING 'FIE))  
"FIE"

*Here you ask the Exec (using the ?? command) what it has on its history list for the last input. Since the event corresponds to a command, the Exec displays both the original command and the generated input.*

The most common interaction with the Exec occurs at the top level or in the debugger, where you type in expressions for evaluation, and see the values printed out. In this mode, the Exec acts much like a standard Common Lisp top-level loop, except that before attempting to evaluate an input, the Exec first stores it in a new entry on the history list. Thus if the operation is aborted or causes an error, the input is still saved and available for modification and/or re-execution. The Exec also notes new functions and variables to be added to its spelling lists to enable future corrections.

After updating the history list, the Exec executes the computation (i.e., evaluates the form or applies the function to its arguments), saves the value in the entry on the history list corresponding to the input, and prints the result. Finally the Exec displays a prompt to indicate it is again ready for input.

---

## Input Formats

The Exec accepts three forms of input: an expression to be evaluated (EVAL-format), a function-name and arguments to apply it to (APPLY-format), and Exec commands, as follows:

### EVAL-format input

If you type a single expression, either followed by a carriage-return, or, in the case of a list, terminated with balanced parenthesis, the expression is evaluated and the value is returned. For example, if the value of the variable **FOO** is the list (**A B C**):

```
32>FOOcr
(A B C)
```

Similarly, if you type a Lisp expression, beginning with a left parenthesis and terminated by a matching right parenthesis, the form is simply passed to **EVAL** for evaluation. Notice that it is not necessary to type a carriage return at the end of such a form; the reader will supply one automatically. If a carriage-return is typed before the final matching right parenthesis or bracket, it is treated the same as a space, and input continues. The following examples are interpreted identically:

```
123> (+ 1 (* 2 3))
7
124> (+ 1 (*cr
2 3))
7
```

#### APPLY-format input

Often, when typing at the keyboard, you call functions with constant argument values, which would have to be quoted if you typed them in "EVAL-format." For convenience, if you type a symbol immediately followed by a list form, the symbol is **APPLY**ed to the elements within the list, unevaluated. The input is terminated by the matching right parenthesis. For example, typing **LOAD(FOO)** is equivalent to typing **(LOAD 'FOO)**, and **GET(X COLOR)** is equivalent to **(GET 'X 'COLOR)**. As a simple special case, a single right parenthesis is treated as a balanced set of parentheses, e.g.

```
125>UNBREAK)
```

is equivalent to

```
125>UNBREAK()
```

The reader will only supply the "carriage return" automatically if no space appears between the initial symbol and the list that follows; if there is a space after the initial symbol on the line and the list that follows, the input is not terminated until a carriage return is explicitly typed.

Note that APPLY-format input cannot be used for macros or special forms.

#### Exec commands

The Exec recognizes a number of commands, which usually refer to past events on the history list. These commands are treated specially; for example, they may not be put on the history list. The format of a command is always a line beginning with the command name. (The Exec looks up the command name independent of package, so that Exec commands are package independent.) The remainder of the line, if any, is treated as "arguments" to the command. For example,

```
128>UNDOcr
mapc undone
129>UNDO (FOO --)cr
foo undone
```

are all valid command inputs.

## Multiple Execs and the Exec's Type

#### Multiple Execs

More than one Exec can be active at any one time. New Execs can be created by selecting the Exec menu item in the background pop-up menu. When a prompt is printed for an event in other than the

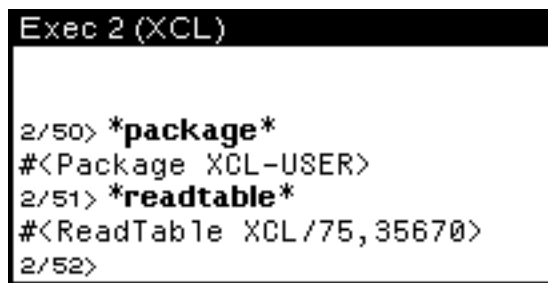
first Exec, the prompt is preceded with the Exec number; for example:

2/50>

might be a prompt in Exec 2. All Execs share the same history list, but each event records which Exec it goes with. That is, although a single global list exists, the Xerox Lisp history system maintains the separate threads of control within each Exec.

Exec type

Several variables are very important to an Exec since they control the format of reading and printing. Together these variables describe a type of exec. Put another way, this is the Exec's mode. To allow easier setting of these modes some standard bindings for the variables have been named. The names provide the user an Exec of the Common Lisp (CL), Interlisp (IL) or Xerox Extended Common Lisp (XCL) type. An Exec's type is usually displayed in the title bar of its window in parentheses:



```
Exec 2 (XCL)
2/50> *package*
#<Package XCL-USER>
2/51> *readtable*
#<ReadTable XCL/75,35670>
2/52>
```

---

## Event Specification

Exec commands, like **UNDO**, frequently refer to previous events in the session's history. All Exec commands use the same conventions and syntax for indicating which event(s) the command refers to. This section shows you the syntax used to specify previous events.

An event address identifies one event on the history list. For example, the event address **42** refers to the event with event number 42, and **-2** refers to two events back in the *current* Exec. Usually, an event address will contain only one or two commands.

Event addresses can be concatenated. For example, if **FOO** refers to event *N*, **FOO FIE** will refer to the first event before event *N* which contains **FIE**.

The symbols used in event addresses (such as **AND**, **F**, **=**, etc. are compared with **STRING-EQUAL**, so that it does not matter what the current package is when you type an event address symbol to an Exec.

Event addresses are interpreted as follows:

*N* (an integer)

If *N* is positive, it refers to the event with event number *N* (no matter which Exec the event occurred in.) If *N* is negative, it always refers to the event *-N* events backwards counting *only* events belonging to the *current* Exec.

**F**

Specifies that the next object in the event address is to be searched for, regardless of what it is. For example, **F -2** looks for an event containing **-2**.

= Specifies that the next object is to be searched for in the *values* of events, instead of the inputs.

**SUCHTHAT** *PRED* Specifies an event for which the function *PRED* returns true. *PRED* should be a function of two arguments, the input portion of the event, and the event itself.

*PAT* Any other event address command specifies an event whose input contains an expression that matches *PAT*. When multiple Execs are active, all events are searched, no matter which Exec they belong to. The pattern can be a simple symbol, or a more complex search pattern.

Note: Specifications used below of the form *EventAddress* refer to event addresses, as described above. Since an event address may contain multiple words, the event address is parsed by searching for the words which delimit it. For example, in *EventAddress AND EventAddress*, the notation *EventAddress* corresponds to all words up to the **AND** in the event specification, and *EventAddress* to all words after the **AND** in the event specification.

**FROM** *EventAddress* All events since *EventAddress*, inclusive. For example, if there is a single Exec and the current event is number 53, then **FROM 49** specifies events 49, 50, 51, and 52. **FROM** will include events from *all* Execs.

**ALL** *EventAddress* Specifies all events satisfying *EventAddress*. For example, **ALL LOAD, ALL SUCHTHAT FOO-P**.

empty If nothing is specified, it is the same as specifying **-1**, i.e., the last event in the current Exec.

*EventSpec AND EventSpec AND . . . AND EventSpec*

Each of the *EventSpec* is an event specification. The lists of events are concatenated. For example, **ALL MAPC AND ALL STRING AND 32** specifies all events containing **MAPC**, all containing **STRING**, and also event **32**. Duplicate events are removed.

## Exec Commands

All Exec commands are input as lines which begin with the name of the command. The name of an Exec command is not a symbol and therefore is not sensitive to the setting of the current package ( the value of **\*PACKAGE\***).

*EventSpec* is used to denote an event specification which in most cases will be either a specific event address (e.g., 42) or a relative one (e.g., -3). Unless specified otherwise, omitting *EventSpec* is the same as specifying *EventSpec=-1*. For example, **REDO** and **REDO -1** are the same.

**REDO** *EventSpec*

[Exec command]

Redoes the event or events specified by *EventSpec*. For example, **REDO 123** redoes the event numbered 123.



**RETRY *EventSpec***

[Exec command]

Similar to **REDO** except sets the debugger parameters so that any errors that occur while executing *EventSpec* will cause breaks.

**USE NEW [FOR OLD ] [IN *EventSpec* ]**

[Exec command]

Substitutes *NEW* for *OLD* in the events specified by *EventSpec*, and redoes the result. *NEW* and *OLD* can include lists or symbols, etc.

For example, **USE SIN (- X) FOR COS X IN -2 AND -1** will substitute **SIN** for every occurrence of **COS** in the previous two events, and substitute **(- X)** for every occurrence of **X**, and reexecute them. (The substitutions do not change the previous information saved about these events on the history list.)

If *IN EventSpec* is omitted, the first member of *OLD* is used to search for the appropriate event. For example, **USE DEFAULTFONT FOR DEFLATFONT** is equivalent to **USE DEFAULTFONT FOR DEFLATFONT IN F DEFLATFONT**. The **F** is inserted to handle correctly the case where the first member of *OLD* could be interpreted as an event address command.

If *OLD* is omitted, substitution is for the "operator" in that command. For example **FBOUNDP(FF)** followed by **USE CALLS** is equivalent to **USE CALLS FOR FBOUNDP IN -1**.

If *OLD* is not found, **USE** will print a question mark, several spaces and the pattern that was not found. For example, if you specified **USE Y FOR X IN 104** and *X* was not found, "X ?" is printed to the Exec.

You can also specify more than one substitution simultaneously as follows:

**USE NEW FOR OLD AND ... AND NEW FOR OLD [IN *EventSpec* ]**

[Exec command]

Note: The **USE** command is parsed by a small finite state parser to distinguish the expressions and arguments. For example, **USE FOR FOR AND AND AND FOR FOR** will be parsed correctly.

Every **USE** command involves three pieces of information: the expressions to be substituted, the arguments to be substituted for, and an event specification that defines the input expression in which the substitution takes place. If the **USE** command has the same number of expressions as arguments, the substitution procedure is straightforward. For example, **USE X Y FOR U V** means substitute *X* for *U* and *Y* for *V*, and is equivalent to **USE X FOR U AND Y FOR V**.

However, the **USE** command also permits distributive substitutions for substituting several expressions for the same argument. For example, **USE A B C FOR X** means first substitute *A* for *X* then substitute *B* for *X* (in a new copy of the expression), then substitute *C* for *X*. The effect is the same as three separate **USE** commands.

Similarly, **USE A B C FOR D AND X Y Z FOR W** is equivalent to **USE A FOR D AND X FOR W**, followed by **USE B FOR D AND Y**

**FOR W**, followed by **USE C FOR D AND Z FOR W. USE A B C FOR D AND X FOR Y** also corresponds to three substitutions, the first with **A** for **D** and **X** for **Y**, the second with **B** for **D**, and **X** for **Y**, and the third with **C** for **D**, and again **X** for **Y**. However, **USE A B C FOR D AND X Y FOR Z** is ambiguous and will cause an error.

Essentially, the **USE** command operates by proceeding from left to right handling each **AND** separately. Whenever the number of expressions exceeds the number of expressions available, multiple **USE** expressions are generated. Thus **USE A B C D FOR E F** means substitute **A** for **E** at the same time as substituting **B** for **F**, then in another copy of the indicated expression, substitute **C** for **E** and **D** for **F**. This is also equivalent to **USE A C FOR E AND B D FOR F**.

Note: The **USE** command correctly handles the situation where one of the old expressions is the same as one of the new ones, **USE X Y FOR Y X**, or **USE X FOR Y AND Y FOR X**.

---

**? &OPTIONAL NAME** [Exec command]

If *NAME* is not provided describes all available Exec commands by printing the name, argument list, and description of each. With *NAME*, only that command is described.

---

**?? EventSpec** [Exec command]

Prints the most recent event matching the given *EventSpec*.

---

**CONN DIRECTORY** [Exec command]

Changes default pathname to *DIRECTORY*.

---

**DA** [Exec command]

Returns current date and time.

---

**DIR &OPTIONAL PATHNAME &REST KEYWORDS** [Exec command]

Shows a directory listing for *PATHNAME* or the connected directory. If provided, *KEYWORDS* indicate information to be displayed for each file. Some keywords are: *AUTHOR*, *AU*, *CREATIONDATE*, *DA*, etc.

---

**DO-EVENTS &REST INPUTS &ENVIRONMENT ENV** [Exec command]

**DO-EVENTS** is intended as a way of putting together several different events, which can include commands. It executes the multiple *INPUTS* as a single event. The values returned by the **DO-EVENTS** event are the concatenation of the values of the inputs. An input is not an *EventSpec*, but a call to a function or command. If *ENV* is provided it is a lexical environment in which all evaluations (functions and commands) will take place. Event specification in the *INPUTS* should be explicit, not relative, since referring to the last event will reinvoked the executing **DO-EVENTS** command.

---

|                                                                                                                                                                                                                                                   |                |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| <b>FIX</b> &REST <i>EventSpec</i>                                                                                                                                                                                                                 | [Exec command] |
| Edits the specified event prior to reexecuting it. If the number of characters in the <b>Fixed</b> line is less than the variable <b>TTYINFIXLIMIT</b> then it will be edited using TTYIN, otherwise the Lisp editor is called via <b>EDITE</b> . |                |
| <b>FORGET</b> &REST <i>EventSpec</i>                                                                                                                                                                                                              | [Exec command] |
| Erases <b>UNDO</b> information for the specified events.                                                                                                                                                                                          |                |
| <b>NAME</b> <i>COMMAND-NAME</i> &OPTIONAL <i>ARGUMENTS</i> &REST <i>EVENT-SPEC</i>                                                                                                                                                                | [Exec command] |
| Defines a new command, <i>COMMAND-NAME</i> , and its <i>ARGUMENTS</i> , containing the events in <i>EVENT-SPEC</i> .                                                                                                                              |                |
| <b>NDIR</b> &OPTIONAL <i>PATHNAME</i> &REST <i>KEYWORDS</i>                                                                                                                                                                                       | [Exec command] |
| Shows a directory listing for <i>PATHNAME</i> or the connected directory in abbreviated format. If provided, <i>KEYWORDS</i> indicate information to be displayed for each file. Some keywords are: AUTHOR, AU, CREATIONDATE, DA, etc.            |                |
| <b>PL</b> <i>SYMBOL</i>                                                                                                                                                                                                                           | [Exec command] |
| Prints the property list of <i>SYMBOL</i> in an easy to read format.                                                                                                                                                                              |                |
| <b>REMEMBER</b> &REST <i>EVENT-SPEC</i>                                                                                                                                                                                                           | [Exec command] |
| Tells File Manager to remember type-in from specified event(s), <i>EVENT-SPEC</i> , as expressions to save.                                                                                                                                       |                |
| <b>SHH</b> &REST <i>LINE</i>                                                                                                                                                                                                                      | [Exec command] |
| Executes <i>LINE</i> without history list processing.                                                                                                                                                                                             |                |
| <b>UNDO</b> &REST <i>EventSpec</i>                                                                                                                                                                                                                | [Exec command] |
| Undoes the side effects of the specified event (see below under "Undoing").                                                                                                                                                                       |                |
| <b>PP</b> &OPTIONAL <i>NAME</i> &REST <i>TYPES</i>                                                                                                                                                                                                | [Exec command] |
| Shows (prettyprinted) the definitions for <i>NAME</i> specified by <i>TYPES</i> .                                                                                                                                                                 |                |
| <b>SEE</b> &REST <i>FILES</i>                                                                                                                                                                                                                     | [Exec command] |
| Prints the contents of <i>FILES</i> in the Exec window, hiding comments.                                                                                                                                                                          |                |
| <b>SEE*</b> &REST <i>FILES</i>                                                                                                                                                                                                                    | [Exec command] |
| Prints the contents of <i>FILES</i> in the Exec window, showing comments.                                                                                                                                                                         |                |

---

**TIME FORM &KEY REPEAT &ENVIRONMENT ENV** [Exec command]

Times the evaluation of *FORM* in the lexical environment *ENV*, repeating *REPEAT* number of times. Information is displayed in the Exec window.

---

**TY &REST FILES** [Exec command]

Exactly like the **TYPE** Exec command.

---

**TYPE &REST FILES** [Exec command]

Prints the contents of *FILES* in the Exec window, hiding comments.

---

## Variables

---

A number of variables are provided for convenience in the Exec.

**IL:IT** [Variable]

Whenever an event is completed, the global value of the variable **IT** is reset to the event's value. For example,

```
312>(SQRT 2)
1.414214
313>(SQRT IL:IT)
1.189207
```

Following a **??** command, **IL:IT** is set to the value of the last event printed. The inspector has an option for setting the variable **IL:IT** to the current selection or inspected object, as well. The variable **IL:IT** is global, and is shared among all Execs. **IL:IT** is a convenient mechanism for passing values from one process to another.

Note: **IT** is in the INTERLISP package and these examples are intended for an Exec whose **\*PACKAGE\*** is set to **XCL-USER**. Thus, **IT** must be package qualified (the **IL:**).

---

The following variables are maintained independently by each Exec. (When a new Exec is started, the initial values are **NIL**, or, for a nested Exec, the value for the "parent" Exec. However, events executed under a nested Exec will not affect the parent values.)

**CL:-** [Variable]

**CL:+** [Variable]

**CL:++** [Variable]

**CL:+++** [Variable]

While a form is being evaluated by the Exec, the variable **-** is bound to the form, **CL:+** is bound to the previous form, **CL:++** the one before, etc. If the input is in apply-format rather than eval-format, the value of the respective variable is just the function name.

---

**CL:\*** [Variable]

---

**CL:\*\*** [Variable]

---

**CL:\*\*\*** [Variable]

---

While a form is being evaluated by the Exec, the variable **CL:\*** is bound to the (first) value returned by the last event, **CL:\*\*** to the event before that, etc. The variable **CL:\*** differs from **IT** in that **IT** is global while each separate Exec maintains its own copy of **CL:\***, **CL:\*\*** and **CL:\*\*\***. In addition, the history commands change **IT**, but only inputs which are retained on the history list can change **CL:\***.

---

**CL:/** [Variable]

---

**CL://** [Variable]

---

**CL:///** [Variable]

---

While a form is being evaluated by an Exec, the variable **CL:/** is bound to a list of the results of the last event in that Exec, **CL://** to the values of the event before that, etc.

---

## Fonts in the Exec

---

The Exec can use different fonts for displaying the prompt, user's input, intermediate printout, and the values returned by evaluation. The following variables control the Exec's font use:

**PROMPTFONT** [Variable]

---

Font used for printing the event prompt.

---

**INPUTFONT** [Variable]

---

Font used for echoing user's type-in.

---

**PRINTOUTFONT** [Variable]

---

Font used for any intermediate printing caused by execution of a command or evaluation of a form. Initially the same as **DEFAULTFONT**.

---

**VALUEFONT** [Variable]

---

Font used to print the values returned by evaluation of a form. Initially the same as **DEFAULTFONT**.

---

## Changing the Exec

---

**(CHANGESLICE *N* HISTORY —)** [Function]

---

Changes the time-slice of the history list *HISTORY* to *N*. If **NIL**, *HISTORY* defaults to the top level history **LISPXHISTORY**.

Note: The effect of *increasing* the time-slice is gradual: the history list is simply allowed to grow to the corresponding length before any events are forgotten. *Decreasing* the time-slice will immediately remove a sufficient number of the older events to bring the history list down to the proper size. However, **CHANGESLICE** is undoable, so that these events are (temporarily) recoverable. Therefore, if you want to recover the storage associated with these events without waiting *N* more events until the **CHANGESLICE** event drops off the history list, you must perform a **FORGET** command.

## Defining New Commands

You can define new Exec commands using the **XCL:DEFCOMMAND** macro.

**(XCL:DEFCOMMAND NAME ARGUMENT-LIST &REST BODY)** [Macro]

**XCL:DEFCOMMAND** is similar to **XCL:DEFMACRO**, but defines new Exec commands. The *ARGUMENT-LIST* can have keywords, defstructure, and use all of the features of macro argument lists. When *NAME* is subsequently typed to the Exec, the rest of the line is processed like the arguments to a macro, and the *BODY* is executed. **XCL:DEFCOMMAND** is a definer; the File Manager will remember typed-in definitions and allow them to be saved, edited with **EDITDEF**, etc.

There are actually three kinds of commands that can be defined, **:EVAL**, **:QUIET**, and **:INPUT**. Commands can also be marked as only for the debugger, in which case they are labelled as **:DEBUGGER**. The command type is noted by supplying a list for the *NAME* argument to **XCL:DEFCOMMAND**, where the first element of the list is the command name, and the other elements are keyword(s) for the command type and, optionally **:DEBUGGER**.

Note: The documentation string in user defined Exec commands is automatically added to the documentation descriptions by the **CL:DOCUMENTATION** function under the **COMMANDS** type and can be shown using the **? Exec** command.

**:EVAL** This is the default. The body of the command just gets executed, and its value is the value of the event. For example (in an XCL Exec),

```
(DEFCOMMAND (LS :EVAL)
(&OPTIONAL (NAMESTRING *DEFAULT-PATHNAME-DEFAULTS*)
&REST DIRECTORY-KEYWORDS)
(MAPC
 #'(LAMBDA (PATHNAME) (FORMAT T "~&~A" (NAMESTRING PATHNAME)))
 (APPLY #'DIRECTOR NAMESTRING DIRECTORY-KEYWORDS))
(VALUES))
```

would define the **LS** command to print out all file names that match the input namestring. The **(VALUES)** means that no value will be printed by the event, only the intermediate output from the **FORMAT**.

**:QUIET** These commands are evaluated, but neither your input nor the results of the command are stored on the history list. For example, the **??** and **SHH** commands are quiet.

**:INPUT** These commands work more like macros, in that the result of evaluating the command is treated as a new line of input. The **FIX** command is an input command. The result is treated as a line; a single expression in EVAL-format should be returned as a list of the expression to **EVAL**.

---

The new Exec now will not consider unparenthesized input with more than one argument to be in apply format. This is the same behavior as the older execs, e.g.:

list(1) ; is apply format (executes after close paren is typed)

list (1) ; is apply format (second arg is a list, no trailing args given)

list '(1) 2 3 ; is NOT apply format, arguments are evaluated

list 1 2 3 ; is NOT apply format, arguments are evaluated

list 1 ; not legal input: second argument is not a list

---

## Undoing

Note: This discussion only applies to undoing under the Exec, Debugger and within the UNDOABLY macro; editors handle undoing in a different fashion.

The **UNDO** facility allows recording of destructive changes such that they can be played back to restore a previous state. There are two kinds of **UNDO**ing: one is done by the Exec, the other is available for use in a programmer's code. Both methods share information about what kind of operations can be undone and where the changes are recorded.

---

## Undoing in the Exec

---

**UNDO** *EventSpec*

[Exec command]

The Exec's **UNDO** command is implemented by watching the evaluation of forms and requiring undoable operations in that evaluation to save enough information on the history list to reverse their side effects. The Exec simply executes operations, and any undoable changes that occur are automatically saved on the history list by the responsible functions. The **UNDO** command works on itself the same way: it recovers the saved information and performs the corresponding inverses. Thus, **UNDO** is effective on itself, so that you can **UNDO** an **UNDO**, and **UNDO** that, etc.

Only when you attempt to undo an operation does the Exec check to see whether any information has been saved. If none has been saved, and you have specifically named the event you want undone, the Exec types **nothing saved**. (When you just type **UNDO**, the Exec only tries to undo the last operation.)

**UNDO** watches evaluation using **CL:EVALHOOK** (thus, calling **CL:EVALHOOK** cannot be undone). Each form given to **EVAL** is

examined against the list **LISPFNS** to see if it has a corresponding undoable version. If an undoable version of a call is found, it is called with the same arguments instead of the original. Therefore, before evaluating all subforms of your input, the Exec substitutes the corresponding undoable call for any destructive operation. For example, if you type **(DEFUN FOO ...)**, undoable versions of the forms that set the definition into the symbol function cell are evaluated. **FOO**'s function definition itself is not made undoable.

---

## Undoing in Programs

---

There are two ways to make a program undoable. The simplest method is to wrap the program's form in the **UNDOABLY** macro. The other is to call undoable versions of destructive operations directly.

**(XCL:UNDOABLY &REST FORMS)** [Macro]

Executes the forms in *FORMS* using undoable versions of all destructive operations. This is done by "walking" (see **WALKFORM**) all of the *FORMS* and rewriting them to use the undoable versions of destructive operations (**LISPFNS** makes the association).

---

**(STOP-UNDOABLY &REST FORMS )** [Macro]

Normally executes as **PROGN**; however, within an **UNDOABLY** form, explicitly causes *FORMS* not to be done undoably. Turns off rewriting of the *FORMS* to be undoable inside an **UNDOABLY** macro.

---

## Undoable Versions of Common Functions

---

Efficiency and overhead are serious considerations for the execution of a user program. Thus, the programmer may need more control over the saving of undo information than that provided by the **UNDOABLY** macro.

To make a function undoable, you can simply substitute the corresponding undoable function if you want to make a destructive operation in your own program undoable. When the undoable function is called, it will save the undo information in the current event on the history list.

Various operations, most notably **SETF**, have undoable versions. The following undoable macros are initially available:

**UNDOABLY-POP**

**UNDOABLY-PUSH**

**UNDOABLY-PUSHNEW**

**UNDOABLY-REMF**

**UNDOABLY-ROTATEF**

**UNDOABLY-SHIFTF**

**UNDOABLY-DECF**

**UNDOABLY-INCF**



**UNDOABLY-SET-SYMBOL**  
**UNDOABLY-MAKUNBOUND**  
**UNDOABLY-FMAKUNBOUND**  
**UNDOABLY-SETQ**  
**XCL:UNDOABLY-SETF**  
**UNDOABLY-PSETF**  
**UNDOABLY-SETF-SYMBOL-FUNCTION**  
**UNDOABLY-SETF-MACRO-FUNCTION**

Note: Many destructive Common Lisp functions do not currently have undoable versions, e.g., **CL:NREVERSE**, **CL:SORT**, etc. The current list of undoable functions is saved on the association list **LISPFNS**.

---

## Modifying the UNDO Facility

---

You will usually wish to extend the **UNDO** facility after creating a form whose side effects it might be desirable to undo, for instance a file renaming function.

An undoable version of the function needs to be written. This can be done by explicitly saving previous state information away, or by renaming calls in the function to their undoable equivalent. Undo information should be saved on the history list using **IL:UNDOSAVE**.

You must then hook the undoable version of the function into the undo facility. You do this by either using the **IL:LISPFNS** association list, or in the case of a **SETF** modifier, on the **IL:UNDOABLE-SETF-INVERSE** property of the **SETF** function.

---

### LISPFNS

[Variable]

Contains an association list which maps from destructive operations to their undoable form. Initially this list contains:

**((CL:POP . UNDOABLY-POP)**  
**(CL:PSETF . UNDOABLY-PSETF)**  
**(CL:PUSH . UNDOABLY-PUSH)**  
**(CL:PUSHNEW . UNDOABLY-PUSHNEW)**  
**((CL:REMF) . UNDOABLY-REMF)**  
**(CL:ROTATEF . UNDOABLY-ROTATEF)**  
**(CL:SHIFTF . UNDOABLY-SHIFTF)**  
**(CL:DECF . UNDOABLY-DECF)**  
**(CL:INCF . UNDOABLY-INCF)**  
**(CL:SET . UNDOABLY-SET-SYMBOL)**  
**(CL:MAKUNBOUND . UNDOABLY-MAKUNBOUND)**  
**(CL:FMAKUNBOUND . UNDOABLY-FMAKUNBOUND)**  
**. . . plus the original Interlisp undo associations)**

(XCL:UNDOABLY-SETF PLACE VALUE ...)

[Macro]

Like **CL:SETF** but saves information so it may be undone. **UNDOABLY-SETF** uses undoable versions of the setf function located on the **UNDOABLE-SETF-INVERSE** property of the function being **SETF**ed. Initially these **SETF** names have such a property:

**CL:SYMBOL-FUNCTION**       -       **UNDOABLY-SETF-SYMBOL-FUNCTION**

**CL:MACRO-FUNCTION**       -       **UNDOABLY-SETF-MACRO-FUNCTION**

(UNDOABLY-SETQ &REST FORMS)

[Function]

Typed-in **SETQ**s (and **SETF**s on symbols) are made undoable by substituting a call to **UNDOABLY-SETQ**. **UNDOABLY-SETQ** operates like **SETQ** on lexical variables or those with dynamic bindings; it only saves information on the history list for changes to global, "top-level" values.

(UNDOSAVE UNDOFORM HISTENTRY)

[Function]

Adds the undo information **UNDOFORM** to the **SIDE** property of the history event **HISTENTRY**. If there is no **SIDE** property, one is created. If the value of the **SIDE** property is **NOSAVE**, the information is not saved. **HISTENTRY** specifies an event. If **HISTENTRY=NIL**, the value of **LISPMHIST** is used. If both **HISTENTRY** and **LISPMHIST** are **NIL**, **UNDOSAVE** is a no-op.

The form of **UNDOFORM** is **(FN . ARGS)**. Undoing is done by performing **(APPLY (CAR UNDOFORM) (CDR UNDOFORM))**.

\#UNDOSAVES

[Variable]

The value of **\#UNDOSAVES** is the maximum number of **UNDOFORM**s to be saved for a single event. When the count of **UNDOFORM**s reaches this number, **UNDOSAVE** prints the message **CONTINUE SAVING?**, asking if you want to continue saving. If you answer **NO** or default, **UNDOSAVE** discards the previously saved information for this event, and makes **NOSAVE** be the value of the property **SIDE**, which disables any further saving for this event. If you answer **YES**, **UNDOSAVE** changes the count to -1, which is then never incremented, and continues saving. The purpose of this feature is to avoid tying up large quantities of storage for operations that will never need to be undone.

If **\#UNDOSAVES** is negative, then when the count reaches **(ABS \#UNDOSAVES)**, **UNDOSAVE** simply stops saving without printing any messages or other interactions. **\#UNDOSAVES=NIL** is equivalent to **\#UNDOSAVES=infinity**. **\#UNDOSAVES** is initially **NIL**.

The configuration described here has been found to be a very satisfactory one. You pay a very small price for the ability to undo what you type in, since the interpreted evaluation is simply watched for destructive operations, or if you wish to protect yourself from

malfunctioning in your own programs, you can explicitly call, or have your program rewritten to explicitly call, undoable functions.

## Undoing Out of Order

---

**UNDOABLY-SETF** operates undoably by saving (on the history list) the cell that is to be changed and its original contents. Undoing an **UNDOABLY-SETF** restores the saved contents.

This implementation can produce unexpected results when multiple modifications are made to the same piece of storage and then undone out of order. For example, if you type **(SETF (CAR FOO) 1)**, followed by **(SETF (CAR FOO) 2)**, then undo both events by undoing the most recent event first, then undoing the older event, **FOO** will be restored to its state before either event operated. However if you undo the first event, *then* the second event, **(CAR FOO)** will be **1**, since this is what was in **CAR** of **FOO** before **(UNDOABLY-SETF (CAR FOO) 2)** was executed. Similarly, if you type **(NCONC FOO '(1))**, followed by **(NCONC FOO '(2))**, undoing just **(NCONC FOO '(1))** will remove both **1** and **2** from **FOO**. The problem in both cases is that the two operations are not independent.

In general, operations are always independent if they affect different lists or different sublists of the same list. Undoing in reverse order of execution, or undoing independent operations, is always guaranteed to do the right thing. However, undoing dependent operations out of order may not always have the predicted effect.

## Format and Use of the History List

---

### **LISPXHISTORY**

[Variable]

The Exec currently uses one primary history list, **LISPXHISTORY** for the storing events.

The history list is in the form **(EVENTS EVENT# SIZE MOD)**, where **EVENTS** is a list of events with the most recent event first, **EVENT#** is the event number for the most recent event on **EVENTS**, **SIZE** is the the maximum length **EVENTS** is allowed to grow. **MOD** is the maximum event number to use, after which event numbers roll over. **LISPXHISTORY** is initialized to **(NIL 0 100 1000)**.

The history list has a maximum length, called its time-slice. As new events occur, existing events are aged, and the oldest events are forgotten. The time-slice can be changed with the function **CHANGESLICE**. Larger time-slices enable longer memory spans, but tie up correspondingly greater amounts of storage. Since a user seldom needs really ancient history, a relatively small time-slice such as 30 events is usually adequate, although some users prefer to set the time-slice as large as 200 events.

Each individual event on **EVENTS** is a list of the form **(INPUT ID VALUE . PROPS)**. For Exec events, **ID** is a list **(EVENT-NUMBER EXEC-ID)**. The **EVENT-NUMBER** is the number of the event, while the **EXEC-ID** is a string that uniquely identifies the Exec. (The **EXEC-ID** is used to identify which events belong to the "same"

Exec.) *VALUE* is the (first) value of the event. *PROPS* is a property list used to associate other information with the event (described below).

*INPUT* is the input sequence for the event. Normally, this is just the input that the user typed-in. For an APPLY-format input this is a list consisting of two expressions; for an EVAL-format input, this is a list of just one expression; for an input entered as list of atoms, *INPUT* is simply that list. For example,

|                                       |                                       |
|---------------------------------------|---------------------------------------|
| User Input                            | <i>INPUT</i> is:                      |
| <b>LIST(1 2)</b>                      | <b>(LIST (1 2))</b>                   |
| <b>(LIST 1 1)</b>                     | <b>((LIST 1 1))</b>                   |
| <b>DIR "{DSK}&lt;LISPFILES&gt;"cr</b> | <b>(DIR "{DSK}&lt;LISPFILES&gt;")</b> |

If you type in an Exec command that executes other events (**REDO**, **USE**, etc.), several events might result. When there is more than one input, they are wrapped together into one invocation of the **DO-EVENTS** command.

The same convention is used for representing multiple inputs when a **USE** command involves sequential substitutions. For example, if you type **FBOUNDP(FOO)** and then **USE FIE FUM FOR FOO**, the input sequence that will be constructed is **DO-EVENTS (EVENT FBOUNDP (FIE)) (EVENT FBOUNDP (FUM))**, which is the result of substituting **FIE** for **FOO** in **(FBOUNDP (FOO))** concatenated with the result of substituting **FUM** for **FOO** in **(FBOUNDP (FOO))**.

*PROPS* is a property list of the form (*PROPERTY VALUE PROPERTY VALUE ...*), that can be used to associate arbitrary information with a particular event. Currently, the following properties are used by the Exec:

|                     |                                                                                    |
|---------------------|------------------------------------------------------------------------------------|
| <b>SIDE</b>         | A list of the side effects of the event. See <b>UNDOSAVE</b> .                     |
| <b>*LISPXPRINT*</b> | Used to record calls to <b>EXEC-FORMAT</b> , and printed by the <b>??</b> command. |

## Making or Changing an Exec

---

|                                                      |            |
|------------------------------------------------------|------------|
| <b>(XCL:ADD-EXEC &amp;KEY PROFILE REGION TTY ID)</b> | [Function] |
|------------------------------------------------------|------------|

Creates a new process and window with an Exec running in it. *PROFILE* is the type of the Exec to be created (see below under XCL:SET-EXEC-TYPE). *REGION* optionally gives the shape and location of the window to be used. If not provided the user will be prompted. *TTY* is a flag, which, if true, causes the tty to be given to the new Exec process. *ID* is a string identifier to use for events generated in this exec. *ID* defaults to the number given to the Exec process created.

---

|                                                                                                           |            |
|-----------------------------------------------------------------------------------------------------------|------------|
| <b>(XCL:EXEC &amp;KEY WINDOW PROMPT COMMAND-TABLES ENVIRONMENT PROFILE TOP-LEVEL-P TITLE FUNCTION ID)</b> | [Function] |
|-----------------------------------------------------------------------------------------------------------|------------|

This is the main entry to the Exec. The arguments are:

*WINDOW* defaults to the current TTY display stream, or can be provided a window in which the Exec will run.

*PROMPT* is the prompt to print.

*COMMAND-TABLES* is a list of hash-tables for looking up commands (e.g., **\*EXEC-COMMAND-TABLE\*** or **\*DEBUGGER-COMMAND-TABLE\***).

*ENVIRONMENT* is a lexical environment used to evaluate things in.

*READTABLE* is the default readtable to use (defaults to the "Common Lisp" readtable).

*PROFILE* is a way to set the Exec's type (see above, "Multiple Execs and the Exec's Type").

*TOP-LEVEL-P* is a boolean, which should be true if this Exec is at the top level.

*TITLE* is an identifying title for the window title of the Exec.

*FUNCTION* is a function used to actually evaluate events, default is **EVAL-INPUT**.

*ID* is a string identifier to use for events generated in this Exec. *ID* defaults to the number given to the Exec process.

---

---

**XCL:\*PER-EXEC-VARIABLES\***

[Variable]

A list of pairs of the form (*VAR INIT*). Each time an Exec is entered, the variables in **\*PER-EXEC-VARIABLES\*** are rebound to the value returned by evaluating *INIT*. The initial value of **\*PER-EXEC-VARIABLES\*** is:

```
((*PACKAGE* *PACKAGE*)
 (* *)
 (** **)
 (***) (***)
 (+ +)
 (++) ++
 (+++ +++)
 (- -)
 (/ /)
 (// //)
 (/// ///)
 (HELPFLAG T)
 (*EVALHOOK* NIL)
 (*APPLYHOOK* nil)
 (*ERROR-OUTPUT* *TERMINAL-IO*)
 (*READTABLE* *READTABLE*)
 (*package* *package*)
 (*eval-function* *eval-function*)
 (*exec-prompt* *exec-prompt*)
 (*debugger-prompt* *debugger-prompt*))
```

Most of these cause the values to be (re)bound to their current value in any inferior Exec, or to **NIL**, their value at the "top level".

---

---

**XCL:\*EVAL-FUNCTION\***

[Variable]

Bound to the function used by the Exec to evaluate input. Typically in an INTERLISP Exec this is **IL:EVAL**, and in a Common Lisp Exec, **CL:EVAL**.

---

**XCL:\*EXEC-PROMPT\***

[Variable]

Bound to the string printed by the Exec as a prompt for input. Typically in an INTERLISP Exec this is " ← ", and in a Common Lisp Exec, ">".

**XCL:\*DEBUGGER-PROMPT\***

[Variable]

Bound to the string printed by the debugger Exec as a prompt for input. Typically in an INTERLISP Exec this is " ← : ", and in a Common Lisp Exec, ": ".

**(XCL:EXEC-EVAL FORM &OPTIONAL ENVIRONMENT)**

[Function]

Evaluates *FORM* (using **EVAL**) in the lexical environment *ENVIRONMENT* the same as though it were typed in to **EXEC**, i.e., the event is recorded, and the evaluation is made undoable by substituting the UNDOABLE-functions for the corresponding destructive functions. **XCL:EXEC-EVAL** returns the value(s) of the form, but does not print it, and does not reset the variables \*, \*\*, \*\*\*, etc.

**(XCL:EXEC-FORMAT CONTROL-STRING &REST ARGUMENTS)**

[Function]

In addition to saving inputs and values, the Exec saves many system messages on the history list. For example, **FILE CREATED ...**, **FN redefined**, **VAR reset**, output of **TIME**, **BREAKDOWN**, **ROOM**, save their output on the history list, so that when ?? prints the event, the output is also printed. The function **XCL:EXEC-FORMAT** can be used in user code similarly. **XCL:EXEC-FORMAT** performs (APPLY #'CL:FORMAT \*TERMINAL-IO\* *CONTROL-STRING ARGUMENTS*) and also saves the format string and arguments on the history list associated with the current event.

**(XCL:SET-EXEC-TYPE NAME)**

[Function]

Sets the type of the current Exec to that indicated by *NAME*. This can be used to set up the Exec to your liking. *NAME* may be an atom or string. Possible names are:

**INTERLISP, IL**

\*READTABLE\* INTERLISP

\*PACKAGE\* INTERLISP

XCL:\*DEBUGGER-PROMPT\* "←: "

XCL:\*EXEC-PROMPT\* "←"

XCL:\*EVAL-FUNCTION\* IL:EVAL

**XEROX-COMMON-LISP, XCL**

\*READTABLE\* XCL

\*PACKAGE\* XCL-USER

XCL:\*DEBUGGER-PROMPT\* ": "

XCL:\*EXEC-PROMPT\* "&gt; "

XCL:\*EVAL-FUNCTION\* CL:EVAL

|                        |                                                                                                                                                                                  |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>COMMON-LISP, CL</b> | <b>*READTABLE*</b> LISP<br><b>*PACKAGE*</b> USER<br>XCL: <b>*DEBUGGER-PROMPT*</b> ": "<br>XCL: <b>*EXEC-PROMPT*</b> "> "<br>XCL: <b>*EVAL-FUNCTION*</b> CL:EVAL                  |
| <b>OLD-INTERLISP-T</b> | <b>*READTABLE*</b> OLD-INTERLISP-T<br><b>*PACKAGE*</b> INTERLISP<br>XCL: <b>*DEBUGGER-PROMPT*</b> "←: "<br>XCL: <b>*EXEC-PROMPT*</b> ": "<br>XCL: <b>*EVAL-FUNCTION*</b> IL:EVAL |

---

**(XCL:SET-DEFAULT-EXEC-TYPE NAME)**

[Function]

Like **XCL:SET-EXEC-TYPE** , but sets the type of Execs created by default, as from the background menu. Initially **XCL**. This can be used in your greet file to set default Execs to your liking.

---

## Editing Exec Input

---

The Exec features an editor for input which provides completion, spelling correction, help facility, and character-level editing. The implementation is borrowed from the Interlisp module **TTYIN**. This section describes the use of the **TTYIN** editor from the perspective of the Exec.

## Editing Your Input

---

Some editing operations can be performed using any of several characters; characters that are interrupts will, of course, not be read, so several alternatives are given. The following characters may be used to edit your input:

|                      |                                                                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CONTROL-A, BACKSPACE | Deletes a character. At the start of the second or subsequent lines of your input, deletes the last character of the previous line.                          |
| CONTROL-W            | Deletes a "word". Generally this means back to the last space or parenthesis.                                                                                |
| CONTROL-Q            | Deletes the current line, or if the current line is blank, deletes the previous line.                                                                        |
| CONTROL-R            | Refreshes the current line. Two in a row refreshes the whole buffer (when doing multiline input).                                                            |
| ESCAPE               | Tries to complete the current word from the spelling list <b>USERWORDS</b> . In the case of ambiguity, completes as far as is uniquely determined, or beeps. |

|                                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| UNDO key (on 1108 and 1186)<br>Middle-blank key (on 1132) | Retrieves characters from the previous non-empty buffer when it is able to; e.g., when typed at the beginning of the line this command restores the previous line you typed; when typed in the middle of a line fills in the remaining text from the old line; when typed following CONTROL-Q or CONTROL-W restores what those commands erased.                                                                                                                                                                                                                                                                                                                                                                                                                 |
| CONTROL-X                                                 | <p>Goes to the end of your input (or end of expression if there is an excess right parenthesis) and returns if parentheses are balanced.</p> <p>If you are already at the end of the input and the expression is balanced except for lacking one or more right parentheses, CONTROL-X adds the required right parentheses to balance and returns.</p> <p>During most kinds of input, lines are broken, if possible, so that no word straddles the end of the line. The pseudo-carriage return ending the line is still read as a space, however; i.e., the program keeps track of whether a line ends in a carriage return or is merely broken at some convenient point. You will not get carriage returns in your strings unless you explicitly type them.</p> |

## Using the Mouse

|                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                    | Editing with the mouse during TTYIN input is slightly different than with other modules. The mouse buttons are interpreted as follows during TTYIN input:                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>LEFT</i>                        | Moves the caret to where the cursor is pointing. As you hold down <i>LEFT</i> , the caret moves around with the cursor; after you let up, any type-in will be inserted at the new position.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>MIDDLE</i> or <i>LEFT+RIGHT</i> | Like <i>LEFT</i> , but moves only to word boundaries.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <i>RIGHT</i>                       | <p>Deletes text from the caret to the cursor, either forward or backward. While you hold down <i>RIGHT</i>, the text to be deleted is inverted; when you let up, the text goes away. If you let up outside the scope of the text, nothing is deleted (this is how to cancel this operation).</p> <p>If you hold down <i>MOVE</i>, <i>COPY</i>, <i>SHIFT</i> or <i>CTRL</i> while pressing the mouse buttons, you instead get secondary selection, move selection or delete selection. The selection is made by holding the appropriate key down while pressing the mouse buttons <i>LEFT</i> (to select a character) or <i>MIDDLE</i> (to select a word), and optionally extend the selection either left or right using <i>RIGHT</i>. While you are doing this, the caret does not move, but the selected text is highlighted in a manner indicating what is about to happen. When the selection is complete, release the mouse buttons and then lift up on <i>MOVE/COPY/CTRL/SHIFT</i> and the appropriate action will occur:</p> |
| <i>COPY</i> or <i>SHIFT</i>        | The selected text is inserted as if it were typed. The text is highlighted with a broken underline during selection.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <i>CTRL</i>                        | The selected text is deleted. The text is complemented during selection.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <i>MOVE</i> or <i>CTRL+SHIFT</i>   | Combines copy and delete. The selected text is moved to the caret.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |



You can cancel a selection in progress by pressing *LEFT* or *MIDDLE* as if to select, and moving outside the range of the text.

The most recent text deleted by mouse command can be inserted at the caret by typing the UNDO key (on the Xerox 1108/1186/1185) or the Middle-blank key (on the Xerox 1132). This is the same key that retrieves the previous buffer when issued at the end of a line.

## Editing Commands

---

A number of characters have special effects while typing to the Exec. Some of them merely move the caret inside the input stream. While caret positioning can often be done more conveniently with the mouse, some of the commands, such as the case changing commands, can be useful for modifying the input.

In the descriptions below, current word means the word the cursor is under, or if under a space, the previous word. Currently, parentheses are treated as spaces, which is usually what you want, but can occasionally cause confusion in the word deletion commands. The notation *[CHAR]* means meta-*CHAR*. The notation \$ stands for the ESCAPE/EXPAND key. Most commands can be preceded by numbers or escape (means infinity), only the first of which requires the meta key (or the edit prefix). Some commands also accept negative arguments, but some only look at the magnitude of the argument. Most of these commands are confined to work within one line of text unless otherwise noted.

### Cursor Movement Commands

---

|                            |                                                                                                                                                                                                      |
|----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| [bs]                       | Backs up one (or n) characters.                                                                                                                                                                      |
| [space]                    | Moves forward one (or n) characters.                                                                                                                                                                 |
| [^]                        | Moves up one (or n) lines.                                                                                                                                                                           |
| [lf]                       | Moves down one (or n) lines.                                                                                                                                                                         |
| [l]                        | Moves back one (or n) words.                                                                                                                                                                         |
| [d]                        | Moves ahead one (or n) words.                                                                                                                                                                        |
| [tab]                      | Moves to end of line; with an argument moves to nth end of line; [\$tab] goes to end of buffer.                                                                                                      |
| [control-L]                | Moves to start of line (or nth previous, or start of buffer).                                                                                                                                        |
| [{] and [}]                | Goes to start and end of buffer, respectively (like [\$control-L] and [\$tab]).                                                                                                                      |
| [ [ ] (meta-left-bracket)  | Moves to beginning of the current list, where cursor is currently under an element of that list or its closing paren. (See also the auto-parenthesis-matching feature below under "Assorted Flags".) |
| [ ] ] (meta-right-bracket) | Moves to end of current list.                                                                                                                                                                        |
| [Sx]                       | Skips ahead to next (or nth) occurrence of character x, or rings the bell.                                                                                                                           |
| [Bx]                       | Backward search, i.e., short for [-S] or [-nS].                                                                                                                                                      |

---

**Buffer Modification Commands**


---

- [Zx] Zaps characters from cursor to next (or nth) occurrence of x. There is no unzap command.
- [A] or [R] Repeats the last S, B, or Z command, regardless of any intervening input.
- [K] Kills the character under the cursor, or n chars starting at the cursor.
- [cr] When the buffer is empty is the same as undo i.e. restores buffer's previous contents. Otherwise is just like a <cr> (except that it also terminates an insert). Thus, [<cr><cr>] will repeat the previous input (as will undo<cr> without the meta key).
- [O] Does "Open line", inserting a crlf after the cursor, i.e., it breaks the line but leaves the cursor where it is.
- [T] Transposes the characters before and after the cursor. When typed at the end of a line, transposes the previous two characters. Refuses to handle odd cases, such as tabs.
- [G] Grabs the contents of the previous line from the cursor position onward. [nG] grabs the nth previous line.
- [L] Puts the current word, or n words on line, in lower case. [\$L] puts the rest of the line in lower case; or if given at the end of line puts the entire line in lower case.
- [U] Analogous to [L], for putting word, line, or portion of line in upper case.
- [C] Capitalizes. If you give it an argument, only the first word is capitalized; the rest are just lowercased.
- [control-Q] Deletes the current line. [\$control-Q] deletes from the current cursor position to the end of the buffer. No other arguments are handled.
- [control-W] Deletes the current word, or the previous word if sitting on a space.

---

**Miscellaneous Commands**


---

- [P] Prettyprints buffer. Clears the buffer and reprints it using prettyprint. If there are not enough right parentheses, it will supply more; if there are too many, any excess remains unprettyprinted at the end of the buffer. May refuse to do anything if there is an unclosed string or other error trying to read the buffer.
- [N] Refreshes line. Same as control-R. [\$N] refreshes the whole buffer; [nN] refreshes n lines. Cursor movement in TTYIN depends on TTYIN being the only source of output to the window; in some circumstances, you may need to refresh the line for best results.
- [control-Y] Gets an Interlisp Exec.
- [\$control-Y] Gets an Interlisp Exec, but first unread the contents of the buffer from the cursor onward. Thus if you typed at TTYIN something destined for Interlisp, you can do [control-L\$control-Y] and give it to Lisp.
- [←] Adds the current word to the spelling list **USERWORDS**. With zero argument, removes word. See **TTYINCOMPLETEFLG**.

## Useful Macros

---

If the event is considered short enough, the Exec command **FIX** will load the buffer with the event's input, rather than calling the structure editor. If you really wanted the Lisp editor for your fix, you can say **FIX EVENT - [TTY:]**.

## ?= Handler

---

Typing the characters `?=<cr>` displays the arguments to the function currently in progress. Since TTYIN wants you to be able to continue editing the buffer after a `?=`, it prints the arguments below your type-in and then puts the cursor back where it was when `?=` was typed.

## Assorted Flags

---

These flags control aspects of TTYIN's behavior. Some have already been mentioned. In Interlisp-D, the flags are all initially set to **T**.

---

**?ACTIVATEFLG****[Variable]**

If true, enables the feature whereby `?` lists alternative completions from the current spelling list.

---

---

**SHOWPARENFLG****[Variable]**

If true, then whenever you are typing Lisp input and type a right parenthesis, TTYIN will briefly move the cursor to the matching parenthesis, assuming it is still on the screen. The cursor stays there for about 1 second, or until you type another character (i.e., if you type fast you will never notice it).

---

---

**USERWORDS****[Variable]**

**USERWORDS** contains words you mentioned recently: functions you have defined or edited, variables you have set or evaluated at the executive level, etc. This happens to be a very convenient list for context-free escape completion; if you have recently edited a function, chances are good you may want to edit it again (typing "ED(xx\$)") or type a call to it. If there is no completion for the current word from **USERWORDS**, or there is more than one possible completion, TTYIN beeps. If typed when not inside a word, Escape completes to the value of **LASTWORD**, i.e., the last thing you typed that the Exec noticed, except that Escape at the beginning of the line is left alone (it is an Old Interlisp Exec command).

If you really wanted to enter an escape, you can, of course, just quote it with a CONTROL-V, like you can other control characters.

You may explicitly add words to **USERWORDS** yourself that would not get there otherwise. To make this convenient online the edit command `[←]` means "add the current atom to **USERWORDS**" (you might think of the command as pointing out this atom). For

example, you might be entering a function definition and want to point to one or more of its arguments or prog variables. Giving an argument of zero to this command will instead remove the indicated atom from **USERWORDS**.

Note that this feature loses some of its value if the spelling list is too long, if there are too many alternative completions for you to get by with typing a few characters followed by escape. Lisp's maintenance of the spelling list **USERWORDS** keeps the temporary section (which is where everything goes initially unless you say otherwise) limited to **#USERWORDS** atoms, initially 100. Words fall off the end if they haven't been used (they are used if **FIXSPELL** corrects to one, or you use <escape> to complete one).

[This page intentionally left blank]

## APPENDIX B. SEDIT—THE LISP EDITOR

---

SEdit is the Lisp structure editor. It allows you to edit Lisp code directly in memory. This editor replaces DEdit in Chapter 16, Structure Editor, of the *Interlisp-D Reference Manual*. First introduced in Lyric, the SEdit structure editor has been greatly enhanced in the Medley release. Medley additions are indicated with revision bars in the right margin.

---

### 16.1 SEdit - The Structure Editor

---

As a structure editor, SEdit alters Lisp code directly in memory. The effect this has on the running system depends on what is being edited.

For Common Lisp definitions, SEdit always edits a copy of the object. For example, with functions, it edits the definition of the function. What the system actually runs is the installed function, either compiled or interpreted. The primary difference between the definition and the installed function is that comment forms are removed from the definition to produce the installed function. The changes made while editing a function will not be installed until the edit session is complete.

For Interlisp functions and macros, SEdit edits the actual structure that will be run. An exception to this is an edit of an EXPR definition of a compiled function. In this case, changes are included and the function is unsaved when the edit session is completed.

SEdit edits all other structures, such as variables and property lists, directly. SEdit installs all changes as they are made.

If an error is made during an SEdit session, abort the edit with an Abort command (see Section 16.1.7, Command Keys). This command undoes all changes from the beginning of the edit session and exits from SEdit without changing your environment.

If the definition being edited is redefined while the edit window is open, SEdit redisplay the new definition. Any edits on the old definition will be lost. If SEdit was busy when the redefinition occurred, the SEdit window will be gray. When SEdit is no longer busy, position the cursor in the SEdit window and press the left mouse button; SEdit will get the new definition and display it.

### 16.1.1 An Edit Session

---

The List Structure Editor discussion in Chapter 3, Language Integration, explains how to start an editor in Lisp.

Whenever you call SEdit, a new SEdit window is created. This SEdit window has its own process, and thus does not rely on an Exec to run in. You can make edits in the window, shrink it while you do something else, expand it and edit some more, and finally close the window when you are done.

Throughout an edit session, SEdit remembers everything that you do through a change history. All edits can be undone and redone sequentially. When an edit session ends, SEdit forgets this information and installs the changes in the system.

The session ends with an event signalling to the editor that changes are complete. Three events signal completion:

- Closing the window.

Do this to terminate the edit session when you are finished.

- Shrinking the window.

Shrink the window when you have made some edits and may want to continue the editing session at a later time.

- Typing one of the Completion Commands, listed below.

Each of these commands has the effect of installing your changes, completing the edit, and returning the TTY process to the Exec. They vary in what is done in addition to completing. Using these commands the definition that you were editing can be automatically compiled, the edit window can be closed, or both.

A new edit session begins when you come back to an SEdit after completing. The change history is discarded at this point.

If the Exec is waiting for SEdit to return before going on, complete the edit session using any of the methods above to alert the Exec that SEdit is done. The TTY process passes back to the Exec.

### 16.1.2 SEdit Carets

---

There are two carets in SEdit, the edit caret and the structure caret. The edit caret appears when characters are edited within a single structure, such as an atom, string, or comment. Anything typed in will appear at the edit caret as part of the structure that the caret is within. The edit caret looks like this:

(a  b)

The structure caret appears when the edit point is between structures, so that anything inserted will go into a new structure. It looks like this:

(a  b)

SEdit changes the caret frequently, depending on where you are in the structure you are editing, and how the caret is positioned. The left mouse button allows an edit caret position to be set. The middle mouse button allows the structure caret position to be set.

### 16.1.3 The Mouse

In SEdit, the mouse buttons are used as follows. The left mouse button positions the mouse cursor to point to parts of Lisp structures. The middle mouse button positions the mouse cursor to point to whole Lisp structures. Thus, selecting the Q in LEQ using the left mouse button selects that character, and sets the edit caret after the Q:

```
(LEQ▲ n 1)
```

Any characters typed in at this point would be appended to the atom LEQ.

Selecting the same letter using the middle mouse button selects the whole atom (this convention matches TEdit's character/word selection convention), and sets a structure caret between the LEQ and the n:

```
(LEQ▲ n 1)
```

At this point, any characters typed in would form a new atom between the LEQ and the n.

Larger structures can be selected in two ways. Use the middle mouse button to position the mouse cursor on the parenthesis of the desired list to select that list. Press the mouse button multiple times, without moving the mouse, extends the selection. Using the previous example, if the middle button were pressed twice, the list (LEQ ...) would be selected:

```
(LEQ n 1)
```

Pressing the button a third time would cause the list containing the (LEQ n 1) to be selected.

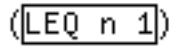
The right mouse button positions the mouse cursor for selecting sequences of structures or substructures. Extended selections are indicated by a box enclosing the structures selected. The selection is extended in the same mode as the original selection. That is, if the original selection were a character selection, the right button could be used to select more characters in the same atom. Extended selections also have the property of being marked for pending deletion. That is, the selection takes the place of the caret, and anything typed in is inserted in place of the selection.

For example, selecting the E by pressing the left mouse button and selecting the Q by pressing the right mouse button would produce:

```
(LEQ□ n 1)
```



Similarly, pressing the middle mouse button and then selecting with the right mouse button extends the selection by whole structures. Thus, in our example, pressing the middle mouse button to select LEQ and pressing the right mouse button to select the 1 would produce:



This is not the same as selecting the entire list, as above. Instead, the elements in the list are collectively selected, but the list itself is not.

---

### 16.1.4 Gaps

The SEdit structure editor requires that everything edited must have an underlying Lisp structure, even if the structure is not directly displayed. For example, with quoted forms the actual structure might be (**QUOTE** GREEN), although this would be displayed as 'GREEN. Even when the user is in the midst of typing in a form, the underlying Lisp structure must exist.

Because of this necessity, SEdit provides gaps to serve as dummy Lisp objects during typing. SEdit does not need a gap for every form typed in, but gaps are necessary for quoted objects. When something is typed that requires SEdit to build a Lisp structure and thus create a gap, as the quote character does, the gap will appear marked for pending deletion. This means it is ready to be replaced by the structure to be typed in. In this way it is possible to type special structures, like quotes, directly, while SEdit maintains the structure.

A gap looks like: -x-

A gap displayed after a quote has been typed in would look like this:



with the gap marked for pending deletion, ready for typein of the object to be quoted.

---

### 16.1.5 Broken Atoms

When you are typing an atom (a symbol or a number), SEdit saves the characters you type until you finish the atom. SEdit determines that you've finished the atom when you type a character that cannot (without being escaped) belong to an atom, such as a space or open parenthesis. SEdit then tries to create an atom with these characters, just as if it were the Lisp reader. If it succeeds, the atom becomes part of the structure you're editing. However, if it fails, SEdit intercepts the reader error that would otherwise occur and instead creates a special SEdit structure called a Broken-Atom. A Broken-Atom looks and behaves in SEdit just like a normal atom, but is printed in italics to alert you to its needing correction.

SEdit has to create a Broken-Atom when the characters typed don't make a legal atom. For example, the characters "DECLARE:" cannot make a symbol because the colon is a package specifier,

but the form is not correct for a package-qualified symbol. Similarly, the characters "#b123" cannot represent an integer in base two, because 2 and 3 are not legal digits in base two, so SEdit would make a Broken-Atom that looks like #b123.

Broken-Atoms can be edited in SEdit just like real atoms. Whenever you finish editing a Broken-Atom, SEdit again tries to create an atom from the characters. If it succeeds, it reprints the atom in SEdit's default font, rather than in italics. You should be sure to correct any Broken-Atoms you create before exiting SEdit, since Broken-Atoms do not behave in any useful way outside SEdit.

### 16.1.6 Special Characters

A few characters have special meaning in Lisp, and are treated specially by SEdit. SEdit must always have a complete structure to work on at any level of the edit. This means that SEdit needs a special way to type in structures such as lists, strings, and quoted objects. In most instances these structures can be typed in just as they would be to a regular Exec, but in a few cases this is not possible.

#### Lists- ( and )

Lists begin with an open parenthesis character ( . Typing an open parenthesis gives a balanced list, that is, SEdit inserts both an open and a close parenthesis. The structure caret is between the two parentheses. List elements can be typed in at the structure caret. When a close parenthesis, ) is typed, the caret will be moved outside the list (and the close parenthesis), effectively finishing the list. Square bracket characters, [ and ], have no special meaning in SEdit, as they have no special meaning in Common Lisp.

#### Quoted Structures:

SEdit handles the quote keys so that it is possible to type in all quote forms directly. When typing one of the following quote keys at a structure caret, the quote character typed will appear, followed by a gap to be replaced by the object to be quoted.

##### Single Quote – ’

Use to enter quoted structures.

##### Backquote – `

Use to enter backquoted structures.

##### Comma – ,

Use to enter comma forms, as used with a Backquote form.

##### At Sign – @

Use after a comma to create a comma-at-sign gap. This allows type-in of comma-at forms, e.g. ,@list, as used within a Backquote form.

##### Dot – .

Use the dot (period) after a comma to create a comma-dot gap. This allows type-in of comma-dot forms, e.g. ,.list, as used within a Backquote form.

##### Hash Quote – #'

Use this two character sequence to enter the **CL:FUNCTION** abbreviation hash-quote (#').

#### Dotted Lists:

The dot, or period, character (.) is used to type dotted lists in SEdit. After typing a dot, SEdit inserts a dot and a gap to fill in for the tail of the list. To dot an existing list, point the cursor between the last and second to the last element in the list, and type a dot. To undot a list, select the tail of the list before the dot while holding down the SHIFT key.

**Escape- \ or %**

Use to escape from a specific typed in character. Use the escape key to enter characters, like parentheses, which otherwise have special meaning to the SEdit reader. Press the escape key then type in the character to escape. SEdit uses the escape key appropriate to the environment it is editing in; it depends on the readtable that was current when the editor was started. The backslash key (\) is used when editing Common Lisp, and the percent key (%) is used when editing Interlisp.

**Multiple Escape- |**

Use the multiple escape key, the vertical bar character (|), to escape a sequence of typed in characters. SEdit always balances multiple escape characters. When one multiple escape character is typed, SEdit produces a balanced pair, with the caret between them, ready for typing in the characters to be escaped. If you type a second vertical bar, the caret moves after the second vertical bar, and is still within the same atom, so that you can add more unescaped characters to the atom.

**Comments- ;**

The comment key, a semicolon (;), starts a comment. When a semicolon is typed, an empty comment is inserted with the caret in position for typing in the comment. Comments can be edited like strings. There are three levels of comments supported by SEdit: single, double, and triple. Single semicolon comments are formatted at the comment column, about three-quarters of the way across the SEdit window, towards the right margin. Double semicolon comments are formatted at the current indentation of the code that they are in. Triple semicolon comments are formatted against the left margin of the SEdit window. The level of a comment can be increased or decreased by pointing after the semicolon, and either typing another semicolon, or backspacing over the preceding semicolon. Comments can be placed anywhere in your Common Lisp code. However, in Interlisp code, they must follow the placement rules for Interlisp comments.

**Strings- "**

Enter strings in SEdit by typing a double quote ("). SEdit balances the double quotes. When one is typed, SEdit produces a second, with the caret between the two, ready for typing the characters of the string. If a double quote character is typed in the middle of a string, SEdit breaks the string into two smaller strings, leaving the caret between them.

---

**16.1.7 Commands**

---

SEdit commands are most easily entered through the keyboard. When possible, SEdit uses a named key on the keyboard, for example, the DELETE key. The other commands are either Meta, Control, or Meta-Control key combinations. For the alphabetic command keys, either uppercase or lowercase will work.

There are two menus available, as an alternative means of invoking commands. They are the middle button popup menu, and the attached command menu. These menus are described in more detail below.

---

**16.1.8 Editing Commands**

---

Redisplay: Control-L

[Editor Command]

|                                 |                                                                                                                                             |                  |
|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <b>Delete Selection: DELETE</b> | Redisplays the structure being edited.                                                                                                      | [Editor Command] |
| <b>Delete Word: Control-W</b>   | Deletes the current selection.                                                                                                              | [Editor Command] |
| <b>Control-Meta-O</b>           | Deletes the previous atom or whole structure. If the caret is in the middle of an atom, deletes backward to the beginning of the atom only. | [Editor Command] |
|                                 | Performs a fast edit by calling ED with its CURRENT option.                                                                                 |                  |

### 16.1.9 Completion Commands

|                       |                                                                                                                                                                                                              |                  |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <b>Abort: Meta-A</b>  |                                                                                                                                                                                                              | [Editor Command] |
|                       | Aborts. This command must be confirmed. All changes since the beginning of the edit session are undone, and the edit is closed.                                                                              |                  |
|                       | The following commands signal completion of an edit session and install the structure you were editing.                                                                                                      |                  |
| <b>Control-X</b>      |                                                                                                                                                                                                              | [Editor Command] |
|                       | Signals the system that this edit is complete. The window remains open, though, so the user can see the edit and start editing again directly.                                                               |                  |
| <b>Control-C</b>      |                                                                                                                                                                                                              | [Editor Command] |
|                       | Signals the system that this edit is complete and compiles the definition being edited. The variable *compile-fn* determines the function to be called to do the compilation. See the Options section below. |                  |
| <b>Control-Meta-X</b> |                                                                                                                                                                                                              | [Editor Command] |
|                       | Signals the system that this edit is complete and closes the window.                                                                                                                                         |                  |
| <b>Control-Meta-C</b> |                                                                                                                                                                                                              | [Editor Command] |
|                       | Signals the system that this edit is complete, compiles the definition being editing, and closes the window.                                                                                                 |                  |

### 16.1.10 Undo Commands

|                              |                                                                                                                                                  |                  |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| <b>Undo: Meta-U or UNDO</b>  |                                                                                                                                                  | [Editor Command] |
|                              | Undoes the last edit. All changes since the beginning of the edit session are remembered, and can be undone sequentially.                        |                  |
| <b>Redo: Meta-R or AGAIN</b> |                                                                                                                                                  | [Editor Command] |
|                              | Redoes the edit change that was just undone. Redo only works directly following an Undo. Any number of Undo commands can be sequentially redone. |                  |

### 16.1.11 Find Commands

|                             |  |                  |
|-----------------------------|--|------------------|
| <b>Find: Meta-F or FIND</b> |  | [Editor Command] |
|-----------------------------|--|------------------|

Finds a specified structure, or sequence of structures. If there is a current selection, SEdit looks for the next occurrence of the selected structure. If there is no selection, SEdit prompts for the structure to find, and searches forward from the position of the caret. The found structure will be selected, so the Find command can be used to easily find the same structure again.

If a sequence of structures is selected, SEdit will look for the next occurrence of the same sequence. Similarly, when SEdit prompts for the structure to find, you can type a sequence of structures to look for.

The variable *\*wrap-search\** controls whether or not SEdit wraps around from the end of the structure being edited and continues searching from the beginning.

---

**Reverse Find: Control-Meta-F**[Editor Command]

---

Finds a specified structure, searching in reverse from the position of the caret.

The variable *\*wrap-search\** controls whether or not SEdit wraps around from the beginning of the structure being edited and continues searching from the end.

---

**Find Gap: Meta-N or SKIP-NEXT**[Editor Command]

---

Skips to the next gap in the structure, leaving it selected for pending deletion.

---

**Substitute: Meta-S or SHIFT-FIND**[Editor Command]

---

Substitutes one structure, or sequence of structures, for another structure, or sequence, within the current selection. SEdit prompts you in the SEdit prompt window for the structures to replace, and the structures to replace with.

The selection to substitute within must be a structure selection. To get a structure selection, click with the middle mouse button (not the left), and extend it, if necessary, with the right mouse button. If you begin with the left button, you will get an informational message "Select the structure to substitute within", because the selection was of characters, rather than structures.

---

**Delete Structure: Control-Meta-S**[Editor Command]

---

Removes all occurrences of a structure or sequence of structures within the current selection. SEdit prompts the user in the SEdit prompt window for the structures to delete.

---

## 16.1.12 General Commands

---

---

**Arglist: Meta-H or HELP**[Editor Command]

---

Shows the argument list for the function currently selected, or currently being typed in, in the SEdit prompt window. If the argument list will not fit in the SEdit prompt window, it is displayed in the main Prompt Window.

---

**Convert Comments: Meta-;**[Editor Command]

---

Converts old style comments in the selected structure to new style comments. This converter notices any list that begins with an asterisk (\*) in the INTERLISP package (IL:\*) as an old style comment. Section 16.1.18, Options, describes the converter options.

**Comment Out Selection: Control-Meta-;** [Editor Command]

This command puts the contents of a structure selection into a comment. This provides an easy way to "comment out" a chunk of code. The Extract command can be used to reverse this process, returning the comment to the structures contained therein.

**Edit: Meta-O** [Editor Command]

Edits the definition of the current selection. If the selected name has more than one type of definition, SEdit asks for the type to be edited. If the selection has no definition, a menu pops up. This menu lets the user specify either the type of definition to be created, or no definition if none needs to be created.

**Eval: Meta-E** [Editor Command]

Evaluates the current selection. If the result is a structure, the inspector is called on it, allowing the user to choose how to look at the result. Otherwise, the result is printed in the SEdit prompt window. The evaluation is done in the process from which the edit session was started. Thus, while editing a function from a break window, evaluations are done in the context of the break.

**Expand: Meta-X or EXPAND** [Editor Command]

Replaces the current selection with its definition. This command can be used to expand macros and translate CLISP.

**Extract: Meta- /** [Editor Command]

Extracts one level of structure from the current selection. If there is no selection, but there is a structure caret, the list containing the caret is used. This command can be used to strip the parentheses off a list, or to unquote a quoted structure, or to replace a comment with the structures contained therein.

**Inspect: Meta-I** [Editor Command]

Inspect the current selection.

**Join: Meta-J** [Editor Command]

Joins. This command joins any number of sequential Lisp objects of the same type into one object of that type. Join is supported for atoms, strings, lists, and comments. In addition, SEdit permits joining of a sequence of atoms and strings, since either type can easily be coerced into the other. In this case, the result of the Join will be an atom if the first object in the selection is an atom, otherwise the result will be a string.

**Mutate: Meta-Z** [Editor Command]

Mutates. This command allows the user to do arbitrary operations on a LISP structure. First select the structure to be mutated (it must be a whole structure, not an extended selection). When the

user presses Meta-Z SEdit prompts for the function to use for mutating. This function is called with the selected structure as its argument, and the structure is replaced with the result of the mutation.

For example, an atom can be put in upper case by selecting the atom and mutating by the function U-CASE. You can replace a structure with its value by selecting it and mutating by EVAL.

**Quote: Meta-'  
Meta-'  
Meta-,  
Meta-.  
Meta-@ or Meta-2  
Meta-# or Meta-3** [Editor Command]

---

Quotes the current selection with the specified kind of quote, respectively, Single Quote, Backquote, Comma, Comma-At-Sign, Comma-Dot, or Hash-Quote.

**Normalize Selection: Meta-Space or Meta-Return** [Editor Command]

---

Scrolls the current selection to the center of the window. Similarly, the Space or Return key can be used to normalize the caret.

**Parenthesize: Meta- ) or Meta-0** [Editor Command]

---

Parenthesizes the current selection, positioning the caret after the new list.

**Parenthesize: Meta- ( or Meta-9** [Editor Command]

---

Parenthesizes the current selection, positioning the caret at the beginning of the new list. Only a whole structure selection or an extended selection of a sequence of whole structures can be parenthesized.

### 16.1.13 Miscellaneous

#### Change Print Base: Meta-B

[Editor Command]

Changes Print Base. Prompts for entry of the desired Print Base, in decimal. SEdit redisplay fixed point numbers in this new base.

#### Set Package: Meta-P

[Editor Command]

Changes the current package for this edit. Prompts the user, in the SEdit prompt window, for a new package name. SEdit will redisplay atoms with respect to that package.

#### Attached Menu: Meta-M

[Editor Command]

Attaches a menu of the commonly used commands (the SEdit Command Menu) to the top of the SEdit window. Each SEdit window can have its own menu, if desired.

### 16.1.14 Help Menu

When the mouse cursor is positioned in the SEdit title bar and the middle mouse button is pressed, a Help Menu of commands pops up. The menu looks like this:

| Commands               |       |
|------------------------|-------|
| Abort                  | M-A   |
| Done                   | C-X   |
| Done & Compile         | C-C   |
| Done & Close           | M-C-X |
| Done, Compile, & Close | M-C-C |
| Undo                   | M-U   |
| Redo                   | M-R   |
| Find                   | M-F   |
| Reverse Find           | M-C-F |
| Remove                 | M-C-S |
| Substitute             | M-S   |
| Find Gap               | M-N   |
| Arglist                | M-H   |
| Convert Comment        | M-;   |
| Edit                   | M-O   |
| Eval                   | M-E   |
| Expand                 | M-X   |
| Extract                | M-/   |
| Inspect                | M-I   |
| Join                   | M-J   |
| Mutate                 | M-Z   |
| Parenthesize           | M-(   |
| Quote                  | M-'   |
| Set Print-Base         | M-B   |
| Set Package            | M-P   |
| Attach Menu            | M-M   |

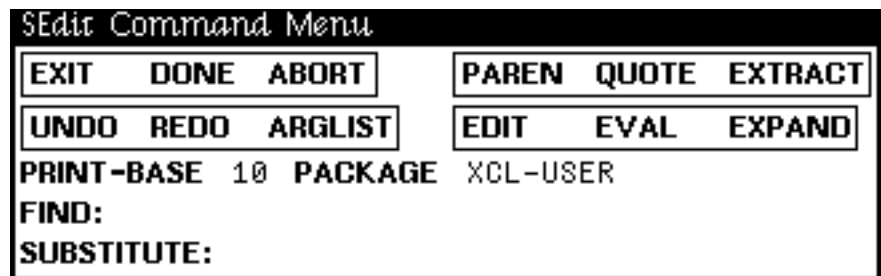


The Help Menu lists each command and its corresponding Command Key. (In the menu, the letter C stands for CONTROL, while M indicates Meta.) The command selected is executed just as if the command had been entered from the keyboard. The menu remembers which command was selected last, and pops up with the mouse cursor next to that same command the next time the menu is used. This provides a very fast way to repeat the same command when using the mouse.

### 16.1.15 Command Menu

---

The SEdit Attached Command Menu contains the commonly used commands. Use the Meta-M keyboard command to bring up this menu. The menu can be closed, independently of the SEdit window, when desired. The menu looks like:



All of the commands in the menu function identically to their corresponding keyboard commands, except for Find and Substitute.

When Find is selected with the mouse cursor, SEdit prompts in the menu window, next to the Find button, for the structures to find. Type in the structures then select Find again. The search begins from the caret position in the SEdit window.

Similarly, Substitute prompts, next to the Find button, for the structures to find, and next to the Substitute button for the structures to substitute with. After both have been typed in, selecting Substitute replaces all occurrences of the Find structures with the Substitute structures, within the current selection.

To do a confirmed substitute, set the edit point before the first desired substitution, and select Find. Then if you want to substitute that occurrence of the structure, select Substitute. Otherwise, select Find again to go on.

Selecting either Find or Substitute with the right mouse button erases the old structure to find or substitute from the menu, and prompts for a new one.

### 16.1.16 SEdit Programmer's Interface

---

The following sections describe SEdit's programmer's interface. All symbols are external in the package named "SEdit".

### 16.1.17 SEdit Window Region Manager

---

SEdit provides user redefinable functions which control how SEdit chooses the region for a new edit window.

**(get-window-region context reason name type)****[Function]**

This function is called when SEdit wants to know where to place a window it is about to open. This happens whenever the user starts a new SEdit or expands an Sedit icon. The default behavior is to pop a window region off SEdit's stack of regions that have been used in the past. If the stack is empty, SEdit prompts for a new region.

This function can be redefined to provide different behavior. It is called with the edit *context*, a *reason* for needing a region, the *name* of the structure to be edited, and the *type* of the structure to be edited. The edit *context* is SEdit's main data structure and can be useful for associating particular edits with specific regions. The *reason* argument specifies why SEdit wants a region, and will be one of the keywords :CREATE or :EXPAND.

**(save-window-region context reason name type region)****[Function]**

This function is called whenever SEdit is finished with a region and wants to make the region available for other SEdits. This happens whenever an SEdit window is closed or shrunk, or when an SEdit icon is closed. The default behavior is simply to push the region onto SEdit's stack of regions.

This function can be redefined to provide different behavior. It is also called with the edit *context*, the *reason*, the *name*, the *type*, and additionally the window *region* that is being released. The *reason* argument specifies why SEdit is releasing the region, and will be one of the keywords :CLOSE, :SHRINK, or :CLOSE-ICON.

**keep-window-region****[Variable]**

Default **T**. This flag determines the behavior of the default SEdit region manager, explained above, for shrinking and expanding windows. When set to **T**, shrinking an SEdit window will not give up that window's region; the icon will always expand back into the same region. When set to **NIL**, the window's region is made available for other SEdits when the window is shrunk. Then when an SEdit icon is expanded, the window will be reshaped to the next available region.

This variable is only used by the default implementations of the functions **get-window-region** and **save-window-region**. If these functions are redefined, this flag is no longer used.

**16.1.18 Options**

The following parameters can be set as desired.

**\*wrap-parens\*****[Variable]**

This SEdit pretty printer flag determines whether or not trailing close parenthesis characters, **)**, are forced to be visible in the window without scrolling. By default it is set to **NIL**, meaning that close parens are allowed to "fall off" the right edge of the window. If set to **T**, the pretty printer will start a new line before the structure preceding the close parens, so that all the parens will be visible.

**\*wrap-search\*****[Variable]**

This flag determines whether or not SEdit find will wrap around to the top of the structure when it reaches the end, or vice versa in the case of reverse find. The default is NIL.

**\*clear-linear-on-completion\*** [Variable]

This flag determines whether or not SEdit completely re-pretty prints the structure being edited when you complete the edit. The default value is NIL, meaning that SEdit reuses the pretty printing.

**\*ignore-changes-on-completion\*** [Variable]

Sometimes the structure that you are editing is changed by the system upon completion. Editdates are an example of this behavior. When this flag is NIL, the default, SEdit will redisplay the new structure, capturing the changes. When T, SEdit will ignore the fact that changes were made by the system and keep the old structure.

**convert-upgrade** [Variable]

Default 100. When using Meta-; to convert old-style single-asterisk comments, if the length of the comment exceeds **convert-upgrade** characters, the comment is converted into a double semicolon comment. Otherwise, the comment is converted into a single semicolon comment.

Old-style double-asterisk comments are always converted into new-style triple-semicolon comments.

---

### 16.1.19 Control Functions

**(reset)** [Function]

This function recomputes the SEdit edit environment. Any changes made in the font profile, or any changes made to SEdit's commands are captured by resetting. Close all SEdit windows before calling this function.

**(add-command *key-code form &optional scroll? key-name command-name help-string*)**

[Function]

This function allows you to write your own SEdit keyboard commands. You can add commands to new keys, or you can redefine keys that SEdit already uses as command keys. If you mistakenly redefine an SEdit command, the function Reset-Commands will remove all user-added commands, leaving SEdit with its default set of commands.

*key-code* can be a character code, or any form acceptable to il:charcode.

*form* determines the function to be called when the key command is typed. It can be a symbol naming a function, or a list, whose first element is a symbol naming a function and the rest of the elements are extra arguments to the function. When the command is invoked, SEdit will apply the function to the edit context (SEdit's main data structure), the charcode that was typed, and any extra arguments supplied in *form*. The extra arguments do not get evaluated, but are useful as keywords or flags, depending on how the command was invoked. The command function must return T if

it handled the command. If the function returns NIL, SEdit will ignore the command and insert the character typed.

The first optional argument, *scroll?*, determines whether or not SEdit scrolls the window after running the command. This argument defaults to NIL, meaning don't scroll. If the value of SCROLL is T, then SEdit will scroll the window to ensure that the caret is visible.

The rest of the optional arguments are used to add this command to SEdit's middle button menu. When the item is selected from the menu, the command function will be called as described above, with the *charcode* argument set to NIL.

*key-name* is a string to identify the key (combination) to be typed to invoke the command. For example "M-A" to represent the Meta-A key combination, and "C-M-A" for Control-Meta-A.

*command-name* is a string to identify the command function, and will appear in the menu next to the *key-name*.

*help-string* is a string to be printed in the prompt window when a mouse button is held down over the menu item.

After adding all the commands that you want, you must call Reset-Commands to install them.

For example:

```
(add-command "↑U" (my-change-case t))
(add-command "↑Y" (my-change-case nil))
(add-command "l,r" my-remove-nil
 "M-R" "Remove NIL"
 "Remove NIL from the selected structure"))
(reset-commands)
```

will add three commands. Suppose *my-change-case* takes the arguments *context*, *charcode*, and *upper-case?*. *upper-case?* will be set to T when *my-change-case* is called from Control-U, and NIL when called from Control-Y. *my-remove-nil* will be called with only *context* and *charcode* arguments when Meta-R is typed.

Below are some SEdit functions which are useful in writing new commands.

### (reset-commands)

[Function]

This function installs all commands added by **add-command**. SEdits which are open at the time of the **reset-commands** will not see the new commands; only new SEdits will have the new commands available.

### (default-commands)

[Function]

This function removes all commands added by **add-command**, leaving SEdit with its default set of commands. As in **reset-commands**, open SEdits will not be changed; only new SEdits will have the user commands removed.

**(get-prompt-window *context*)****[Function]**

This function returns the attached prompt window for a particular SEdit.

**(get-selection *context*)****[Function]**

This function returns two values: the selected structure, and the type of selection, one of NIL, T, or :SUB-LIST. The selection type NIL means there is not a valid selection (in this case the structure is meaningless). T means the selection is one complete structure. :SUB-LIST means a series of elements in a list is selected, in which case the structure returned is a list of the elements selected.

**(replace-selection *context structure selection-type*)****[Function]**

This function replaces the current selection with a new structure, or multiple structures, by deleting the selection and then inserting the new structure(s). The *selection-type* argument must be one of T or :SUB-LIST. If T the *structure* is inserted as one complete structure. If :SUB-LIST, the *structure* is treated as a list of elements, each of which is inserted.

**\*edit-fn\*****[Variable]**

This function is funcalled with the selected structure and the edit options as its arguments from the Edit (M-O) command. It should start the editor as appropriate, or else generate an error if the selection is not editable.

**\*compile-fn\*****[Variable]**

This function is funcalled with the arguments *name*, *type*, and *body*, from the compile completion commands. It should compile the definition, *body*, and install the code as appropriate.

**(sedit *structure props options*)****[Function]**

This function provides a means of starting SEdit directly. *structure* is the structure to be edited.

*props* is a property list, which may specify the following properties:

:name - the name of the object being edited

:type - the file manager type of the object being edited. If NIL, SEdit will not call the file manager when it tries to refetch the definition it is editing. Instead, it will just continue to use the structure that it has.

:completion-fn - the function to be called when the edit session is completed. This function is called with the *context*, *structure*, and *changed?* arguments. *context* is SEdits main data structure. *structure* is the structure being edited. *changed?* specifies if any changes have been made, and is one of NIL, T, or :ABORT, where :ABORT means the user is aborting the edit and throwing away any changes made. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments that the function is applied to following the main arguments above.

:root-changed-fn - the function to be called when the entire structure being edited is replaced with a new structure. This function is called with the new structure as its argument. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments that the function is applied to following the structure argument.

*options* is one or a list of any number of the following keywords:

:close-on-completion - This option specifies that SEdit cannot remain active for multiple completions. That is, the SEdit window cannot be shrunk, and the completion commands that normally leave the window open will in this case close the window and terminate the edit.

:compile-on-completion - This option specifies that SEdit should call the \*compile-fn\* to compile the definition being edited upon completion, regardless of the completion command used.

---

## Warning with Declarations

**CAUTION:** There is a feature of the BYTECOMPILER that is not supported by SEdit or the XCL compiler. It is possible to insert a comment at the beginning of your function that looks like

```
(* DECLARATIONS: --)
```

The tail, or -- section, of this comment is taken as a set of local record declarations which are then used by the compiler in that function just as if they had been declared globally. See the "Compiler" section in Chapter 3 of these Notes for additional behavior in XCL.

SEdit does not recognize such declarations. Thus, if the "Expand" command is used, the expansion will not be done with these record declarations in effect. The code that you see in SEdit will not be the same code compiled by the BYTECOMPILER.

[This page intentionally left blank]

**ICONW**, used to build small windows that will appear as icons on the display, is a standard input/output feature. This feature was introduced in Lyric and has been enhanced in Medley. The following description of **ICONW** should be appended to Section 28.4, Windows, of the *Interlisp-D Reference Manual*. Medley changes are indicated with revision bars in the right margin.

### 28.4.16 Creating Icons with ICONW

---

**ICONW** is a group of functions available for building small windows of arbitrary shape. These windows are principally for use as icons for shrinking windows; i.e., these functions are likely to be invoked from within the **ICONFN** of a window. An icon is specified by supplying its image (a bitmap) and a mask that specifies its shape. The mask is a bitmap of the same dimensions as the image whose bits are on (black) in those positions considered to be in the image, and off (white) in those positions where the background should show through. By using the mask and appropriate window functions, **ICONW** maintains the illusion that the icon window is nonrectangular, even though the actual window itself is rectangular. The illusion is not complete, of course. For example, if you try to select what looks like the background (or an occluded window) around the icon but still within its rectangular perimeter, the icon window itself is selected. Also, if you move a window occluded by an icon, the icon never notices that the background changed behind it. Icons created with **ICONW** can also have titles; some part of the image can be filled with text computed at the time the icon is created, or text may be changed after creation.

#### 28.4.16.1 Creating Icons

---

Two types of icons can be created with **ICONW**, a borderless window containing an image defined by a mask and a window with a title.

**(ICONW IMAGE MASK POSITION NOOPENFLG)**

[Function]

Creates a window at *POSITION*, or prompts for a position if *POSITION* is **NIL**. The window is borderless, and filled with *IMAGE*, as cookie-cut by *MASK*. If *MASK* is **NIL**, the image is considered rectangular (i.e., *MASK* defaults to a black bitmap of the same dimensions as *IMAGE*). If *NOOPENFLG* is **T**, the window is returned unopened.



(**TITLEDICONW** *ICON TITLE FONT POSITION NOOPENFLG JUST BREAKCHARS  
OPERATION*)

[Function]

Creates a titled icon at *POSITION*, or prompts for a position if *POSITION* is **NIL**. If *NOOPENFLG* is **T**, the window is returned unopened. The argument *ICON* is an instance of the record **TITLEDICON**, which specifies the icon image and mask, as with **ICONW**, and a region within the image to be used for displaying the title. Thus, the *ICON* argument is usually of the form

```
(create TITLEDICON ICON ← someIconImage
 MASK ← iconMask TITLereg ←
 someRegionWithinICON)
```

The title region is specified in coordinates relative to the icon, i.e., the lower-left corner of the image bitmap is (0, 0). The mask can be **NIL** if the icon is rectangular. The image should be white where it is covered by the title region. **TITLEDICONW** clears the region before printing on it. The title is printed into the specified region in the image, using *FONT*. If *FONT* is **NIL** it defaults to the value of **DEFAULTICONFONT**, initially Helvetica 10. The title is broken into multiple lines if necessary; **TITLEDICONW** attempts to place the breaks at characters that are in the list of character codes *BREAKCHARS*. *BREAKCHARS* defaults to (**CHARCODE (SPACE y )**). In addition, line breaks are forced by any carriage returns in *TITLE*, independent of *BREAKCHARS*. *BREAKCHARS* is ignored if a long title would not otherwise fit in the specified region. For convenience, *BREAKCHARS* = **FILE** means the title is a file name, so break at file name field delimiters. The argument *JUST* indicates how the text should be justified relative to the region. It is an atom or list of atoms chosen from **TOP**, **BOTTOM**, **LEFT**, or **RIGHT**, which indicate the vertical positioning (flush to top or bottom) and/or horizontal positioning (flush to left edge or right). If *JUST* = **NIL**, the text is centered. The argument *OPERATION* is a display stream operation indicating how the title should be printed. If *OPERATION* is **INVERT**, then the title is printed white-on-black. The default *OPERATION* is **REPLACE**, meaning black-on-white. **ERASE** is the same as **INVERT**; **PAINT** is the same as **REPLACE**.

For convenience, **TITLEDICONW** can also be used to create icons that consist solely of a title, with no special image. If the argument *ICON* is **NIL**, **TITLEDICONW** creates a rectangular icon large enough to contain *TITLE*, with a border the same width as that on a regular window. The remaining arguments are as described above, except that a *JUST* of **TOP** or **BOTTOM** is not meaningful.

In the Medley release, **TITLEDICONW** can create icons with white text on a black background. To get this effect, your icon image must be black in the correct area, and you must specify the *OPERATION* argument as **INVERT**.

In Medley, you can copy- select the title of an icon.

### 28.4.16.2 Modifying Icons

---

**(ICONW.TITLE *ICON TITLE*)** [Function]

---

Returns the current title of the window *ICON*, which must be a window returned by **TITLEDICONW**. In addition, if *TITLE* is non-**NIL**, makes *TITLE* the new title of the window and repaints it accordingly. To erase the current title, make *TITLE* a null string.

**(ICONW.SHADE WINDOW SHADE)** [Function]

---

Returns the current shading of the window *ICON*, which must be a window returned by **ICONW** or **TITLEDICONW**. In addition, if *SHADE* is non-**NIL**, paints the texture *SHADE* on *WINDOW*. A typical use for this function is to communicate a change of state in a window that is shrunk, without reopening the window. To remove any shading, make *SHADE* be **WHITESHADE**.

### 28.4.16.3 Default Icons

When you shrink a window that has no **ICONFN**, the system currently creates an icon that looks like the window's title bar. You can make the system instead create titled icons by setting the global variable **DEFAULTICONFN** to the value **TEXTICON**.

**(TEXTICON WINDOW TEXT)** [Function]

---

Creates a titled icon window for the main window *WINDOW* containing the text *TEXT*, or the window's title if *TEXT* is **NIL**.

**DEFAULTTEXTICON** [Variable]

---

The value that **TEXTICON** passes to **TITLEDICONW** as its *ICON* argument. Initially it is **NIL**, which creates an unadorned rectangular window. However, you can set it to a **TITLEDICON** record of your choosing if you would like default icons to have a different appearance.

### 28.4.16.4 Sample Icons

The LispUsers StockIcons module contains a collection of icons and their masks usable with **ICONW**, including:

- **FOLDER, FOLDERMASK** - a file folder
- **PAPERICON, PAPERICONMASK** - a sheet of paper with the top right corner turned
- **FILEDRAWER, FILEDRAWERMASK** - front of a file drawer
- **ENVELOPEICON, ENVELOPEMASK** - envelope
- **TITLED.FILEDRAWER** - TitledIcon of the filedrawer front (capacity, about three lines of 10-point text)
- **TITLED.FILEFOLDER** - TitledIcon of the file folder (capacity, about three lines of 10-point text)
- **TITLED.ENVELOPE** - TitledIcon of the envelope (capacity, one short line of 10-point text)

[This page intentionally left blank]

Free Menus are powerful and flexible menus that are useful for applications needing menus with different types of items, including command items, state items, and items that can be edited. A Free Menu is part of a window. It can be opened and closed as desired, or attached as a control menu to the application window.

### Making a Free Menu

A Free Menu is built from a description of the contents and layout of the menu. As a Free Menu is simply a group of items, a Free Menu Description is simply a specification of a group of items. Each group has properties associated with it, as does each Free Menu Item. These properties specify the format of the items in the group, and the behavior of each item. The function `FREEMENU` takes a Free Menu Description, and returns a closed window with the Free Menu in it.

The easiest way to make a Free Menu is to define a specific function which calls `FREEMENU` with the Free Menu Description in the function. This function can then also set up the Free Menu window as required by the application. The Free Menu Description is saved as part of the specific function when the application is saved. Alternately, the Free Menu Description can be saved as a variable in your file; then just call `FREEMENU` with the name of the variable. This may be a more difficult alternative if the backquote facility is used to build the Free Menu Description.

### Free Menu Formatting

A Free Menu can be formatted in one of four ways. The items in any group can be automatically laid out in rows, in columns, or in a table, or else the application can specify the exact location of each item in the group. Free Menu keeps track of the region that a group of items occupies, and items can be justified within that region. This way an item can be automatically positioned at one of the nine justification locations, top-left, top-center, top-right, middle-left, etc.

## Free Menu Description

A Free Menu Description, specifying a group of items, is a list structure. The first entry in the list is an optional list of the properties for this group of items. This entry is in the form:

```
(PROPS <PROP> <VALUE> <PROP> <VALUE> ...)
```

The keyword `PROPS` determines whether or not the optional group properties list is specified..

One important group property is `FORMAT`. The four types of formatting, `ROW`, `TABLE`, `COLUMN`, or `EXPLICIT`, determine the syntax of the rest of the Free Menu Description. When using `EXPLICIT` formatting, the rest of the description is any number of Item Descriptions which have `LEFT` and `BOTTOM` properties specifying the position of the item in the menu. The syntax is:

```
((PROPS FORMAT EXPLICIT ...)

 <ITEM DESCRIPTION>

 <ITEM DESCRIPTION> ...)
```

When using `ROW` or `TABLE` formatting, the rest of the description is any number of item groups, each group corresponding to a row in the menu. These groups are identical in syntax to an `EXPLICIT` group description. The groups have an optional `PROPS` list and any number of Item Descriptions. The items need not have `LEFT` and `BOTTOM` properties, as the location of each item is determined by the formatter. However, the order of the rows and items is important. The menu is laid out top to bottom by row, and left to right within each row. The syntax is:

```

((PROPS FORMAT ROW ...) ; props of this group

(<ITEM DESCRIPTION> ; items in first row

<ITEM DESCRIPTION> ...)

((PROPS ...) ; props of second row

<ITEM DESCRIPTION> ; items in second row

<ITEM DESCRIPTION> ...))

```

(The comments above only describe the syntax.)

For COLUMN formatting, the syntax is identical to that of ROW formatting. However, each group of items corresponds to a column in the menu, rather than a row. The menu is laid out left to right by column, top to bottom within each column.

Finally, a Free Menu Description can have recursively nested groups. Anywhere the description can take an Item Description, it can take a group, marked by the keyword GROUP. A nested group inherits all of the properties of its mother group, by default. However, any of these properties can be overridden in the nested groups PROPS list, including the FORMAT. The syntax is:

```

(; no PROPS list, default row format

(<ITEM DESCRIPTION> ; first in row

(GROUP ; nested group, second in row

 (PROPS FORMAT COLUMN ...) ; optional props

 (<ITEM DESCRIPTION> ...) ; first column

 (<ITEM DESCRIPTION> ...))

 <ITEM DESCRIPTION>)) ; third in row

```

Here is an example of a simple Free Menu Description for a menu which might provide access to a simple data base:

```

(((LABEL LOOKUP SELECTEDFN MYLOOKUPFN)

 (LABEL EXIT SELECTEDFN MYEXITFN))

 ((LABEL Name: TYPE DISPLAY) (LABEL "" TYPE EDIT ID NAME))

 ((LABEL Address: TYPE DISPLAY) (LABEL "" TYPE EDIT ID ADDRESS))

 ((LABEL Phone: TYPE DISPLAY)

```

```
((LABEL "" TYPE EDIT LIMITCHARS MYPHONEP ID PHONE)))
```

This menu has two command buttons, LOOKUP and EXIT, and three edit fields, with IDs NAME, PHONE, and ADDRESS. The Edit items are initialized to the empty string, as in this example they need no other initial value. The user could select the Name: prompt, type a person's name, and then press the LOOKUP button. The function MYLOOKUPFN would be called. That function would look at the NAME Edit item, look up that name in the data base, and fill in the rest of the fields appropriately. The PHONE item has MYPHONEP as a LIMITCHARS function. This function would be called when editing the phone number, in order to restrict input to a valid phone number. After looking up Perry, the Free Menu might look like:

|                            |      |
|----------------------------|------|
| LOOKUP                     | EXIT |
| Name: Herbert Q Perry      |      |
| Address: 13 Middleperry Dr |      |
| Phone: (411) 767-1234      |      |

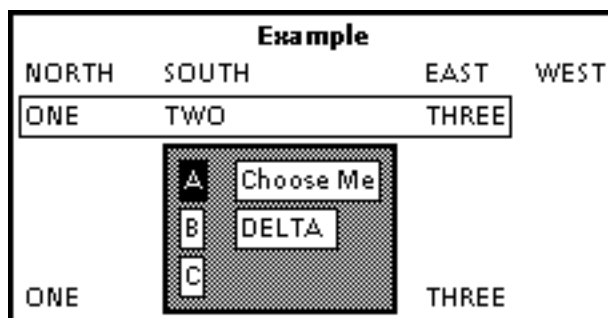
Here is a more complicated example:

```
((PROPS FONT (MODERN 10))
((LABEL Example FONT (MODERN 10 BOLD) HJUSTIFY CENTER))
((LABEL NORTH) (LABEL SOUTH) (LABEL EAST) (LABEL WEST))
((PROPS ID ROW3 BOX 1)
(LABEL ONE) (LABEL TWO) (LABEL THREE))
((PROPS ID ROW4)
(LABEL ONE ID ALPHA)
(GROUP (PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4)
((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT
T))
(TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
(TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
(TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)))
(LABEL THREE)))
```

which will produce the following Free Menu:

| Example |                                                                                                                                           |       |      |           |   |       |   |  |  |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------|-------|------|-----------|---|-------|---|--|--|
| NORTH   | SOUTH                                                                                                                                     | EAST  | WEST |           |   |       |   |  |  |
| ONE     | TWO                                                                                                                                       | THREE |      |           |   |       |   |  |  |
|         | <table border="1"> <tr> <td>A</td> <td>Choose Me</td> </tr> <tr> <td>B</td> <td>DELTA</td> </tr> <tr> <td>C</td> <td></td> </tr> </table> |       | A    | Choose Me | B | DELTA | C |  |  |
| A       | Choose Me                                                                                                                                 |       |      |           |   |       |   |  |  |
| B       | DELTA                                                                                                                                     |       |      |           |   |       |   |  |  |
| C       |                                                                                                                                           |       |      |           |   |       |   |  |  |
| ONE     |                                                                                                                                           | THREE |      |           |   |       |   |  |  |

And if the Free Menu were formatted as a Table, instead of in Rows, it would look like:



The following breakdown of the example explains how each part contributes to the Free Menu shown above.

```
((PROPS FONT (MODERN 10))
```

This line specifies the properties of the group that is the entire Free Menu. These properties are described in Section 28.7.4, Free Menu Group Properties. In this example, all items in the Free Menu, unless otherwise specified, will be in Modern 10.

```
((LABEL Example FONT (MODERN 10 BOLD) HJUSTIFY CENTER))
```

This line of the Free Menu Description describes the first row of the menu. Since the FORMAT specification of a Free Menu is, by default, ROW formatting, this line sets the first row in the menu. If the menu were in COLUMN formatting, this position in the description would specify the first column in the menu.

In this example the first row contains only one item. The item is, by default, a type MOMENTARY item. It has its own Font declaration (FONT (MODERN 10 BOLD)), that overrides the font specified for the Free Menu as a whole, so the item appears bolded.

Finally, the item is justified, in this case centered. The HJUSTIFY Item Property indicates that the item is to be centered horizontally within its row.

```
((LABEL NORTH) (LABEL SOUTH) (LABEL EAST) (LABEL WEST))
```

This line specifies the second row of the menu. The second row has four very simple items, labeled NORTH, SOUTH, EAST, and WEST next to each other within the same row.

```
((PROPS ID ROW3 BOX 1)
(LABEL ONE) (LABEL TWO) (LABEL THREE))
```

The third row in the menu is similar to the second row, except that it has a box drawn around it. The box is specified in the PROPS declaration for this row. Rows (and columns) are just like Groups in that the first thing in the declaration can be a list of properties for that row. In this case the row is named by giving it an ID property of ROW3. It is useful to name your groups if you want to be able to access and modify their properties later (via the function FM.GROUPPROP). It is boxed by specifying the BOX property with a value of 1, meaning draw the box one dot wide.

```
((PROPS ID ROW4)
(LABEL ONE ID ALPHA)
(GROUP (PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4)
((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT T))
(TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
(TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
(TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)))
(LABEL THREE)))
```

This part of the description specifies the fourth row in the menu. This row consists of: an item labelled ONE, a group of items, and an item labelled THREE. That is, Free Menu thinks of the group as an entry, and formats the rest of the row just as it were a large item.

```
(GROUP (PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4)
 ((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT T))
 (TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
 (TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
 ((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
 INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
 (TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)))
```

The second part of this row is a nested group of items. It is declared as a group by placing the keyword GROUP as the first word in the declaration. A group can be declared anywhere a Free Menu Description can take a Free Menu Item Description (as opposed to a row or column declaration).

The first thing in what would have been the second item declaration in this row is the keyword GROUP. Following this keyword comes a normal group description, starting with an optional list of properties, and followed by any number of things to go in the group (based on the format of the group).

This group's Props declaration is:

```
(PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4) .
```

It specifies that the group is to be formatted as a number of columns (instead of rows, the default). The entire group will have a background shade of 23130, and a box of width 2 around it, as you can see in the sample menu. The BOXSPACE declaration tells Free Menu to leave an extra four dots of room between the edge of the group (ie the box around the group) and the items in the group.

The first column of this group is a Collection of NWAY items:

```
((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT T))
 (TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
 (TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
```

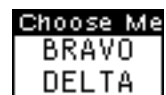
The three items, labelled A, B, and C are all declared as NWAY items, and are also specified to belong to the same NWAY Collection, Col1. This is how a number of NWAY items are collected together. The property NWAYPROPS (DESELECT T) on the first NWAY item specifies that the Col1 Collection is to have the Deselect property enabled. This simply means that the NWAY collection can be put in the state where none of the items (A, B, or C) are selected (highlighted). Additionally, each item is declared with a box whose width is one dot (pixel) around it.

The second column in this nested group is specified by:

```
((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
 INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
 (TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35))
```

Column two contains two items, a STATE item and a DISPLAY item. The STATE item is labelled "Choose Me." A Label can be a string or a bitmap, as well as an atom. Selecting the STATE item will cause a pop-up menu to appear with two choices for the state of the item, BRAVO and DELTA. The items to go in the pop-up menu are designated by the MENUITEMS property.

The pop-up menu would look like:



The initial state of the "Choose Me" item is designated to be DELTA by the INITSTATE Item Property. The initial state can be anything; it does not have to be one of the items in the pop-up menu.

Next, the STATE item is Linked to a DISPLAY item, so that the current state of the item will be displayed in the Free Menu. The link's name is DISPLAY (a special link name for STATE items), and the item linked to is described



by the Link Description, (GROUP ALPHA). Normally the linked item can just be described by its ID. But in this case, there is more than one item whose ID is ALPHA (for the sake of this example), specifically the first item in the fourth row and the display item in this nested group. The form (GROUP ALPHA) tells Free Menu to search for an item whose ID is ALPHA, limiting the search to the items that are within this lexical group. The lexical group is the smallest group that is declared with the GROUP keyword (i.e., not row and column groups) that contains this item declaration. So in this case, Free Menu will link the STATE item to the DISPLAY item, rather than the first item in the fourth row, since *that* item is outside of the nested group. For further discussion of linking items, see Section 28.7.12, Free Menu Item Links.

Now, establish the DISPLAY item:

```
(TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)
```

We have given it the ID of Alpha that the above STATE item uses in finding the proper DISPLAY item to link to. This display item is used to display the current state of the item "Choose Me." Every item is required to have a Label property specified, but the label for this DISPLAY item will depend on the state of "Choose Me." That is, when the state of the "Choose Me" item is changed from DELTA to BRAVO, the label of the DISPLAY item will also change. The null string serves to hold the place for the changeable label.

A box is specified for this item. Since the label is the empty string, Free Menu would draw a very small box. Instead, the MAXWIDTH property indicates that the label, whatever it becomes, will be limited to a stringwidth of 35. The width restriction of 35 was chosen because it is big enough for each of the possible labels for this display item. So Free Menu draws the box big enough to enclose any item within this width restriction.

Finally we specify the final item in row four:

```
(LABEL THREE)
```

## Free Menu Group Properties

Each group has properties. Most group properties are relevant and should be set in the group's PROPS list in the Free Menu Description. User properties can be freely included in the PROPS list. A few other properties are set up by the formatter. The macros FM.GROUPPROP or FM.MENUPROP allow access to group properties after the Free Menu is created.

|             |                                                                                                                                                                                                                                                                                                                                            |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ID          | The identifier of this group. Setting the group ID is desirable, for example, if the application needs to get handles on items in particular groups, or access group properties.                                                                                                                                                           |
| FORMAT      | One of ROW, COLUMN, TABLE, or EXPLICIT. The default is ROW.                                                                                                                                                                                                                                                                                |
| FONT        | A font description of the form (FAMILY SIZE FACE), or a FONTDESCRIPTOR data type. This will be the default font for each item in this group. The default font of the top group is the value of the variable DEFAULTFONT.                                                                                                                   |
| COORDINATES | One of GROUP or MENU. This property applies only to EXPLICIT formatting. If GROUP, the items in the EXPLICIT group are positioned in coordinates relative to the lower left corner of the group, as determined by the mother group. If MENU, which is the default, the items are positioned relative to the lower left corner of the menu. |
| LEFT        | Specifies a left offset for this group, pushing the group to the right.                                                                                                                                                                                                                                                                    |
| BOTTOM      | Specifies a bottom offset for this group, pushing the group up.                                                                                                                                                                                                                                                                            |
| ROWSPACE    | Specifies the number of dots between rows in this group.                                                                                                                                                                                                                                                                                   |
| COLUMNSPACE | Specifies the number of dots between columns in this group.                                                                                                                                                                                                                                                                                |
| BOX         | Specifies the number of dots in the box around this group of items.                                                                                                                                                                                                                                                                        |
| BOXSHADE    | Specifies the shade of the box.                                                                                                                                                                                                                                                                                                            |
| BOXSPACE    | Specifies the number of bits between the box and the items.                                                                                                                                                                                                                                                                                |

**BACKGROUND** The background shade of this group. Nested groups inherit this background shade, but items in this group and nested groups do not. This is because, in general, it is difficult to read text on a background, so items appear on a white background by default. This can be overridden by the **BACKGROUND** Item Property.

## Other Group Properties

The following group properties are set up and maintained by Free Menu. The application should probably not change any of these properties.

**ITEMS** A list of the items in the group.

**REGION** The region that is the extent of the items in the group.

**MOTHER** The ID of the group that is the mother of this group.

**DAUGHTERS** A list of ID of groups which are daughters to this group.

## Free Menu Items

Each Free Menu Item is stored as an instance of the data type **FREEMENUITEM**. Free Menu Items can be thought of as objects, each item having its own particular properties, such as its type, label, and mouse event functions. A number of useful item types, described in Section 28.7.11, **Predefined Item Types**, are predefined by Free Menu. New types of items can be defined by the application, using Display items as a base. Each Free Menu Item is created from a Free Menu Item Description when the Free Menu is created.

**CAUTION:** Edit (and thus Number) Freemenu Items do not perform well when boxed or when there is another item to the right in the same row. The display to the right of the edit item may be corrupted under editing and `fm.changelabel` operations.

## Free Menu Item Descriptions

A Free Menu Item Description is a list in property list format, specifying the properties of the item. For example:

```
(LABEL Refetch SELECTEDFN MY.REFETCHFN)
```

describes a **MOMENTARY** item labelled Refetch, with the function `MY.REFETCHFN` to be called when the item is selected. None of the property values in an item description are evaluated. When constructing Free Menu descriptions that incorporate evaluated expressions (for example labels that are bitmaps) it is helpful to use the backquote facility. For instance, if the value of the variable `MYBITMAP` is a bitmap, then

```
(FREEMENU `(((LABEL A) (LABEL ,MYBITMAP))))
```

would create a Free Menu of one row, with two items in that row, the second of which has the value of `MYBITMAP` as its label.

## Free Menu Item Properties

The following Free Menu Item Properties can be set in the Item Description. Any other properties given in an Item Description will be treated as user properties, and will be saved on the `USERDATA` property of the item.

|                 |                                                                                                                                                                                                                                                                                                                                                           |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TYPE            | The type of the item. Choose from one of the Free Menu Item type keywords MOMENTARY, TOGGLE, 3STATE, STATE, NWAY, EDITSTART, EDIT, NUMBER, or DISPLAY. The default is MOMENTARY.                                                                                                                                                                          |
| LABEL           | An atom, string, or bitmap. Bitmaps are always copied, so that the original will not be changed. This property must be specified for every item.                                                                                                                                                                                                          |
| FONT            | The font in which the item appears. The default is the font specified for the group containing this item. Can be a font description of the form (FAMILY SIZE FACE), or a FONTDESCRIPTOR data type.                                                                                                                                                        |
| ID              | May be used to specify a unique identifier for this item, but is not necessary.                                                                                                                                                                                                                                                                           |
| LEFT and BOTTOM | When ROW, COLUMN, or TABLE formatting, these specify offsets, pushing the item right and up, respectively, from where the formatter would have put the item. In EXPLICIT formatting, these are the actual coordinates of the item, in the coordinate system given by the group's COORDINATES property.                                                    |
| HJUSTIFY        | Indicates horizontal justification type: LEFT, CENTER, or RIGHT. Specifies that this item is to be horizontally justified within the extent of its group. Note that the main group, as opposed to the smaller row or column group, is used.                                                                                                               |
| VJUSTIFY        | Specifies that this item is to be vertically justified. Values are TOP, MIDDLE, or BOTTOM.                                                                                                                                                                                                                                                                |
| HIGHLIGHT       | Specifies the highlighted looks of the item, that is, how the item changes when a mouse event occurs on it. See Section 28.7.12, Free Menu Item Highlighting, for more details on highlighting.                                                                                                                                                           |
| MESSAGE         | Specifies a string that will be printed in the prompt window after a mouse cursor selects this item for MENUHELDWAIT milliseconds. Or, if an atom, treated as a function to get the message. The function is passed three arguments, ITEM, WINDOW, and BUTTONS, and should return a string. The default is a message appropriate to the type of the item. |
| INITSTATE       | Specifies the initial state of the item. This is only appropriate to TOGGLE, 3STATE, and STATE items.                                                                                                                                                                                                                                                     |
| MAXWIDTH        | Specifies the width allowed for this item. The formatter will leave enough space after the item for the item to grow to this width without collisions.                                                                                                                                                                                                    |
| MAXHEIGHT       | Similar to MAXWIDTH, but in the vertical dimension.                                                                                                                                                                                                                                                                                                       |
| BOX             | Specifies the number of bits in the box around this item. Boxes are made around MAXWIDTH and MAXHEIGHT dimensions. If unspecified, no box is drawn.                                                                                                                                                                                                       |
| BOXSHADE        | Specifies the shade that the box is drawn in. The default is BLACKSHADE.                                                                                                                                                                                                                                                                                  |
| BOXSPACE        | Specifies the number of bits between the box and the label. The default is one bit.                                                                                                                                                                                                                                                                       |
| BACKGROUND      | Specifies the background shade on which the item appears. The default is WHITESHADE, regardless of the group's background.                                                                                                                                                                                                                                |
| LINKS           | Can be used to link this item to other items in the Free Menu.                                                                                                                                                                                                                                                                                            |

## Mouse Properties

The following properties provide a way for application functions to be called under certain mouse events. These functions are called with the ITEM, the WINDOW, and the BUTTONS passed as arguments. These application functions do not interfere with any Free Menu system functions that take care of handling the different item types. In each case, though, the application function is called

after the system function. The default for all of these functions is NIL. The value of each of the following properties can be the name of a function, or a lambda expression.

|            |                                                                                                                                                                                                                                                                                                    |
|------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SELECTEDFN | Specifies the function to be called when this item is selected. The <code>Edit</code> and <code>EditStart</code> items cannot have a <code>SELECTEDFN</code> . See the <code>Edit Free Menu</code> item description in Section 28.7.11, <code>Predefined Item Types</code> , for more information. |
| DOWNFN     | Specifies the function to be called when the item is selected with the mouse cursor.                                                                                                                                                                                                               |
| HELDFN     | Specifies the function to be called repeatedly when the item is selected with the mouse cursor.                                                                                                                                                                                                    |
| MOVEDFN    | Specifies the function to be called when the mouse cursor moves off this item (mouse buttons are still depressed).                                                                                                                                                                                 |

## System Properties

The following Free Menu Item properties are set and maintained by Free Menu. The application should probably not change these properties directly.

|               |                                                                                                                                                                                                                                                        |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GROUPID       | Specifies the ID of the smallest group that the item is in. For example, in a row formatted group, the item's <code>GROUPID</code> will be set to the ID of the row that the item is in, not the ID of the whole group.                                |
| STATE         | Specifies the current state of <code>TOGGLE</code> , <code>3STATE</code> , or <code>STATE</code> items. The state of an <code>NWAY</code> item behaves like that of a toggle item.                                                                     |
| BITMAP        | Specifies the bitmap from which the item is displayed.                                                                                                                                                                                                 |
| REGION        | Specifies the region of the item, in window coordinates. This is used for locating the display position, as well as determining the mouse sensitive region of the item.                                                                                |
| MAXREGION     | Specifies the maximum region the item may occupy, determined by the <code>MAXWIDTH</code> and <code>MAXHEIGHT</code> properties (see Section 28.7.8, <code>Free Menu item Properties</code> ). This is used by the formatter and the display routines. |
| SYSDOWNFN     |                                                                                                                                                                                                                                                        |
| SYSMOVEDFN    |                                                                                                                                                                                                                                                        |
| SYSSELECTEDFN | These are the system mouse event functions, set up by Free Menu according to the item type. These functions are called before the mouse event functions, and are used to implement highlighting, state changes, editing, etc.                          |
| USERDATA      | Specifies how any other properties are stored on this list in property list format. This list should probably not need to be manipulated directly.                                                                                                     |

## Predefined Item Types

**MOMENTARY** [Free Menu Item]

`MOMENTARY` items are like command buttons. When the button is selected, its associated function is called.

**TOGGLE** [Free Menu Item]

Toggle items are simple two-state buttons. When pressed, the button is highlighted; it stays that way until pressed again. The states of a toggle button are `T` and `NIL`; the initial state is `NIL`.

**3STATE** [Free Menu Item]

3STATE items rotate through NIL, T, and OFF, states each time they are pressed. The default looks of the OFF state are with a diagonal line through the button, while T is highlighted, and NIL is normal. The default initial state is NIL.

The following Item Property applies to 3STATE items:

**OFF** Specifies the looks of a 3STATE item in its OFF state. Similar to HIGHLIGHT. The default is that the label gets a diagonal slash through it.

**NOTE:** If you specify special highlighting ( a different bitmap of string) for Toggle or 3State items AND use this item in a group formatted as a Column or a Table, the highlight looks of the item may not appear in the correct place.

## STATE

[Free Menu Item]

STATE items are general multiple state items. The following Item Property determines how the item changes state:

**CHANGESTATE** This Item Property can be changed at any time to change the effect of the item. If a MENU data type, this menu pops up when the item is selected, and the user can select the new state. Otherwise, if this property is given, it is treated as a function name, which is passed three arguments, ITEM, WINDOW, and BUTTONS. This function can do whatever it wants, and is expected to return the new state (an atom, string, or bitmap), or NIL, indicating the state should not change. The state of the item can automatically be indicated in the Free Menu, by setting up a DISPLAY link to a DISPLAY item in the menu (see Section 28.7.13, Free Menu Item Links). If such a link exists, the label of the DISPLAY item will be changed to the new state. The possible states are not restricted at all, with the exception of selections from a pop-up menu. The state can be changed to any atom, string, or bitmap, manually via FM.CHANGESTATE.

The following Item Properties are relevant to STATE items when building a Free Menu:

**MENUIITEMS** If specified, should be a list of items to go in a pop-up menu for this item. Free Menu will build the menu and save it as the CHANGESTATE property of the item.

**MENUFONT** The font of the items in the pop-up menu.

**MENUTITLE** The title of the pop-up menu. The default title is the label of the STATE item.

## NWAY

[Free Menu Item]

NWAY items provide a way to collect any number of items together, in any format within the Free Menu. Only one item from each Collection can be selected at a time, and that item is highlighted to indicate this. The following Item Properties are particular to NWAY items:

**COLLECTION** An identifier that specifies which NWAY Collection this item belongs to.

**NWAYPROPS** A property list of information to be associated with this collection. This property is only noticed in the Free Menu Description on the first item in a COLLECTION. NWAY Collections are formed by creating a number of NWAY items with the same COLLECTION property. Each NWAY item acts individually as a Toggle item, and can have its own mouse event functions. Each NWAY Collection itself has properties, its state for instance. After the Free Menu is created, these Collection properties can be accessed by the macro FM.NWAYPROPS. Note that NWAY Collections are different from Free Menu Groups. There are three NWAY Collection properties that Free Menu looks at:

**DESELECT** If given, specifies that the Collection can be deselected, yielding a state in which no item in the Collection is selected. When this property is set, the

Collection can be deselected by selecting any item in the Collection and pressing the right mouse button .

**STATE** The current state of the Collection, which is the actual item selected.

**INITSTATE** Specifies the initial state of the Collection. The value of this property is an Item Link Description

## **EDIT**

[Free Menu Item]

EDIT items are textual items that can be edited. The label for an EDIT item cannot be a bitmap. When the item is selected an edit caret appears at that cursor position within the item, allowing insertion and deletion of characters at that point. If selected with the right mouse button, the item is cleared before editing starts. While editing, the left mouse button moves the caret to a new position within the item. The right mouse button deletes from the caret to the cursor. CONTROL-W deletes the previous word. Editing is stopped when another item is selected, when the user moves the cursor into another TTY window and clicks the cursor, or when the Free Menu function FM.ENDEDIT is called (called when the Free Menu is reset, or the window is closed). The Free Menu editor will time out after about a minute, returning automatically. Because of the many ways in which editing can terminate, EDIT items are not allowed to have a SELECTEDFN, as it is not clear when this function should be called. Each EDIT item should have an ID specified, which is used when getting the state of the Free Menu, since the string being edited is defined as the state of the item, and thus cannot distinguish edit items. The following Item Properties are specific to EDIT items.

**MAXWIDTH** Specifies the maximum string width of the item, in bits, after which input will be ignored. If MAXWIDTH is not specified, the items becomes infinitely wide and input is never restricted.

**INFINITEWIDTH** This property is set automatically when MAXWIDTH is not specified. This tells Free Menu that the item has no right end, so that the item becomes mouse sensitive from its left edge to the right edge of the window, within the vertical space of the item.

In Medley, Changestate of an infinite width Edit item to a smaller item clears the old item properly.

**LIMITCHARS** The input characters allowed can be restricted in two ways: If this item property is a list, it is treated as a list of legal characters; any character not in the list will be ignored. If it is an atom, it is treated as the name of a test predicate, which is passed three arguments, ITEM, WINDOW, and CHARACTER, when each character is typed. This predicate should return T if the character is legal, NIL otherwise. The LIMITCHARS function can also call FM.ENDEDIT to force the editor to terminate, or FM.SKIPNEXT, to cause the editor to jump to the next edit item in the menu.

**ECHOCHAR** This item property can be set to any character. This character will be echoed in the window, regardless of what character is typed. However the item's label contains the actual string typed. This is useful for operations like password prompting. If ECHOCHAR is used, the font of the item must be fixed pitch. Unrestricted EDIT items should not have other items to their right in the menu, as they will be replaced. If the item is boxed, input is restricted to what will fit in the box. Typing off the edge of the window will cause the window to scroll appropriately. Control characters can be edited, including the carriage return and line feed, and they are echoed as a black box. While editing, the Skip/Next key ends editing the current item, and starts editing the next EDIT item in the Free Menu.

## **NUMBER**

[Free Menu Item]

NUMBER items are EDIT items that are restricted to numerals. The state of the item is coerced to the the number itself, not a string of numerals. There is one NUMBER- specific Item Property:

**NUMBERTYPE** If **FLOATP** (or **FLOAT**), then decimals are accepted. Otherwise only whole numbers can be edited.

#### **EDITSTART**

[Free Menu Item]

**EDITSTART** items serve the purpose of starting editing on another item when they are selected. The associated Edit item is linked to the EditStart item by an **EDIT** link (see Free Menu Item Links below). If the **EDITSTART** item is selected with the right mouse button, the Edit item is cleared before editing is started. Similar to **EDIT** items, **EDITSTART** items cannot have a **SELECTEDFN**, as it is not clear when the associated editing will terminate.

In Medley, **EDITSTART** items linked to a Number item properly set number state when editing has completed.

#### **DISPLAY**

[Free Menu Item]

**DISPLAY** items serve two purposes. First, they simply provide a way of putting dummy text in a Free Menu, which does nothing when selected. The item's label can be changed, though. Secondly, **DISPLAY** items can be used as the base for new item types. The application can create new item types by specifying **DOWNFN**, **HELDFN**, **MOVEDFN**, and **SELECTEDFN** for a **DISPLAY** item, making it behave as desired.

### **Free Menu Item Highlighting**

Each Free Menu Item can specify how it wants to be highlighted. First of all, if the item does not specify a **HIGHLIGHT** property, there are two default highlights. If the item is not boxed, the label is simply inverted, as in normal menus. If the item is boxed, it is highlighted in the shade of the box. Alternatively, the value of the **HIGHLIGHT** property can be a **SHADE**, which will be painted on top of the item when a mouse event occurs on it. Or the **HIGHLIGHT** property can be an alternate label, which can be an atom, string or bitmap. If the highlight label is a different size than the item label, the formatter will leave enough space for the larger of the two. In all of these cases, the looks of the highlighted item are determined when the Free Menu is built, and a bitmap of the item with these looks is created. This bitmap is stored on the item's **HIGHLIGHT** property, and simply displayed when a mouse event occurs. The value of the highlight property in the Item Description is copied to the **USERDATA** list, in case it is needed later for a label change.

### **Free Menu Item Links**

Links between items are useful for grouping items in abstract ways. In particular, links are used for associating **EDITSTART** items with their item to edit, and **STATE** items with their state display. The Free Menu Item property **LINKS** is a property list, where the value of each Link Name property is a pointer to another item. In the Item Description, the value of the **LINK** property should be a property list as above. The value of each Link Name property is a Link Description. A Link Description can be one of the following forms:

- <ID>** An ID of an item in the Free Menu. This is acceptable if items can be distinguished by ID alone.
- (<GROUPID> <ID>)** A list whose first element is a **GROUPID**, and whose second element is the ID of an item in that group. This way items with similar purposes, and thus similar ID's, can be distinguished across groups.
- (GROUP <ID>)** A list whose first element is the keyword **GROUP**, and whose second element is an item ID. This form describes an item with ID, in the same group that this item is in. This way you do not need to know the **GROUPID**, just which group it is in.

Then after the entire menu is built, the links are set up, turning the Link Descriptions into actual pointers to Free Menu Items. There is no reason why circular Item Links cannot be created, although such a link would probably not be very useful. If circular links are

created, the Free Menu will not be garbage collected after it is no longer being used. The application is responsible for breaking any such links that it creates.

## Free Menu Window Properties

|                 |                                                                                                                                                                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FM.PROMPTWINDOW | Specifies the window that Free Menu should use for displaying the item's messages. If not specified, PROMPTWINDOW is used.                                                                                                                                                              |
| FM.BACKGROUND   | The background shade of the entire Free Menu. This property can be set automatically by specifying a BACKGROUND argument to the function FREEMENU. The window border must be 4 or greater when a Free Menu background is used, due to the way the Window System handles window borders. |
| FM.DONTRESHAPE  | Normally, Free Menu will attempt to use empty space in a window by pushing items around to fill the space. When a Free Menu window is reshaped, the items are repositioned in the new shape. This can be disabled by setting the FM.DONTRESHAPE window property.                        |

## Free Menu Interface Functions

(**FREEMENU** *DESCRIPTION TITLE BACKGROUND BORDER*) [Function]

Creates a Free Menu from a Free Menu Description, returning the window. This function will return quickly unless new display fonts have to be created.

## Accessing Functions

(**FM.GETITEM** *ID GROUP WINDOW*) [Function]

Gets item *ID* in *GROUP* of the Free Menu in *WINDOW*. This function will search the Free Menu for an item whose *ID* property matches, or secondly whose *LABEL* property matches *ID*. If *GROUP* is *NIL*, then the entire Free Menu is searched. If no matching item is found, *NIL* is returned.

(**FM.GETSTATE** *WINDOW*) [Function]

Returns in property list format the *ID* and current *STATE* of every *NWAY* Collection and item in the Free Menu. If an item's or Collection's state is *NIL*, then it is not included in the list. This provides an easy way of getting the state of the menu all at once. If the state of only one item or Collection is needed, the application can directly access the *STATE* property of that object using the Accessing Macros described in Section 28.7.20, Free Menu Macros. This function can be called when editing is in progress, in which case it will provide the label of the item being edited at that point.

## Changing Free Menus

Many of the following functions operate on Free Menu Items, and thus take the item as an argument. The *ITEM* argument to these functions can be the Free Menu Item itself, or just a reference to the item. In the second case, **FM.GETITEM** (see Section 28.7.16, Accessing Functions) will be used to find the item in the Free Menu. The reference can be in one of the following forms:

<ID> Specifies the first item in the Free Menu whose *ID* or *LABEL* property matches <ID>.

(<GROUPID> <ID>) Specifies the item whose *ID* or *LABEL* property matches <ID> within the group specified by <GROUPID>.

(**FM.CHANGELABEL** *ITEM NEWLABEL WINDOW UPDATEFLG*) [Function]

Changes an *ITEM*'s label after the Free Menu has been created. It works for any type of item, and *STATE* items will remain in their current state. If the window is open, the item



will be redisplayed with its new appearance. *NEWLABEL* can be an atom, a string, or a bitmap (except for *EDIT* items), and will be restricted in size by the *MAXWIDTH* and *MAXHEIGHT* Item Properties. If these properties are unspecified, the *ITEM* will be able to grow to any size. *UPDATEFLG* specifies whether or not the regions of the groups in the menu are recalculated to take into account the change of size of this item. The application should not change the label of an *EDIT* item while it is being edited. The following Item Property is relevant to changing labels:

**CHANGELABELUPDATE** Exactly like *UPDATEFLG* except specified on the item, rather than as a function parameter.

**(FM.CHANGESTATE *X NEWSTATE WINDOW*)** [Function]

Programmatically changes the state of items and *NWAY* Collections. *X* is either an item or a Collection name. For items *NEWSTATE* is a state appropriate to the type of the item. For *NWAY* Collections, *NEWSTATE* should be the desired item in the Collection, or *NIL* to deselect. For *EDIT* and *NUMBER* items, this function just does a label change. If the window is open, the item will be redisplayed.

**(FM.RESETSTATE *ITEM WINDOW*)** [Function]

Sets an *ITEM* back to its initial state.

**(FM.RESETMENU *WINDOW*)** [Function]

Resets every item in the menu back to its initial state.

**(FM.RESETSHAPE *WINDOW ALWAYSFLG*)** [Function]

Reshapes the *WINDOW* to its full extent, leaving the lower-left corner unmoved. Unless *ALWAYSFLG* is *T*, the window will only be increased in size as a result of resetting the shape.

**(FM.RESETGROUPS *WINDOW*)** [Function]

Recalculates the extent of each group in the menu, updating group boxes and backgrounds appropriately.

**(FM.HIGHLIGHTITEM *ITEM WINDOW*)** [Function]

Programmatically forces an *ITEM* to be highlighted. This might be useful for *ITEMs* which have a direct effect on other *ITEMs* in the menu. The *ITEM* will be highlighted according to its *HIGHLIGHT* property, as described in Section 28.7.12, Free Menu Item Highlighting. This highlight is temporary, and will be lost if the *ITEM* is redisplayed, by scrolling for example.

## Editor Functions

**(FM.EDITITEM *ITEM WINDOW CLEARFLG*)** [Function]

Starts editing an *EDIT* or *NUMBER* *ITEM* at the beginning of the *ITEM*, as long as the *WINDOW* is open. This function will most likely be useful for starting editing of an *ITEM* that is currently the null string. If *CLEARFLG* is set, the *ITEM* is cleared first.

**(FM.SKIPNEXT *WINDOW CLEARFLG*)** [Function]

Causes the editor to jump to the beginning of the next *EDIT* item in the Free Menu. If *CLEARFLG* is set, then the next item will be cleared first. If there is not another *EDIT* item in the menu, this function will simply cause editing to stop. If this function is called when editing is not in progress, editing will begin on the first *EDIT* item in the menu. This function can be called from any process, and can also be called from inside the editor, in a *LIMITCHARS* function.

**(FM.ENDEDIT *WINDOW WAITFLG*)** [Function]

Stops any editing going on in *WINDOW*. If *WAITFLG* is T, then block until the editor has completely finished. This function can be called from another process, or from a *LIMITCHARS* function.

(**FM.EDITP** *WINDOW*) [Function]

If an item is in the process of being edited in the Free Menu *WINDOW*, that item is returned. Otherwise, *NIL* is returned.

## Miscellaneous Functions

(**FM.REDISPLAYMENU** *WINDOW*) [Function]

Redisplays the entire Free Menu in its *WINDOW*, if the *WINDOW* is open.

(**FM.REDISPLAYITEM** *ITEM WINDOW*) [Function]

Redisplays a particular Free Menu *ITEM* in its *WINDOW*, if the *WINDOW* is open.

(**FM.SHADE** *X SHADE WINDOW*) [Function]

*X* can be an item, or a group ID. *SHADE* is painted on top of the item or group. Note that this is a temporary operation, and will be undone by redisplaying. For more permanent shading, the application may be able to add a *REDISPLAYFN* and *SCROLLFN* for the window as necessary to update the shading.

(**FM.WHICHITEM** *WINDOW POSorX Y*) [Function]

Locates and identifies an item from its known location within the *WINDOW*. If *WINDOW* is *NIL*, (*WHICHW*) is used, and if *POSorX* is *NIL*, the current cursor location is used.

(**FM.TOPGROUPID** *WINDOW*) [Function]

Returns the ID of the top group of this Free Menu.

## Free Menu Macros

These Accessing Macros are provided to allow the application to get and set information in the Free Menu data structures. They are implemented as macros so that the operation will compile into the actual access form, rather than figuring that out at run time.

(**FM.ITEMPROP** *ITEM PROP {VALUE}*) [Macro]

Similar to *WINDOWPROP*, this macro provides an easy access to the fields of a Free Menu Item. The function *FM.GETITEM* gets the *ITEM*, described in Section 28.7.16, Accessing Function. *VALUE* is optional, and if not given, the current value of the *PROP* property will be returned. If *VALUE* is given, it will be used as the new value for that *PROP*, and the old value will be returned. When a call to *FM.ITEMPROP* is compiled, if the *PROP* is known (quoted in the calling form), the macro figures out what field to access, and the appropriate Data Type access form is compiled. However, if the *PROP* is not known at compile time, the function *FM.ITEMPROP*, which goes through the necessary property selection at run time, is compiled. The *TYPE* and *USERDATA* properties of a Free Menu Item are Read Only, and an error will result from trying to change the value of one of these properties.

(**FM.GROUPPROP** *WINDOW GROUP PROP {VALUE}*) [Macro]

Provides access to the Group Properties set up in the *PROPS* list for each group in the Free Menu Description. *GROUP* specifies the ID of the desired group, and *PROP* the name of

the desired property. If *VALUE* is specified, it will become the new value of the property, and the old value will be returned. Otherwise, the current value is returned.

(**FM.MENUPROP** *WINDOW PROP {VALUE}*) [Macro]

Provides access to the group properties of the top-most group in the Free Menu, that is to say, the entire menu. This provides an easy way for the application to attach properties to the menu as a whole, as well as access the Group Properties for the entire menu.

(**FM.NWAYPROP** *WINDOW COLLECTION PROP {VALUE}*) [Macro]

This macro works just like `FM.GROUPPROP`, except it provides access to the `NWay Collections`.

## APPENDIX E. ERROR SYSTEM

---

This appendix replaces Chapter 24, Error System, of *Common Lisp Implementation Notes*, Lyric Release, which replaced most of Chapter 24, Errors, of *Common Lisp, the Language*. Text shown with ~~StrikeThru~~ is that text from the Lyric release that no longer applies in Medley. Enhancements added in Medley are indicated with revision bars in the right margin.

The XCL error system has been updated to reflect the current ANSI Common Lisp error system proposal. This version seems to be gaining wide use in other Common Lisp implementations, so no further major changes are anticipated.

The Common Lisp error system is based on proposal number 18 for the Common Lisp error system. Deviations from this proposal are noted. Since the Common Lisp error system has not yet been standardized, this system may change in future releases to accommodate the final version of the Common Lisp error system.

If you have access to the ARPANet, a copy of this proposal may be retrieved from MIT-AI.ARPA as the file "COMMON;COND18 TXT".

All symbols described in the error system proposal that are not already in the "LISP" package are exported from the "CONDITIONS" package. In addition, the "XEROX-COMMON-LISP" package exports these symbols, so you can make them available either by using "XCL" or using "CONDITIONS", whichever is appropriate to your application. The distinction is made so that XCL extensions of the Common Lisp error system will be clear. All unqualified symbols are assumed to be in the "LISP" package.

---

### Summary of Error System Changes

---

The semantics of HANDLER-BIND where multiple bindings are set up or mutiple condition types are being handled are slightly different. Old code that used this will probably not behave as expected.

HANDLER-BIND and HANDLER-CASE (a.k.a. CONDITION-CASE) now always take a typespec instead of a list of condition types to indicate the conditions to be handled. Old code that uses this will only handle the first condition type in the list. The function, CONDITIONS::CONVERT-HANDLER-CASE is provided to aid in converting old code. It may be used as a mutation function in SEdit.

HANDLER-CASE now supports a :NO-ERROR option that is executed if none of the other clauses are taken. This is handy for writing code that depends on the normal completion of some operation, for example, creating auxilliary files if a particular stream is successfully opened.

SERIOUS-CONDITION no longer forces entry to the debugger. The function used to signal the condition now determines what happens if the condition is not handled. This means that SERIOUS-CONDITION has no more interesting properties and is likely to be removed in the final version of the error standard.

Several new condition types have been defined. Others have moved in the hierarchy. For example, ILLEGAL-GO is now a subtype of PROGRAM-ERROR.

No standard condition type has a default handler.

The standard debugger entry point is now called INVOKE-DEBUGGER instead of DEBUG.

The syntax of DEFINE-CONDITION has been changed to make it more like CLOS' DEFCLASS. The function CONDITIONS::CONVERT-OLD-DEFINE-CONDITION is provided to aid in converting old code. It may be used as a mutation function in SEdit.

Several DEFINE-CONDITION options have been merged, while others have been removed. In particular, there are no more "instant variables."

PROCEED-CASE has been replaced by RESTART-CASE. The semantics of restarts have been cleaned up and several new features added. Related functions, such as COMPUTE-PROCEED-CASES, have been renamed appropriately.

INVOKE-PROCEED-CASE has been renamed to INVOKE-RESTART.

DEFINE-PROCEED-FUNCTION has been removed, although XCL will continue to support it for compatibility.

The arguments to a restart's report function are different. Old code that used something other than a string for the report method will not work correctly.

A distinction is now made between invoking a restart interactively and simply invoking one. To this end, there is the function INVOKE-RESTART-INTERACTIVELY and the :INTERACTIVE option to RESTART-CASE.

RESTART-BIND, in analogy to HANDLER-BIND, has been added.

A new variable, \*BREAK-ON-SIGNALS\* exists to aid in debugging. It is a generalization of \*BREAK-ON-WARNINGS\*. The latter has been retained for compatibility.

The proceed function PROCEED has been changed to CONTINUE.

Old compiled code will continue to work except in the following cases, some of which have been mentioned above:

A proceed case's report function was not a simple string. Such code can cause stack overflow trying to report the condition (\*STANDARD-OUTPUT\* ends up being bound to NIL). Such code should be rewritten.

A handler binding is made to a list of condition types. Only the first type in the list will be handled.

Multiple handler bindings were created by the same HANDLER-BIND or HANDLER-CASE. Such code will work as expected, but if recompiled in Medley, will not. To get the effect of the current semantics, you must use nested HANDLER-BINDs.

Under the new error system, `use-value` and `store-value` no longer prompt for a value.

---

## Introduction to Error System Terminology

---

*condition* A *condition* is a kind of object which is created when an exceptional situation arises in order to represent the relevant features of that situation.

*signal, handlers* Once a condition is created, it is common to *signal* it. When a condition is signaled, a set of *handlers* are tried in some pre-defined order until one decides to *handle* the condition or until no more handlers are found. A condition is said to have been handled if a handler performs a non-local transfer of control to exit the signalling process.

*restart* Although such transfers of control may be done directly using traditional Lisp mechanisms such as `catch` and `throw`, `block` and `return`, or `tagbody` and `go`, the condition system also provides a more structured way to *restart* a computation. Among other things, the use of these structured primitives for restarting allows a better and more integrated relationship between the user program and the interactive debugger.

~~*serious conditions* It is not necessary that all conditions be handled. Some conditions are trivial enough that a failure to handle them may be disregarded. Others, which we will call *serious conditions* must be handled in order to assure correct program behavior. If a serious condition is signalled but no handler is found, the debugger will be entered so that the user may interactively specify how to proceed.~~

*errors* conditions which result from incorrect programs or data are called *errors*. Not all conditions are errors, however. Storage conditions are examples of conditions that are not errors. For example, the control stack may legitimately overflow without a program being in error. Even though a stack overflow is not necessarily a program error, it is serious enough to warrant entry to the debugger if the condition goes unhandled.

Some types of conditions are predefined by the system. All types of conditions are subtypes of `conditions:condition`. That is,

```
(typep c 'conditions:condition)
```

is true if `c` is a condition.

*creating conditions* The only standard way to define a new condition type is `conditions:define-condition`. The only standard way to instantiate a condition is `conditions:make-condition`.

When a condition object is created, the most common operation to be performed upon it is to *signal* it (although there may be applications in which this does not happen, or does not happen immediately).

When a condition is signaled, the system tries to locate the most appropriate handler for the condition and invoke that handler. Handlers are located according to the following rules:

- bound*
- Check for locally defined (ie, *bound*) handlers.
  - If no appropriate bound handler is found, check first for the default handler of the signaled type and then of each of its superiors.

*decline* If an appropriate handler is found, the handler may *decline* by simply returning without performing a non-local transfer of control. In such cases, the search for an appropriate handler is picked up where it left off, as if the called handler had never been present. When a handler is running, the "handler binding stack" is popped back to just below the binding that caused that handler to be invoked. This is done to avoid infinite recursion in the case that a handler also signals a condition.

`conditions:handler-bind` When a condition is signaled, handlers are searched for in the dynamic environment of the signaller. Handlers can be established within a dynamic context by use of `conditions:handler-bind` and other forms based on it.

*handler* A *handler* is a function of one argument, the condition to be handled. The handler may inspect the object (using primitives described in another section) to be sure it is interested in handling the condition. After inspecting the condition, the handler must take one of the following actions:

- It may decline to handle the condition by simply returning. When this happens, any returned values are ignored and the effect on the signaling process is the same as if the handler had not run. The next handler in line will be tried, or if no such handler exists, the default action for the given condition will be taken. A default handler may also decline, in which case the condition will go unhandled. What happens then depends on which function was used to signal the condition (`xcl:signal`, `error`, `cerror`, `warn`).
- It may perform some non-local transfer of control using `go`, `return`, `throw`, `abort`, or `conditions:invoke-restart`.
- It may signal another condition.
- It may invoke the debugger.

`conditions:restart-case` When a condition is signalled, a facility is available for use by handlers to transfer control to an outer dynamic contour of the program. The form which creates contours that may be returned to is `conditions:restart-case`. Each contour is set up by a `conditions:restart-case` clause, and is called a *restart*. The function that transfers control to a restart is `conditions:invoke-restart`.

~~*proceed function*~~ Also, control may be transferred along with parameters to a named ~~`xcl:proceed-case`~~ clause by invoking a ~~*proceed function*~~ of that name.

~~*Proceed functions*~~ are created with the macro ~~`xcl:define-proceed-function`~~.

*restart type* A restart with a particular name is sometimes called a *restart type*.

*report* In some cases, it may be useful to *report* a condition or a restart to a user or a log file of some sort. When the printer is invoked on a condition or proceed case and `*print-escape*` is nil, the report function for that object is invoked. In particular, this means that an expression like

```
(princ condition)
```

will invoke `condition's` report function. Because of this, no special function is provided for invoking the report function of a condition or a restart.

---

## Program Interface to the Condition System

---

### Defining and Creating Conditions

---

**`conditions:define-condition`** *name* (*parent-type*) [(*{slot}\**) *{option}\**]

[Macro]

Defines a new condition type with the given *name*, making it a subtype of the given *parent-type*.

Except as otherwise noted, the arguments are not evaluated.

The valid *options* are:

```
(:documentation doc-string)
```

*doc-string* should be a string which describes the purpose of the condition type or NIL. If this option is omitted, NIL is assumed. (`documentation` *name* 'type) will retrieve this information.

```
(:conc-name symbol-or-string)
```

As in `defstruct`, this sets up automatic prefixing of the names of slot accessors. Also as in `defstruct` if no prefix is specified the default behavior for automatic prefixing is to use the name of the new type followed by a hyphen interned in the



package which is current at the time that the `conditions:define-condition` is processed.

~~`:report-function expression`~~

~~*expression* should be a suitable argument to the function special form, e.g., a symbol or a lambda expression. It designates a function of two arguments, a condition and a stream, which prints the condition to the stream when `*print-escape*` is nil.~~

~~The `:report-function` describes the condition in a human-sensible form. This item is somewhat different than a structure's `:print-function` in that it is only used if `*print-escape*` is nil.~~

`(:report exp)`

This option specifies the report function for this condition type. Report function are inherited, so if a particular condition type does not have one, the report function of its parent will be used.

If *exp* is a string, it is a shorthand for

```
(:report (lambda (condition stream)
 (declare (ignore conditions))
 (princ exp stream)))
```

If *exp* is not a string, `(function exp)` will be evaluated in the current lexical environment. This should return a function of two arguments, a condition and a stream. It will be called when a condition of this type is to be printed and `*print-escape*` is nil. The report function will be called with the condition to be reported and the stream to which the report is to be made.

~~`:handler-function expression`~~

~~*expression* should be a suitable argument to the function special form. It designates a function of one argument, a condition, which may handle that condition if no dynamically-bound handler did.~~

`(:handle exp)`

This option specifies a default handler for conditions of this type. `(function exp)` will be evaluated in the current lexical context. This should result in a function of one argument, a condition, to be used as the default handler for this condition type.

Each *slot* is a `defstruct slot-description`. In addition to those specified, the slots of the *parent-type* are also available. No slot-options are allowed, only an optional default-value expression. Condition objects are immutable, i.e., all of their slots are automatically declared to be `:read-only`.

conditions:make-condition will accept keywords with the same name as any of the slots, and will initialize the corresponding slots in conditions it creates.

Accessors are created according to the same rules as used by defstruct. For example:

```
(conditions:define-condition bad-food-color (food-lossage)
 (food color)
 (:report (lambda (c s) (format s "The food ~A was ~A"
 (bad-food-color-food c) (bad-food-
color-color c)))))
```

defines a condition of type bad-food-color which inherits from the food-lossage condition type. The new type has slots food and color so that conditions:make-condition will accept :food and :color keywords and accessors bad-food-color-food and bad-food-color-color will apply to objects of this type.

The report function for a condition will be implicitly called any time a condition is printed with \*print-escape\* being nil. Hence,

```
(princ condition)
```

is a way to invoke the condition's report function.

Here are some examples of defining condition types. This form defines a condition called machine-error which inherits from error:

```
(conditions:define-condition machine-error (error) (machine-
name)
 (:report (lambda (c s) (format s
 "There is a problem with ~A."
 (machine-error-machine-name c)))))
```

The following defines a new error condition (a subtype of machine-error) for use when machines are not available:

```
(conditions:define-condition machine-not-available-error
 (machine-error) (machine-name)
 (:report (lambda (c s) (format s
 "The machine ~A is not available."
 (machine-error-machine-name c)))))
```

The following defines a still more specific condition, built upon machine-not-available-error, which provides a default for machine-name but which does not provide any new slots:

```
(conditions:define-condition
 my-favorite-machine-not-available-error
 (machine-not-available-error)
 ((machine-name "Tesuji:AISDev")))
```

This gives the machine-name slot a default initialization. Since no :report clause was given, the information supplied in the definition of machine-not-available-error will be used if a condition of this type is printed while \*print-escape\* is nil.

Returns the object used to report conditions of the given *type*. This will be either a string, a function of two arguments (condition and stream) or `nil` if there is no report function. `setf` may be used with this form to change the report function for a condition type.

---

**`xcl:condition-handler`** *type* [Macro]

Returns the default handler for conditions of the given *type*. This will be a function of one argument or `nil` if there is no default handler. `setf` may be used with this form to change the default handler for a condition type.

---

**`conditions:make-condition`** *type* &rest *slot-initializations* [Function]

Calls the appropriate constructor function for the given *type*, passing along the given slot initializations to the constructor, and returning an instantiated condition.

The *slot-initializations* are given in alternating keyword/value pairs. eg,

```
(conditions:make-condition 'bad-food-color
 :food my-food
 :color my-color)
```

This function is provided mainly for writing subroutines that manufacture a condition to be signaled. Since all of the condition signalling functions can take a *type* and *slot-initializations*, it is usually easier to call them directly.

---

## Signalling Conditions

---

**`xcl:*current-condition*`** [Variable]

This variable is bound by condition-signalling forms (`conditions:signal`, `error`, `cerror`, and `warn`) to the condition being signaled. This is especially useful in restart filters. The top-level value of `xcl:*current-condition*` is `nil`.

---

**`conditions:signal`** *datum* &rest *arguments* [Function]

Invokes the signal facility on a condition. If the condition is not handled, `conditions:signal` returns the condition object that was signaled.

If *datum* is a condition then that condition is used directly. In this case, it is an error for *arguments* to be non-`nil`.

If *datum* is a condition type, then the condition used is the result of doing

```
(apply #'conditions:make-condition
 datum arguments)
```

If *datum* is a string, then the condition used is the result of doing

```
(conditions:make-condition
 'conditions:simple-condition
 :format-string datum
 :format-arguments arguments).
```

~~If the condition is of type `xcl:serious-condition`, then `xcl:signal` will behave exactly like `error`, i.e., it will call~~

~~xcl:debug~~ if the condition isn't handled, and will never return to its caller.

If (typep *condition* conditions:\*break-on-signals\*) is true, then the debugger will be entered prior to the signalling process. This is true for all other functions and macros that signal conditions, such as warn, error, cerror, assert and check-type.

### **conditions:\*break-on-signals\***

[Variable]

This flag is primarily for use when debugging programs that do signaling. Its value is a type specifier.

When (typep *condition* conditions:\*break-on-signals\*) is true, then calls to conditions:signal and other functions that implicitly call conditions:signal will enter the debugger prior to signalling the condition. The conditions:continue restart may be used to continue with the normal signalling process.

The default value of this variable is nil.

Note: the variable \*break-on-warnings\* continues to be supported for compatibility, but conditions:\*break-on-signals\* offers that power and more. New code should not use \*break-on-warnings\*.

### **error datum &rest arguments**

[Function]

Like conditions:signal except if the condition is not handled, the debugger is called with the given condition, and error never returns.

*datum* is treated as in conditions:signal. If *datum* is a string, a condition of type conditions:simple-error is made. This form is compatible with that described in Steele's *Common Lisp, the Language*.

### **ceerror proceed-format-string datum &rest arguments**

[Function]

Like error, if the condition is not handled the debugger is called with the given condition. However, cerror enables the restart conditions:continue, which will simply return the condition being signalled from cerror.

*datum* is treated as in error. If *datum* is a condition, then that condition is used directly. In this case, *arguments* will be used only with the *proceed-format-string* and will not be used to initialize *datum*.

The *proceed-format-string* must be a string. Note that if *datum* is not a string, then the format arguments used by the *proceed-format-string* will still be the *arguments* (in the keyword format as specified). In this case, some care may be necessary to set up the *proceed-format-string* correctly. The format directive ~\* may be particularly useful in this situation.

The value returned by cerror is the condition which was signaled.

See Steele's *Common Lisp, the Language*, page 430 for examples of the use of `cerror`.

**warn** *datum* &rest *arguments*

[Function]

Invokes the signal facility on a condition. If the condition is not handled, then the text of the warning is printed on `*error-output*`. If the variable `*break-on-warnings*` is true, then in addition to printing the warning, the debugger is entered using the function `break`. The value returned by `warn` is the condition that was signalled.

If *datum* is a condition, then that condition is used directly. In this case, if the condition is not of type `conditions:warning` or *arguments* is non-null, then an error of type `conditions:type-error` is signalled.

If *datum* is a condition type, then the condition used is the result of doing `(apply #'conditions:make-conditions datum arguments)`. This result must be of type `conditions:warning` or an error of type `conditions:type-error` is signalled.

If *datum* is a string, then the condition used is the result of `(conditions:make-conditions 'conditions:simple-warning :format-string datum :format-arguments arguments)`.

The precise mechanism for warning is as follows:

- 1) If `*break-on-warnings*` is true, the debugger will be entered. This feature is primarily for compatibility with old code: use of `conditions:*break-on-signals*` is preferred. If the break is continued using the `conditions:continue` restart, `warn` proceeds with step 2.
- 2) The warning condition is signalled. While it is being signalled, the `conditions:muffle-warning` restart is established for use by a handler to bypass further action by `warn`, i.e., to cause `warn` to immediately return.
- 3) The warning condition is reported to `*error-output*` by the `warn` function. Note that `warn` will indicate that the condition being signalled is a warning when it reports it, so there is no need for the condition to do so in its report method.

**\*break-on-warnings\***

[Variable]

**check-type**

[Macro]

**ecase**

[Macro]

**ccase**

[Macro]

**etypecase**

[Macro]

**ctypecase**

[Macro]

**assert**

[Macro]

All of the above behave as described in *Common Lisp: the Language*. The default clauses of `ecase` and `ccase` forms signal `conditions:simple-error` conditions. The default clauses of

`etypecase` and `ctypecase` forms signal `conditions:type-error` conditions. `assert` signals the `xcl:assertion-failed` condition. `ccase` and `ctypecase` set up a `conditions:store-value` restart.

## Handling Conditions

**`conditions:handler-bind`** *bindings* &*rest forms* [Macro]

Executes the forms in a dynamic context where the given local handler *bindings* are in effect. The elements of *bindings* must take the form (*type-spec handler*). The handlers are bound in the order they are given, i.e., when searching for a handler, the error system will consider the leftmost binding in a particular `conditions:handler-bind` form first. However, while one of these handlers is running, none of the bindings established by the `conditions:handler-bind` will be in effect.

*type* must be a type specifier. To make a binding for several condition types, use (*or type1 type2 ...*).

*handler* should evaluate to a function of one argument, a condition, to be used to handle a signalled condition during execution of the *forms*.

An example of the use of `conditions:handler-bind` appears at the end of the `conditions:restart-case` macro description.

**`conditions:handler-case`** *form* &*rest cases* [Macro]

**`xcl:condition-case`** *form* &*rest cases* [Macro]

Executes the given *form*. Each case has the form

(*type* ([*var*]) . *body*)

If a condition is signalled (and not handled by an intervening handler) during the execution of the form, and there is an appropriate clause—i.e., one for which

(*typep condition 'type*)

is true—then control is transferred to the body of the relevant clause, binding *var*, if present, to the condition that was signaled. If no condition is signalled, then the values resulting from the *form* are returned by the `xcl:condition-case`. If the condition is not needed, *var* may be omitted.

Earlier clauses will be considered first by the error system. I.e.,

```
(xcl:condition-case form
 (cond1 ...)
 (cond2 ...))
```

is equivalent to

```
(xcl:condition-case
 (xcl:condition-case form
 (cond1 ...))
 (cond2 ...))
```

~~type may also be a list of types, in which case it will catch conditions of any of the specified types.~~

One may also specify an action to be taken if execution of *form* completes normally. This may be done by specifying a clause that has `:no-error` as its type. Such a clause, if provided, must be last. A `:no-error` clause looks like:

```
(:no-error lambda-list . body)
```

If execution of the form completes normally and there is a `:no-error` clause, the values produced by the form will be bound to variables in the clause's *lambda-list* and the *body* will be executed with none of the handler bindings in effect. In this case the value of the `xcl:condition-case` form is the value returned by the last form of the *body* of its `:no-error` clause. Having a `:no-error` clause is equivalent to wrapping `(multiple-value-call #'(lambda (lambda-list) . body) ...)` around the `xcl:condition-case` form.

`conditions:handler-case` is synonymous with `xcl:condition-case`.

Examples:

```
(xcl:condition-case (/ x y)
 (division-by-zero () nil))

(xcl:condition-case (open *the-file*
 :direction :input)
 (file-error (condition)
 (format t "~&Open failed: ~A~%" condition)))

(xcl:condition-case (some-user-function)
 (file-error (condition) condition)
 (division-by-zero () 0)
 ((or unbound-variable undefined-function) ()
 'unbound))

(xcl:condition-case (open my-file)
 (file-error ()
 (format *error-output* "Couldn't open ~S."
 my-file))
 (:no-error (stream)
 (open-more-files my-file stream) stream)))
```

Note the difference between `xcl:condition-case` and `conditions:handler-bind`. In `conditions:handler-bind`, you are specifying functions that will be called in the dynamic context of the condition signalling form. In `xcl:condition-case`, you are specifying continuations to be used instead of the original form if a condition of a particular type is signaled. These continuations will be executed in the same dynamic context as the original form.

**`conditions:ignore-errors` & *body forms***

[Macro]

Executes the forms in a context that handles conditions of type `error` by returning control to this form. If no error is signaled, all

values returned by the last form are returned by `conditions:ignore-errors`. Otherwise, the form returns the two values `nil` and the condition that was signaled. Synonym for

```
(xcl:condition-case (progn . forms)
 (error (condition))
 (values nil condition)).
```

~~`xcl:debug` &optional *datum* &rest *arguments*~~ [Function]

~~Enters the debugger with a given condition without signalling that condition. When the debugger is entered, it will announce the condition by invoking the condition's report function.~~

~~*datum* is treated the same as for `xcl:signal` except if *datum* is not specified, it defaults to "Call to DEBUG".~~

~~This function will never directly return to its caller. Return can occur only by a special transfer of control, such as to a `catch`, `block`, `tagbody`, `xcl:proceed-case` or `xcl:catch-abort`.~~

**`conditions:invoke-debugger` *condition*** [Function]

Invokes the debugger with the given condition. This is intended to be used as a portable entry point to the debugger. For finer control over the debugging state, see the function `xcl:debugger`.

**`break` &optional *format-string* &rest *format-arguments*** [Function]

Enters the debugger with a simple condition with the given arguments. If no *format-string* is provided, it defaults to "Break." Computation may be continued by invoking the `conditions:continue` restart. If continued, `break` returns `nil`.

`break` is approximately:

```
(defun break (&optional (format-string "Break")
 &rest format-arguments)
 (conditions:restart-case (conditions:invoke-debugger
 (conditions:make-conditions 'conditions:simple-condition
 :format-string format-string :format-arguments format-arguments)
 (conditions:continue ()
 :report "Return from BREAK."
 nil)))
```

## Restarts

**`conditions:restart-case` *expression* { (*case-name* *arglist* {*keyword* *value*}\* {*form*}\*) }\***

[Macro]

The *expression* is evaluated in a dynamic context where the case clauses have special meanings as points to which control may be transferred. If *expression* runs to completion, all values returned by the form are simply returned by the `conditions:restart-case` form. On the other hand, the computation of *expression* may choose to transfer control to one of the restart clauses. If a transfer to a clause occurs, the forms in the body of that clause will be



evaluated in the same dynamic context as the `conditions:restart-case` form, and any values returned by the last such form will be returned by the `conditions:restart-case` form.

A restart clause has the form given above:

`(case-name arglist {keyword value}* {form}*)`

The *case-name* may be `nil` or any symbol.

The *arglist* is a normal lambda list that will be bound and evaluated in the dynamic context of the `conditions:restart-case` form. They will use whatever values were provided by `conditions:invoke-restart` or `conditions:invoke-restart-interactively`. Definitions of these two functions appear later in this section.

The valid *keyword/value* pairs are:

`:filter` *expression*

*expression* should be suitable as an argument to the function special form. It defines a predicate of no arguments that determines if this clause is visible to `conditions:find-restart`. Default = `true`.

`:condition` *type*

Shorthand for a common special case of `:filter`. The following two *key/value* pairs are equivalent:

```
:condition foo

:filter
 (lambda ()
 (typep xcl:*current-condition*
 'foo))
```

`:interactive` *expression*

The *expression* must be a form suitable as an argument to function. (function *expression*) will be evaluated in the current lexical and dynamic environments. The result should be a function of no arguments which returns a list of values to be used by `conditions:invoke-restart-interactively`. This function will be called in the dynamic environment available prior to any restart attempt. Any interaction with the user should be done here and not in the body of the restart.

If there is no `:interactive` option specified and the restart is invoked interactively, no arguments will be supplied.

`:report` *expression*

The *expression* can either be a constant string or a form suitable as an argument to function.

If *expression* is not a string, (function *expression*) will be evaluated in the current lexical and dynamic environment. The

result should be a function of one argument, a stream, which will be called to report that restart. This function should print a short summary of the action that restart will take if invoked.

If *expression* is a string, it is a shorthand for `(lambda (s) (format s expression))`.

Only one of `:condition` or `:filter` may be specified. If no `:report` is specified, the *case-name* will be used. It is an error to have a null case name and no report function.

Examples:

```
(loop
 (conditions:restart-case
 (return (apply function some-args))
 (new-function (new-fn)
 :report "Use a different function."
 :interactive (lambda ()
 (list (prompt-for 'function "Function:
 "))))
 (setq function new-fn))))

(loop
 (conditions:restart-case
 (return (apply function some-args))
 (nil (new-fn)
 :report "Use a different function."
 :interactive (lambda ()
 (list (prompt-for 'function "Function:
 "))))
 (setq function new-fn))))

(conditions:restart-case (a-command-loop)
 (return-from-command-level ()
 :report
 (lambda (stream)
 (format stream "Return from command level ~D."
 level)))
 nil))

(loop
 (conditions:restart-case (another-computation)
 (conditions:continue () nil)))
```

The first and second examples are equivalent from the point of view of someone using the interactive debugger, but differ in one important aspect for non-interactive handling. If a handler "knows about" restart names, as in:

```
(when (conditions:find-restart 'new-function)
 (conditions:invoke-restart 'new-function the-
 replacement))
```

then only the first example, and not the second, will have control transferred to its correction clause.

Here's a more complete example:

```
(let ((my-food 'milk)
 (my-color 'greenish-blue))
 (do ()
```

```
((not (food-colorable-p my-food
 my-color)))
(conditions:restart-case (error 'bad-food-color
 :food my-food
 :color my-color)
 (use-food (new-food)
 :report "Use another food."
 (setf my-food new-food))
 (use-color (new-color)
 :report "Use another color."
 (setf my-color new-color))))
;; We won't get to here until my-food
;; and my-color are compatible.
(list my-food my-color))
```

Assuming that use-food and use-color have been defined as

```
(defun use-food (new-food)
 (invoke-restart 'use-food new-food))

(defun use-color (new-color)
 (invoke-restart 'use-color new-color))
```

then a handler can proceed from the error in either of two ways. It may correct the color or correct the food. For example:

```
#'(lambda (condition) ...
 ;; Corrects color
 (use-color 'white) ...)

or

#'(lambda (condition) ...
 ;; Corrects food
 (use-food 'cheese) ...)
```

Here is an example using conditions:handler-bind and conditions:restart-case.

```
(conditions:handler-bind ((foo-error
 #'(lambda (condition)
 (conditions:use-value 7))))
 (conditions:restart-case (error 'foo-error)
 (conditions:use-value (x) (* x x))))
```

The above form returns 49.

---

```
xcl:define-proceed-function name [Macro]
 {keyword value}*
 {variable}*
```

---

~~Valid keyword/value pairs are the same as those which are defined for the xcl:proceed-case special form. That is, .filter, .filter-function, .condition, .report, and .report-function. The filter and report functions specified in a xcl:define-proceed-function form will be used for xcl:proceed-case clauses with the same name that do not specify their own filter or report functions, respectively.~~

~~This form defines a function called name which will invoke a proceed case with the same name. The proceed function takes optional arguments which are given by the variables specification. The parameter list for the proceed function will look like~~

~~(*&optional* . *variables*)~~

The only thing that a *proceed* function really does is collect values to be passed on to a *proceed* case clause.

Each element of *variables* has the form *variable-name* or *(variable-name initial-value)*. If *initial-value* is not supplied, it defaults to *nil*.

For example, here are some possible *proceed* functions which might be useful in conjunction with the *bad-food-color* error we used as an example earlier:

```
(xcl:define-proceed-function use-food
 :report "Use another food."
 (food (read-typed-object 'food
 "Food to use instead: ")))

(xcl:define-proceed-function use-color
 :report "Change the food's color."
 (color
 (read-typed-object 'food
 "Color to make the food: ")))

(defun maybe-use-water (condition)
 ; A sample handler
 (when (eq (bad-food-color food condition)
 'milk)
 (use-food 'water)))

(xcl:handler-bind ((bad-food-color
 #'maybe-use-water))
 ...)
```

If a named *proceed* function is invoked in a context in which there is no active *proceed* case by that name, the *proceed* function simply returns *nil*. So, for example, in each of the following pairs of handlers, the first is equivalent to the second but less efficient:

```
;'(lambda (condition) ; OK, but slow
 (when (xcl:find-proceed-case 'use-food)
 (use-food 'milk)))
;'(lambda (condition) ; Preferred
 (use-food 'milk))

;'(lambda (condition)
 (cond ((xcl:find-proceed-case 'use-food)
 (use-food 'chocolate))
 ((xcl:find-proceed-case 'use-color)
 (use-color 'orange))))
;'(lambda (condition)
 (use-food 'chocolate)
 (use-color 'orange))
```

**conditions:restart-bind** (*{ (name function {keyword value}\*) }\* {form}\* [Macro]*)

Executes the *forms* in a dynamic context where the given restart bindings are in effect.

*name* may be *nil* to indicate an anonymous restart, or some other symbol to indicate a named restart.

*function* will be evaluated in the current lexical and dynamic contexts and should produce a function of no arguments to be used to perform the restart. This function will be called when that restart is activated by `conditions:invoke-restart` or `conditions:invoke-restart-interactively`. Note that unlike `conditions:restart-case`, invoking the restart does not automatically transfer control back to the contour in which it was established. If that is appropriate for that restart it is up to the individual restart function to do this.

The valid *keyword/value* pairs are:

`:interactive-function` *form*

*form* will be evaluated in the current lexical and dynamic environments and should produce a function of no arguments that will construct the list of values to be used by `conditions:invoke-restart-interactively`.

`:report-function` *form*

*form* will be evaluated in the current lexical and dynamic environments and should produce a function of one argument, a stream, that will be used to report that restart.

`:filter-function` *form*

*form* will be evaluated in the current lexical and dynamic environments and should produce a function of no arguments that will be used to determine if the given restart is currently active.

This form is a more primitive way of establishing restarts than `conditions:restart-case`. It is expected that `conditions:restart-case` will be sufficient for most uses of the restart facility. An example of where the more general facility provided by `conditions:restart-bind` may be useful is:

```
(conditions:restart-bind ((nil #'(lambda ()
 (expunge-directory the-dir)) :report-function
 #'(lambda (stream) (format stream "Expunge ~A."
 (directory-namestring the-dir)))) (cerror "Try
 this file operation again." 'directory-full
 :directory the-dir))
```

In this case, a restart is provided that allows the user to expunge the full directory and return to the debugger after doing so. He can then try some other restart, such as `conditions:continue` to retry the failed operation.

### **conditions:compute-restarts**

[Function]

Uses the dynamic state of the program to compute a list of *restarts*.

Each restart object represents a point in the current dynamic state of the program to which control may be transferred. The only operations that Lisp defines for such objects are:

```
conditions:restart-name,
conditions:find-restart,
conditions:invoke-restart, conditions:invoke-
restart-interactively,
princ, and
prinl,
```

to identify an object as a restart using (`typep x 'conditions:restart`), and standard Lisp operations that work for all objects, such as `eq`, `eql`, `describe`, etc.

The list which results from a call to `conditions:compute-restarts` is ordered so that the innermost (ie, more-recently established) restarts are nearer the head of the list.

Note also that `conditions:compute-restarts` returns *all* valid restarts, even if some of them have the same name as others and therefore would not be found by `conditions:find-restart`.

It is an error to modify the list returned by `conditions:compute-restarts`.

---

**conditions:restart-name** *restart* [Function]

Returns the name of the given *restart*, or `nil` if it is not named.

---

~~xcl:default-proceed-test~~ *proceed case name* [Macro]

~~Returns the default filter function for proceed cases with the given *proceed case name*. May be used with `setf` to change it.~~

---

~~xcl:default-proceed-report~~ *proceed case name* [Macro]

~~Returns the default report function for proceed cases with the given *proceed case name*. This may be a string or a function just as for condition types. May be used with `setf` to change it.~~

---

**conditions:find-restart** *identifier* [Function]

Searches for a restart by the given *identifier* which is in the current dynamic environment.

If *identifier* is a symbol, then the innermost (ie, most recently established) restart with that name that is active is returned. `nil` is returned if no such restart is found.

If *identifier* is a restart object, then it is simply returned unless it is not currently valid for use. In that case, `nil` is returned.

When searching for a matching restart, the filter function, if any, of potential matches will be called to see if they are active. If it returns

`nil`, then the restart is considered to not have been seen and the search for a match continues.

Although anonymous restarts have a name of `nil`, it is an error for the symbol `nil` to be given as an *identifier* to this function. If it is appropriate to search for anonymous restarts, you should use `conditions:compute-restarts` instead.

---

**`conditions:invoke-restart` *restart* &*rest values***

[Function]

Calls the function associated with the given *restart*, passing the *values* as arguments. The *restart* must be a restart object or the non-null name of a restart which is valid in the current dynamic context. If an argument is not valid, an error of type `conditions:control-error` will be signalled.

~~If the argument is a named proceed case that has a corresponding proceed function, `xcl:invoke-proceed-case` will do the optional argument resolution specified by that function before transferring control to the proceed case.~~

---

**`conditions:invoke-restart-interactively` *restart***

[Function]

Calls the function associated with the given *restart*, providing for any necessary arguments. The *restart* must be a restart object or the non-null name of a restart which is valid in the current dynamic context. If the *restart* is not valid, an error of type `conditions:control-error` will be signalled.

`conditions:invoke-restart-interactively` will first call the *restart*'s interactive function as specified by the `:interactive` keyword of `conditions:restart-case` or the `:interactive-function` keyword of `conditions:restart-bind`. The interactive function should return a list of values to be passed as arguments to the *restart*. This list must be at least as long as the number of required arguments that the *restart* has.

If the *restart* has no interactive function, no arguments will be passed to the restart function. It is an error for a restart to require arguments but not have an interactive function.

Once the arguments have been determined, `conditions:invoke-restart-interactively` will simply do `(apply #'conditions:invoke-restart restart arguments)`.

---

**`conditions:with-simple-restart` (*name* *format-string* {*format-arguments*}\*) {*form*}\***

[Macro]

This is a shorthand for one of the most common uses of `conditions:restart-case`.

If the *restart* designated by *name* is not invoked while executing the *forms*, all values produced by the last *form* are returned. If the restart established by `conditions:with-simple-restart` is

invoked, control is transferred to the `conditions:with-simple-restart` form, which immediately returns the two values `nil` and `t`.

It is permissible for *name* to be `nil`. In that case, an anonymous restart is established.

`conditions:with-simple-restart` is essentially:

```
(defmacro conditions:with-simple-restart
 ((restart-name format-string
 &rest format-arguments)
 &body forms)
 `(conditions:restart-case (progn ,@forms)
 (,restart-name ()
 :report (lambda (stream)
 (format stream
 ,format-string
 ,@format-arguments)))
 (values nil t))))
```

Example:

```
(defun read-eval-print-loop (level)
 (conditions:with-simple-restart
 (conditions:abort "Exit command level ~D." level)
 (loop
 (conditions:with-simple-restart
 (conditions:abort "Return to command level ~D."
 level)
 (print (eval (read)))))))
```

### **xcl:catch-abort** *print-form* &**body forms**

[Macro]

Like `conditions:with-simple-restart`, but always uses the name `conditions:abort`.

`xcl:catch-abort` could be defined by:

```
(defmacro xcl:catch-abort (print-form
 &body forms)
 `(conditions:with-simple-restart
 (conditions:abort ,print-form
 ,@forms))
```

### **conditions:abort**

[Function]

This function transfers control to the nearest active restart named `conditions:abort`. If there is none, this function signals an error of type `conditions:control-error`.

~~`xcl:abort` could be defined by:~~

```
(define-procedure-function xcl:abort
 — :report "Abort")
```

### **conditions:continue**

[Function]



This function transfers control to the nearest active restart named `conditions:continue`. If none exists it simply returns `nil`.

The `conditions:continue` restart is generally part of simple protocols where there is a single "obvious" way to continue, such as in `break` and `cerror`.

NB: `conditions:continue` replaces `xcl:proceed`.

~~`xcl:proceed` & optional `condition` [Function]~~

~~This is a predefined `proceed` function. It is used by such functions as `break`, `cerror`, etc.~~

**`conditions:muffle-warning` [Function]**

This function transfers control to the nearest active restart named `conditions:muffle-warning`. If none exists, an error of type `conditions:control-error` is signalled.

`warn` sets up this restart so that handlers of `conditions:warning` conditions have a way to tell `warn` that the warning has been dealt with and that no further action is warranted.

**`conditions:use-value` *new-value* [Function]**

This function transfers control (and one value) to the nearest active restart named `conditions:use-value`. If no such restart exists, this function simply returns `nil`.

The `conditions:use-value` restart is generally used by handlers trying to recover from errors of types such as `conditions:cell-error`, where the handler may wish to supply a replacement datum for one-time use.

**`conditions:store-value` *new-value* [Function]**

This function transfers control (and one value) to the nearest active restart named `conditions:store-value`. If no such restart exists, this function simply returns `nil`.

The `conditions:use-value` restart is generally used by handlers trying to recover from errors of types such as `conditions:cell-error`, where the handler may wish to supply a replacement datum to be stored in the offending cell.

[This page intentionally left blank]

COVER TEXT:

NOTE: USE STANDARD 8-1/2X11 BINDER COVERS, 3-ring, D-ring binder, 2 1/4 inch spine

ENVOS NAME/LOGO HERE  
(probably sticker)

**ARTIFICIAL INTELLIGENCE SYSTEMS**

**LISP RELEASE NOTES**

**MEDLEY RELEASE**

ENVOS PART NO. XXXXX  
(probably sticker)

HELVETICA  
BOLD ITALIC

MODERN  
BOLD ITALIC

SPINE (2-1/4 INCH) :

**ENVOS**

**ARTIFICIAL INTELLIGENCE SYSTEMS**

**LISP RELEASE NOTES**

**MEDLEY RELEASE**

HELVETICA  
BOLD ITALIC

MODERN  
BOLD ITALIC

# LISP RELEASE NOTES



## LISP RELEASE NOTES

Medley Release

September 1988

The information in this document is subject to change without notice and should not be construed as a commitment by Envos Corporation. While every effort has been made to ensure the accuracy of this document, Envos Corporation assumes no responsibility for any errors that may appear.

Copyright © 1988 by Envos Corporation.

Xerox Common Lisp is a trademark.

All rights reserved.

"Copyright protection claimed includes all forms and matters of copyrightable material and information now allowed by statutory or judicial law or hereinafter granted, including, without limitation, material generated from the software programs which are displayed on the screen, such as icons, screen display looks, etc."

This manual is set in Modern and Terminal typefaces with text written and formatted on Xerox Artificial Intelligence workstations. Xerox laser printers were used to produce text masters.

## A

**Abort** (*Editor Command*) B.7  
**ACCESS** 3.38  
**Add-Command** (*Function*) B.14  
**add.process** (*Function*) 4.12; 7.12  
**ADDMENU** (*Function*) 4.24  
**ADDTOSCRATCHLIST** (*Function*) 4.1  
**ADVICE** (*File Manager Command*) 3.15  
**ADVINFOLST** (*Variable*) 3.14  
**ADVISE** (*File Manager Command*) 3.15  
**ADVISE** (*Function*) 3.13,15  
**ADVISEDfNS** (*Variable*) 3.14  
**ADVISEDUMP** (*Function*) 3.14  
**Advising** 3.14; 7.9  
**AFTERDOMAKESYS** 4.7  
**AFTERDOSAVEVM** 4.7  
**AFTERDOSYSOUT** 4.7  
**AFTERLOGOUT** 4.7  
**AFTERLOGOUTFORMS** 4.7  
**AFTERMAKESYS** 4.7  
**AFTERSAVEVM** 4.7  
**AFTERSYSOUT** 4.7  
**AGAIN** (*Editor Command*) B.8  
**ALL** (*Event Address*) A-5  
**ALLOWED.LOGINS** 4.6  
**append** (*Function*)  
     with non-list argument 7.8  
**Application Menus** D.1  
**APPLY-format input** A-3  
**ARCHIVEFLG** (*Variable*) 3.9  
**ARCHIVEFN** (*Variable*) 3.9  
**Arglist** (*Editor Command*) B.9  
**AROUNDEXITFNS** (*Variable*) 4.7  
**array reference** 7.4  
**arrays** 3.3  
**ASKUSER** (*Function*) 4.16  
**assert** (*Macro*) E.10  
**Attach Menu** (*Editor Command*) B.11  
**Attached Windows** 4.28  
**AUTHENTICATE** 4.6  
**AUTHENTICATION.NET.HINT** (*Variable*) 4.33  
**AUTOHARDRESETFLG** 4.5

## B

**back-quote facility** 3.49  
**BACKGROUND** (*FreeMenu Group Property*) D.8  
**BACKGROUND** (*FreeMenu Item Property*) D.10  
**BACKGROUNDfNS** (*Variable*) 4.12  
**BACKSPACE** (*Editing Command*) A-21  
**BCOMPL** (*Function*) 3.22,25; 4.10  
**BEEPON** (*Function*) 4.31  
**BEFORELOGOUT** 4.7  
**BEFOREMAKESYS** 4.7  
**BEFORESAVEVM** 4.7  
**BEFORESYSOUT** 4.7  
**BEFORESYSOUTFORMS** 4.7

**BITMAP** (*FreeMenu System Property*) D.10  
**BKSYSBUF** (*Function*) 4.30  
**BKSYSCHARCODE** (*Function*) 4.30  
**BLOCKRECORD** (*Record Type*) 4.3  
**BOTTOM** (*FreeMenu Group Property*) D.7  
**bound** E.4  
**BOUNDp** (*Function*) 3.2  
**BOX** (*FreeMenu Group Property*) D.5,8  
**BOX** (*FreeMenu Item Property*) D.10  
**BOXSHADE** (*FreeMenu Group Property*) D.8  
**BOXSHADE** (*FreeMenu Item Property*) D.10  
**BOXSPACE** (*FreeMenu Group Property*) D.8  
**BOXSPACE** (*FreeMenu Item Property*) D.10  
**break** (*Function*) 3.13; E.13  
**break commands** 3.13  
**Break packages** 3.9  
**BREAK0** (*Function*) 3.13  
**BREAK1** (*Function*) 3.9  
**BREAKCONNECTION** (*Function*) 4.14  
**BREAKIN** (*Function*) 3.13  
**breaking** 7.9  
**BREAKREGIONSPEC** (*Variable*) 4.8  
**BRECOMPILE** (*Function*) 3.22,25  
**BRKINFOLST** (*Variable*) 3.13  
**BROKENfNS** (*Variable*) 3.13  
**bulk data transfer** 4.34

## C

**Catch errors** 3.10  
**ccase** (*Macro*) E.10  
**cerror** (*Function*) E.9  
**Change Print Base** (*Editor Command*) B.11  
**CHANGEBACKGROUND** (*Function*) 4.31  
**CHANGEFONT** (*Function*) 4.23  
**CHANGESLICE** (*Function*) A-11,17  
**CHANGESTATE** (*FreeMenu Item Property*) D.11  
**changing a standard readable** 3.22  
**characters** 3.3  
**CHARCODE** (*Function*) 3.3  
**CHCON** (*Function*) 3.42  
**check-type** (*Macro*) E.10  
**CL Exec** 3.7  
**CL:\*** (*Variable*) A-10  
**CL:\*\*** (*Variable*) A-10  
**CL:\*\*\*** (*Variable*) A-10  
**CL:+** (*Variable*) A-10  
**CL:++** (*Variable*) A-10  
**CL:+++** (*Variable*) A-10  
**CL:-** (*Variable*) A-10  
**CL:/** (*Variable*) A-10  
**CL:II** (*Variable*) A-10  
**CL:III** (*Variable*) A-10  
**CL:BREAK** (*Function*) 3.13  
**CL:CATCH** (*Function*) 3.5  
**CL:CODE-CHAR** (*Function*) 3  
**CL:COMPILE-FILE** (*Function*) 3.24-25; 4.10  
**CL:DEFCONSTANT** (*Variable*) 3.20  
**CL:DEFINE-MODIFY-MACRO** (*Function*) 3.20  
**CL:DEFMACRO** (*Function*) 3.20

- CL:DEFMACRO** (Macro) 3.29
- CL:DEFPARAMETER** (Macro) 3.26,29
- CL:DEFPARAMETER** (Variable) 3.20
- CL:DEFUN** (Function) 3.20
- CL:DEFUN** (Macro) 3.29
- CL:DEFVAR** (Macro) 3.29
- CL:DEFVAR** (Variable) 3.20
- CL:ERROR** 3.10
- CL:EVAL-WHEN** (File Package Command) 3.31
- CL:GENSYM** (Function) 3.2
- CL:LOAD** (Function) 3.24
- CL:MAKE-HASH-TABLE** (Function) 3.4
- CL:MAPHASH** (Function) 3.4
- CL:PRIN1** (Function) 3.41-42
- CL:PRINC** (Function) 3.41
- CL:READ** (Function) 3.40
- CL:READ-PRESERVING-WHITESPACE** (Function) 3.41
- CL:THROW** (Function) 3.5,11
- CL:UNWIND-PROTECT** 3.6
- CL:UNWIND-PROTECT** (Function) 3.11
- CL:WITH-INPUT-FROM-STRING** 3.37
- CL:WRITE** (Function) 3.41
- CLEANUP** (Function) 3.25
- cleanup forms 3.6
- CLEARCLISPARRAY** (Function) 4.10
- CLEARSTK** (Function) 4.5
- CLEARSTKLST** (Variable) 4.5
- CLISP infix forms 3.33
- CLISPARRAY 4.2
- CLOSEALL** (Function) 3.38
- closure 7.8
- coerce** (Function) 7.12
- COERCE-TO-NSADDRESS** (Function) 4.33
- collect** (Macro) 7.6
- collecting objects
  - macros for 7.6
- COLLECTION** (FreeMenu Item Property) D.12
- COLLECTION property 4.26
- COLUMN** (FreeMenu Group Property) D.7
- COLUMNSPACE** (FreeMenu Group Property) D.7
- Comment Out Selection** (Editor Command) B.9
- comment treated as declaration 3.32
- Comments
  - in SEdit B.6
- Common Lisp strings 3.3
- Common Lisp Symbols 3.1
- COMMONNUMSYNTAX 3.44
- compile-definer** (Definer) 7.2
- compile-form** (Definer) 7.2
- compiler
  - behavior with FLETed lexical functions 7.12
  - behavior with recursion 7.12
  - ignoring TEdit formatting 7.12
  - retaining special arguments 7.12
- complex numbers 3.4
- coms 7.11
- condition E.3
- conditions:\*break-on-signals\*** (Variable) E.9
- conditions:abort** (Function) E.21
- conditions:compute-restarts** (Function) E.18
- conditions:continue** (Function) E.21
- conditions:define-condition** (Macro) E.5
- conditions:find-restart** (Function) E.19
- conditions:handler-bind** (Macro) E.4,11
- conditions:handler-case** (Macro) E.11
- conditions:ignore-errors** (Macro) E.12
- conditions:invoke-debugger** (Function) E.13
- conditions:invoke-restart** (Function) E.5,20
- conditions:invoke-restart-interactively** (Function) E.20
- conditions:make-condition** (Function) E.6,8
- conditions:muffle-warning** (Function) E.22
- conditions:restart-bind** (Macro) E.17
- conditions:restart-case** (Function) E.5
- conditions:restart-case** (Macro) E.13
- conditions:restart-name** (Function) E.19
- conditions:signal** (Function) E.8
- conditions:store-value** (Function) E.22
- conditions:use-value** (Function) E.22
- conditions:with-simple-restart** (Macro) E.20
- CONN** (Exec Command) A-7
- CONTROL-A** (Editing Command) A-21
- Control-C** (Editor Command) B.7
- Control-L** (Editor Command) B.7
- Control-Meta-;** (Editor Command) B.9
- Control-Meta-F** (Editor Command) B.8
- Control-Meta-O** (Editor Command) B.7
- Control-P 4.29
- CONTROL-Q** (Editing Command) A-21
- CONTROL-R** (Editing Command) A-21
- Control-T 4.29
- CONTROL-W** (Editing Command) A-21
- Control-W** (Editor Command) B.7
- CONTROL-X** (Editing Command) A-21
- Control-X** (Editor Command) B.7
- Convert Comments** (Editor Command) B.9
- Convert-Upgrade** (Variable) B.14
- converting characters 3.3

Converting old code  
     for use with new Error system E.1  
**COORDINATES** (*FreeMenu Group Property*) D.7  
**COPY** (*Function*) 3.49  
**COPYBYTES** (*Function*) 4.16  
**COPYDEF** (*Function*) 4.4  
**COPYFILE** (*Function*) 3.38  
**COPYREADTABLE** (*Function*) 3.46  
**COS** (*Function*) 4.3  
**COURIER.CALL** (*Function*) 4.34  
**COURIER.OPEN** (*Function*) 4.34  
 Creating an Exec process A-18  
 Creating conditions E.4  
 Creating icons  
     with ICONW C.1  
**CTRLUFLG** 4.18  
**ctypcase** (*Macro*) E.10  
**CUHOTSPOTX** 4.30  
**CUHOTSPOTY** 4.30  
**CUIMAGE** 4.30  
 current package 3.45  
**CURSOR** 4.30  
 Cursor Movement Commands A-22  
**CURSORBITMAP** 4.30  
**CURSORCREATE** (*Function*) 4.30  
**CURSORHOTSPOTX** 4.30  
**CURSORHOTSPOTY** 4.30

## D

**DA** (*Exec Command*) A-7  
**DAUGHTERS** (*FreeMenu Group Property*) D.8  
**DC** (*Function*) 3.18  
 Declining by Condition handler E.4  
**DEdit** 3.15  
 Default handlers 3.10  
**Default-Commands** (*Function*) B.15  
**DEFAULT.OSTYPE** (*Variable*) 4.15  
**DEFAULTFONT** (*Variable*) D.7  
**DEFAULTICONFN** (*Variable*) 4.25  
**DEFAULTTEXTICON** (*Variable*) C.3  
**deferredconstant** (*Function*) 7.12  
**define-file-environment** (*Definer*) 7.2  
**define-record** (*Definer*) 7.3  
 Defining New Terms A-11  
**DEFMACRO** (*Macro*) 3.5  
**defstruct** (*Macro*) 7.4  
     warning 7.6  
**DELDEF** (*Function*) 3.28  
**Delete Selection** (*Editor Command*) B.7  
**Delete Structure** (*Editor Command*) B.8  
**Delete Word** (*Editor Command*) B.7  
**DELFILE** (*Function*) 3.38  
**DESELECT** (*FreeMenu Item Property*) D.12  
**DF** (*Function*) 3.18  
 DFASL files 2.1  
**DFNFLG** (*Variable*) 3.27  
**DIR** (*Exec Command*) A-7  
**DISPLAY** (*FreeMenu Item*) D.6-7,14  
 Display icons C.1  
 DISPLAY item 4.26  
**DISPLAYFONTDIRECTORIES** (*Variable*) 4.23  
**DMACRO** (*Property*) 3.5  
 DMACROS 2.1  
**DO-EVENTS** (*Exec Command*) A-8  
**DOCOLLECT** (*Function*) 4.1  
**DOSHAPEFN** (*Window Property*) 4.25  
**DOWNFN** (*FreeMenu Mouse Property*) D.10  
**DP** (*Function*) 3.18

**DRAWARC** (*Function*) 4.19  
**DRAWLINE** (*Function*) 4.19  
**DRAWPOLYGON** (*Function*) 4.20  
**DSPCLEOL** (*Function*) 4.18  
**DSPFONT** 4.16  
**DSPRUBOUTCHAR** (*Function*) 4.18  
**DSPSCALE** 4.19  
 dummy definitions 3.17  
**DV** (*Function*) 3.18  
**DWIMIFYCOMPFLG** (*Variable*) 3.34

## E

**ecase** (*Macro*) E.10  
**ECHOCHAR** (*FreeMenu Item Property*) D.13  
**ED** (*Function*) 3.16  
**Edit** (*Editor Command*) B.9  
**EDIT** (*FreeMenu Item*) 4.27; D.13  
 Edit caret in SEdit B.2  
 Edit Interface 3.18  
**EDITBM** (*Function*) 4.18  
**EDITCALLERS** (*Function*) 3.19  
 Editing Exec Input A-20  
 Editing Lisp Code in Memory B.1  
 Editing VALUES 3.18  
**EDITMODE** (*Function*) 3.16  
**EDITSTART** (*FreeMenu Item*) 4.27; D.14  
**END-OF-FILE** (*Error Type*) 3.12  
**ENDCOLLECT** (*Function*) 4.1  
 Ending an SEdit session B.2  
**ENDOFSTREAMOP** 3.38  
**ENVAPPLY** 3.6  
**ENVEVAL** 3.6  
**EQUAL** (*Function*) 3.26  
**EQUALALL** (*Function*) 4.3  
**ERROR** (*Function*) 3.10  
**error** (*Function*) E.9  
 Error conditions 3.10  
 error system 3.10  
 Error system  
     differences between old and new E.1  
 Error system proposal E.1  
 Error type mapping 3.11  
 Error type name 3.11  
 Error type number 3.11  
**ERROR!** (*Function*) 3.10  
**ERRORMESS** (*Function*) 3.10  
**ERRORMESS1** (*Function*) 3.10  
**ERRORN** (*Function*) 2.2; 3.10  
 Errors  
     definition of E.3  
**ERRORSET** 3.10  
**ERRORSTRING** (*Function*) 3.10  
**ERRORTYPELIST** 3.10  
**ERRORTYPELIST** (*Variable*) 2.2  
**ERSETQ** (*Function*) 3.10; 4.8  
**ERXM** 3.10  
**ESCAPE** (*Editing Command*) A-21  
 Escape  
     in SEdit B.6  
 Establishing handlers within dynamic context E.4  
**etypcase** (*Macro*) E.10  
**Eval** (*Editor Command*) B.9  
**EVAL-format input** A-2  
 Exec Editing Commands A-22  
 Exec type A-4  
**EXEC-EVAL** (*Function*) 3.9  
**EXPAND** (*Editor Command*) B.9  
**EXPANDBITMAP** (*Function*) 4.18



**EXPANDMACRO** (Function) 3.5  
**EXPANDREGIONFN** (Window Property) 4.24  
**EXPLICIT** (FreeMenu Group Property) D.7  
**export** (Function) 7.9  
**Extract** (Editor Command) B.9

**F**  
**F** (Event Address) A-5  
features  
    new Common Lisp 7.1  
**FETCH** 3.33  
File Manager 3.19  
file-reading functions 3.20  
**FILEPKGCOM** (Function) 4.9  
**FILEPKGTYPE** (Function) 4.9  
**FILEPKGYPES** (Variable) 3.16  
**FILEPOS** (Function) 4.16  
**FILERDTBL** 3.22  
files containing bitmaps 3.31  
**FILES?** (Function) 3.28  
**FILETYPE** (Property) 3.25  
**FILLPOLYGON** (Function) 4.19-20  
**FIND** (Editor Command) B.8  
**Find Gap** (Editor Command) B.8  
**FIND-READTABLE** (Function) 3.45  
**FINDCALLERS** (Function) 3.19  
**FIX** (Exec Command) A-8  
**FIXP** (Predicate) 3.4  
**flet** (Special form) 7.4  
floating point 3.4  
**FLOATP** (Predicate) 3.4  
**FM.BACKGROUND** (FreeMenu Window Property) D.15  
**FM.CHANGELABEL** (FreeMenu Function) D.16  
**FM.CHANGELABEL** (Function) 4.27-28  
**FM.CHANGESTATE** (FreeMenu Function) D.16  
**FM.CHANGESTATE** (Function) 4.28  
**FM.DONTRESHAPE** (FreeMenu Window Property) D.15  
**FM.EDITITEM** (FreeMenu Function) D.17  
**FM.EDITP** (FreeMenu Function) D.17  
**FM.ENEDIT** (FreeMenu Function) D.17  
**FM.FIXSHAPE** (Function) 4.28  
**FM.FORMATMENU** (Function) 4.26-27  
**FM.GETITEM** (Function) 4.27  
**FM.GETITEM** (FreeMenu Function) D.15  
**FM.GETSTATE** (FreeMenu Function) D.16  
**FM.GETSTATE** (Function) 4.27  
**FM.GROUPPROP** (FreeMenu Macro) D.7,18  
**FM.HIGHLIGHTITEM** (FreeMenu Function) D.17  
**FM.HIGHLIGHTITEM** (Function) 4.28  
**FM.ITEMFROMID** (Function) 4.27  
**FM.ITEMPROP** (FreeMenu Macro) D.18  
**FM.MAKEMENU** (Function) 4.26-27  
**FM.MENUPROP** (FreeMenu Macro) D.7,19  
**FM.NWAYPROP** (FreeMenu Macro) D.19  
**FM.NWAYPROPS** (Macro) 4.27  
**FM.PROMPTWINDOW** (FreeMenu Window Property) D.15  
**FM.READSTATE** (Function) 4.27  
**FM.REDISPLAYITEM** (FreeMenu Function) D.18  
**FM.REDISPLAYMENU** (FreeMenu Function) D.18  
**FM.RESETGROUPS** (FreeMenu Function) D.17  
**FM.RESETMENU** (FreeMenu Function) D.17  
**FM.RESETSHAPE** (FreeMenu Function) D.17  
**FM.RESETSHAPE** (Function) 4.28  
**FM.RESETSTATE** (FreeMenu Function) D.17  
**FM.SHADE** (FreeMenu Function) D.18

**FM.SHADE** (Function) 4.28  
**FM.SHADEITEM** (Function) 4.28  
**FM.SHADEITEMBM** (Function) 4.28  
**FM.SKIPNEXT** (FreeMenu Function) D.17  
**FM.TOPGROUPID** (FreeMenu Function) D.18  
**FM.WHICHITEM** (FreeMenu Function) D.18  
**FONT** (FreeMenu Group Property) D.7  
**FONT** (FreeMenu Item Property) D.9  
font descriptor 4.22  
**FONTCHANGEFLG** (Variable) 4.23  
**FONTCREATE** (Function) 4.22  
**FONTSAVAILABLE** 4.21  
**FOR** 3.33  
**FOR** (Exec Command) A-6  
**FORGET** 4.6  
**FORGET** (Exec Command) A-8  
**FORMAT** (FreeMenu Group Property) D.4,7  
Free Menu  
    How to make a D.1  
Free Menu format D.2  
Free Menu layout D.1  
**FREEMENU** (FreeMenu Function) D.15  
**FREEMENU** (Function) 4.26-27  
**FROM** (Event Address) A-5  
**FULLNAME** (Function) 3.37  
**FUNARG** 4.4

## G

Gaps  
    in SEdit B.4  
garbage collector 4.11  
**gensym** (Function) 3.2; 7.12  
**GET-ENVIRONMENT-AND-FILEMAP** (Function) 3.23  
**Get-Prompt-Window** (Function) B.15  
**Get-Selection** (Function) B.16  
**Get-Window-Region** (Function) B.13  
**GETDEF** (Function) 3.28  
**GETFILEINFO** (Function) 3.38; 4.13  
**GETPROMPTWINDOW** (Function) 4.28  
**GETREADTABLE** (Function) 3.39  
**GETSYNTAX** 3.45  
global macro shadowing 7.4  
**GROUP** (FreeMenu Group Property) D.7  
**GROUPID** (FreeMenu System Property) D.10

## H

**handler** (Function) E.4  
Handling conditions E.3  
**HARDCOPYW** (Function) 4.29  
**HARDRESET** (Function) 4.4  
**HASDEF** (Function) 3.26,28; 4.9  
hash arrays 3.4  
**HASHARRAY** 3.4  
**HASHARRAY** (Function) 4.2  
**HELDFN** (FreeMenu Mouse Property) D.10  
**HELP** (Editor Command) B.9  
**HELP** (Function) 3.10  
Help Menu Commands B.11  
**HIGHLIGHT** (FreeMenu Item Property) D.9,14  
History list A-16  
**HISTORYSAVEFORMS** (Variable) 3.9  
**HJUSTIFY** (FreeMenu Item Property) D.4,9  
**HORRIBLEVARS** 4.9,15  
**HPRINT** (Function) 4.15

**I**

**ICONW** (Function) C.1  
 ICONW windows  
   from an image defined by a mask C.1  
   with titles C.1  
**ICONW.SHADE** (Function) C.2  
**ICONW.TITLE** (Function) C.2  
**ID** (FreeMenu Group Property) D.7  
**ID** (FreeMenu Item Property) D.9  
**IDLE.PROFILE** 4.6  
**IDLE.RESETVARS** (Variable) 4.6  
**IDLE.SUSPEND.PROCESS.NAMES** (Variable) 4.7  
 IEEE 802.3 specification 4.34  
**IF** 3.33  
 IL Exec 3.7  
**IL:IT** (Variable) A-9  
**IL:LOAD** (Function) 3.24  
**IL:MAPHASH** (Function) 3.4  
**IL:PRIN1** (Function) 3.41  
**IL:PRIN2** (Function) 3.41  
**IL:READ** (Function) 3.40  
**ILLEGAL-GO** (Error Type) 3.11  
**ILLEGAL-RETURN** (Error Type) 3.11  
**ILLEGAL-STACK-ARG** (Error Type) 3.12  
**IN** (Exec Command) A-6  
**in-package** (Function) 7.8  
**INFILEP** (Function) 3.37  
**INFINITEWIDTH** (FreeMenu Item Property) D.13  
**INITSTATE** (FreeMenu Item Prop) 4.26  
**INITSTATE** (FreeMenu Item Property) D.9,12  
**INPUT** (Function) 3.37  
**INPUTFONT** (Variable) A-10  
**Inspect** (Editor Command) B.10  
**INTEGERLENGTH** (Function) 4.3  
 integers 3.4  
 Interlisp Compiler 3.31  
**INTERLISP-ERROR** (Error Type) 3.12  
**INTERPRESSFONTDIRECTORIES** (Variable) 4.22  
**INTERRUPTCHAR** (Function) 4.29  
**INVALID-ARGUMENT-LIST** (Error Type) 3.12  
**ITEMS** (FreeMenu Group Property) D.8

**J**

**Join** (Editor Command) B.10

**K**

**Keep-Window-Region** (Variable) B.13  
**KEYACTION** (Function) 4.31  
**KEYDOWNP** (Function) 4.31

**L**

**LABEL** (FreeMenu Item Property) D.9  
 LABELS construct  
   warning 7.10  
**LASTC** (Function) 4.15  
 Layout  
   of Free Menu D.1  
 LCOM files 2.1  
 ldflg 7.11  
**LEFT** (FreeMenu Group Property) D.7  
**LEFT and BOTTOM** (FreeMenu Item Property) D.9  
 Left mouse button  
   in SEdit B.3  
 lexical bindings 3.33  
 Library modules  
   summary of changes 5.1  
**LIMITCHARS** (FreeMenu Item Property) D.3,13

**LINKS** (FreeMenu Item Property) D.10,15  
**LISP** 3.47  
 Lisp structures  
   SEdit gaps for B.4  
**LISPOURCEFILEP** (Function) 4.10  
**LISPXEVAL** (Function) 3.9  
**LISPXFNS** (Variable) A-15  
**LISPHISTORY** (Variable) A-16  
**LISPHISTORYMACROS** (Variable) 3.9  
**LISPMACROS** 3.8  
**LISPMACROS** (Variable) 3.9  
**LISPXREADFN** (Function) 4.8  
**LISPXUNREAD** (Function) 3.9  
**LISPXUSERFN** (Variable) 3.9  
**LIST** (Function) 3.49  
 Lists

  in SEdit B.5  
**LOAD** (Function) 3.20  
   **loadflg** (Argument) 7.11  
 load-time expression 7.4  
**LOADCOMP** (Function) 3.25  
**LOADFNS** (Function) 3.20,25  
**LOADFROM** (Function) 3.25  
 loading compiled files 3.32  
 loading Medley files into Lyrice 4.10  
**LOADVARS** (Function) 3.25  
 Locally defined handler E.4  
**LOCALVARS** 3.2  
**LOGIN.TIMEOUT** 4.6  
**LOGOUT** (Function) 4.7  
**long-site-name** (Variable) 7.3

**M**

**MACHINETYPE** (Function) 4.7  
**MAKE-READER-ENVIRONMENT** (Function) 3.23  
**MAKEFILE** (Function) 3.20,25,43,49  
**MAKEFILE-ENVIRONMENT** (Property) 3.21  
**MAKESYS** (Function) 4.7  
**MAKETITLEBARICON** 4.25  
**map** (Function) 7.11  
**MAPATOMS** (Function) 3.2-3  
**MAX** (Function) 4.2  
**MAX.INTEGER** (Variable) 4.2  
**MAXHEIGHT** (FreeMenu Item Property) D.9  
**MAXREGION** (FreeMenu System Property) D.11  
**MAXWIDTH** (FreeMenu Item Property) D.7,9,13  
 Medley  
   on Sun workstations 1.1  
   on Xerox workstations 1.1  
 Medley compiled files 2.1  
 Medley enhancements  
   summary 1.1  
**MENU** (FreeMenu Group Property) D.7  
**MENUFONT** (FreeMenu Item Property) D.12  
**MENUITEMS** (FreeMenu Item Property) D.6,12  
**MENUTITLE** (FreeMenu Item Property) D.12  
**MESSAGE** (FreeMenu Item Property) D.9  
**Meta-** (Editor Command) B.10  
**Meta-)** (Editor Command) B.10  
**Meta-/** (Editor Command) B.9  
**Meta-9** (Editor Command) B.10  
**Meta-;** (Editor Command) B.9  
**Meta-A** (Editor Command) B.7  
**Meta-B** (Editor Command) B.11  
**Meta-Control-C** (Editor Command) B.7  
**Meta-Control-S** (Editor Command) B.8  
**Meta-Control-X** (Editor Command) B.7  
**Meta-E** (Editor Command) B.9

**Meta-F** (Editor Command) B.8  
**Meta-H** (Editor Command) B.9  
**Meta-I** (Editor Command) B.10  
**Meta-J** (Editor Command) B.10  
**Meta-M** (Editor Command) B.11  
**Meta-N** (Editor Command) B.8  
**Meta-O** (Editor Command) B.9  
**Meta-P** (Editor Command) B.11  
**Meta-R** (Editor Command) B.8  
**Meta-Return** (Editor Command) B.10  
**Meta-S** (Editor Command) B.8  
**Meta-Space** (Editor Command) B.10  
**Meta-U** (Editor Command) B.7  
**Meta-X** (Editor Command) B.9  
**Meta-Z** (Editor Command) B.10  
Middle mouse button  
    in SEdit B.3  
**MIN** (Function) 4.2  
**MIN.INTEGER** (Variable) 4.2  
minimum window size 4.24  
**MKSTRING** (Function) 3.42  
**MOMENTARY** (FreeMenu Item) D.11  
**MOTHER** (FreeMenu Group Property) D.8  
Mouse buttons  
    in SEdit B.3  
**MOVD** (Function) 4.4  
**MOVEDFN** (FreeMenu Mouse Property) D.10  
multiple escape character 3.42  
Multiple Execs A-4  
multiple streams 3.37  
**MULTIPLE-ESCAPE** 3.45  
**Mutate** (Editor Command) B.10

**N**  
**NAME** (Exec Command) A-8  
**NCHARS** (Function) 3.42  
NCHOOSE item 4.26  
**NDIR** (Exec Command) A-8  
Nesting Free Menu Groups D.2  
**NETWORKOSTYPES** (Variable) 4.15  
**NEW** (MAKEFILE Option) 3.21  
NLAMBDA 3.5  
**NLSETQ** (Function) 3.10; 4.8  
NOBIND 3.2  
**NOCLEARSTKLST** (Variable) 4.5  
**NODIRCORE** (File Device) 4.13  
**Normalize Selection** (Editor Command) B.10  
notational conventions 18  
**NSADDRESS** 4.32  
**NSNAME** 4.32  
**NSNET.DISTANCE** (Function) 4.35  
**NUMBER** (FreeMenu Item) D.14  
**NUMBERP** (Predicate) 3.4  
**NUMBERTYPE** (FreeMenu Item Property) D.14  
**NWAY** (FreeMenu Item) 4.26; D.6; 12  
**NWAYPROPS** (FreeMenu Item Prop) 4.27  
**NWAYPROPS** (FreeMenu Item Property) D.6,12

**O**  
**OLD-INTERLISP-FILE** 3.47  
**OLD-INTERLISP-T** 3.48  
**once-only** (Macro) 7.7  
**OPENFILE** (Function) 3.37  
**OPENFN** (Window Property) 4.25  
**OPENP** (Function) 3.38  
**OPENSTREAM** (Function) 3.11,37  
**OPENSTRINGSTREAM** (Function) 3.37; 4.16

options E.5  
**ORIG** 3.46  
**OUTPUT** (Function) 3.37

**P**  
package delimiter 2.2  
**PACKAGEDELIM** 3.47  
packages 3.19  
**PARSE-NSADDRESS** (Function) 4.33  
**PAT** (Event Address) A-5  
pattern matching 3.6  
**PEEK** (Function) 4.15  
**pkg-goto** (Function) 7.8  
**PL** (Exec Command) A-8  
**PLVLFILFLG** 3.42  
**PP** (Exec Command) A-9  
**PRETTYDEF** (Function) 4.9  
**PRIN1** 4.30  
**PRIN2** 4.30  
**PRINT** (Function) 3.20,48  
**PRINTLEVEL** 4.29  
**PRINTNUM** (Function) 4.15  
**PRINTOUT** 3.43  
**PRINTOUTFONT** (Variable) A-11  
**PRINTSERVICE** (Variable) 4.19  
process status window 4.12  
**PROCESS.APPLY** (Function) 4.12  
**PROCESS.EVAL** (Function) 4.12  
Programmer's interface  
    to SEdit B.12  
**PROMPT#FLG** (Variable) 3.9  
**PROMPTFONT** (Variable) A-10  
**PROMPTCHARFORMS** (Variable) 3.9  
**PROTECTION** 4.13  
**PRXFLG** 3.42  
**PUTDEF** (Function) 3.28

**Q**  
**Quote** (Editor Command) B.10  
Quoted structures  
    in SEdit B.5

**R**  
**RADIX** (Function) 3.44  
ratios 3.4  
**READ** (Function) 3.20,48  
read-eval-print A-1  
read/print consistency 3.44  
**READBUF** (Variable) 3.9  
**READC** (Function) 3.41  
**READER** 4.13  
**READER-ENVIRONMENT** 3.20  
**READLINE** (Function) 4.8  
**READMACROS** 4.16  
**READSYS** (Function) 4.35  
**READTABLEPROP** (Function) 3.45  
**READVISE** (Function) 3.14  
**REALFRAMEP** (Function) 4.5  
**REBREAK** (Function) 3.14  
**RECOMPILE** (Function) 3.22,25  
**record-create** (Macro) 7.4  
**record-fetch** (Macro) 7.4  
**record-ffetch** (Macro) 7.4  
**Redisplay** (Editor Command) B.7  
**Redo** (Editor Command) B.8  
**REDO** (Exec Command) A-6  
**REGION** (FreeMenu Group Property) D.8

**REGION** (*FreeMenu System Property*) D.11

**RELDRAWTO** (*Function*) 4.19

Release Notes

organization of 17

**REMEMBER** (*Exec Command*) A-8

**REMPROP** (*Function*) 3.2

**RENAMEFILE** (*Function*) 3.38

**REPAINTFN** 4.24

**REPAINTFN** (*Window Property*) 4.25

**REPEATUNTIL** 4.3

**Replace-Selection** (*Function*) B.16

Reporting a condition or restart E.5

**Reset** (*Function*) 3.10; B.14

**Reset-Commands** (*Function*) B.15

**RESETFORM** 3.40

**RESETFORM** 3.39

**RESETFORMS** (*Variable*) 3.9

**RESETLST** 3.6

Resetting system state 3.11

**RESETVARS** 4.6

**RESHAPEFN** 4.24

Restart type E.5

Restarting computations E.3

Restarting conditions E.5

**RETAPPLY** 3.6

**RETEVAL** 3.6

**RETFROM** 3.6

**RETFROM** (*Function*) 3.11

**RETRY** (*Exec Command*) A-6

**RETTO** 3.6

**RETURN** 3.13; 4.5

**Reverse Find** (*Editor Command*) B.8

Right mouse button

in SEdit B.3

**ROTATE-BITMAP** (*Function*) 4.18

**ROW** (*FreeMenu Group Property*) D.7

**row-major-aref** (*Function*) 7.4

**ROWSPACE** (*FreeMenu Group Property*) D.7

RS232 or TTY ports 3.38

## S

**Save-Window-Region** (*Function*) B.13

**SAVEVM** (*Function*) 4.7

**SCRATCHLIST** 4.1

SEdit 3.15

**SEdit** (*Function*) B.16

SEdit Command Menu B.12

**SEE** (*Exec Command*) A-9

**SEE\*** (*Exec Command*) A-9

**SELECTEDFN** (*FreeMenu Mouse Property*) D.10

**Set Package** (*Editor Command*) B.11

**SETERRORN** (*Function*) 3.10

**SETFILEINFO** (*Function*) 3.38; 4.13

**SETREADTABLE** (*Function*) 3.48

**SETSTKARGNAME** (*Function*) 4.5

**SETSYNTAX** 3.45, 49

**SHAPEW** (*Function*) 4.24

**SHH** (*Exec Command*) A-8

**SHIFT-FIND** (*Editor Command*) B.8

**short-site-name** (*Variable*) 7.3

**SHOULDCOMPILEMACROATOMS** (*Variable*) 4.4

**SHOULNT** (*Function*) 3.10

**SHOWPARENFLG** (*Variable*) A-25

**SHRINKBITMAP** (*Function*) 4.18

**SHRINKFN** (*Window Property*) 4.24

**SIDE** effects of event A-18

Signalling conditions E.3

**SIN** (*Function*) 4.3

Sketch

summary of changes 6.10

**SKIP-NEXT** (*Editor Command*) B.8

**SKREAD** (*Function*) 3.41

**SORT** (*Function*) 4.1

Special characters

in SEdit B.5

Specifying event addresses A-4

Specifying Free Menu Items D.2

stack manipulations 3.5

**STACK OVERFLOW** (*Error Type*) 4.4

Stack pointers 3.5

**STACK-OVERFLOW** (*Error Type*) 3.11

**STACK-POINTER-RELEASED** (*Error Type*) 3.12

Starting an SEdit session B.2

**STATE** 4.26

**STATE** (*FreeMenu Item*) D.7, 11

**STATE** (*FreeMenu Item Property*) D.12

**STATE** (*FreeMenu System Property*) D.10

**STKARG** (*Function*) 4.5

**STKNARGS** (*Function*) 4.5

**STKPOS** (*Function*) 4.5

**STOP** (*Function*) 4.10

**STOP-UNDOABLY** (*Macro*) A-13

strings 3.3

in SEdit B.6

**STRINGWIDTH** (*Function*) 3.42; 4.22

Structure caret in SEdit B.2

Structure editor 3.15

**Substitute** (*Editor Command*) B.8

**SUCHTHAT** (*Event Address*) A-5

**SUSPEND.PROCESS.NAMES** 4.7

Switching between editors 3.16

Symbols 3.1, 6

in Error system E.1

symbols in the INTERLISP package 3.20

**SYSDOWNFN** (*FreeMenu System Property*) D.11

sysload 3.24; 7.11

**SYSMOVEDFN** (*FreeMenu System Property*) D.11

**YSOUT** (*Function*) 4.7

**SYSPRETTYFLG** (*Variable*) 3.9

**SYSSELECTEDFN** (*FreeMenu System Property*) D.11

## T

**TABLE** (*FreeMenu Group Property*) D.7

**TCOMPL** (*Function*) 3.22, 25; 4.10

TEdit

summary of changes 6.1

TeleRaid Library module 4.35

**TEXTICON** (*Function*) 4.25; C.3

**TIME** (*Exec Command*) A-9

**TIME** (*Function*) 3.36

**TIME** (*Macro*) 3.36

**TITLE** (*FreeMenu Item*) 4.27

titled icons 4.25

**TITLEDICONW** (*Function*) C.1

**TOGGLE** (*FreeMenu Item*) D.11

**TOO-MANY-ARGUMENTS** (*Error Type*) 3.12

**TRACE** (*Function*) 3.13-14

**TTYBACKGROUNDFN** (*Variable*) 4.12

**TTYDISPLAYSTREAM** (*Function*) 4.25

TTYIN display typein editor 4.16

TTYIN Editor from Exec A-20

**TY** (*Exec Command*) A-9

**TYPE** (*Exec Command*) A-9

**TYPE** (*FreeMenu Item Property*) D.9

**U**

UGLYVARS 3.43; 4.9,15  
**UNBOUND-VARIABLE** (Error Type) 3.12  
**UNBREAK** (Function) 3.14  
**UNBREAKIN** (Function) 3.13  
**UNDEFINED-CAR-OF-FORM** (Error Type) 3.12  
**UNDEFINED-FUNCTION-IN-APPLY** (Error Type) 3.12  
**UNDO** (Editor Command) B.7  
**UNDO** (Exec Command) A-4,8,13  
**UNDO key** (Editing Command) A-21  
**UNDOABLY-MAKUNBOUND** (Function) 3.29  
**UNDOABLY-SETQ** (Function) A-15  
Undoing in Functions A-14  
Undoing In Programs A-13  
Undoing out of order A-16  
**UNDOSAVE** (Function) A-15  
**UNIXFTPFLG** (Variable) 4.14  
**UNPACKFILENAME** (Function) 3.37  
**UNSAFEMACROATOMS** (Variable) 4.4  
UNTIL 4.3  
**USE** (Exec Command) A-6  
**USERDATA** (FreeMenu System Property) D.11  
USERDATA LIST D.14  
**USEREXEC** (Function) 3.9  
USERNAME 4.6  
**USERWORDS** (Variable) A-25  
USESILPACKAGE 3.45  
Using Execs 3.7

**V**

**VALUEFONT** (Variable) A-11  
VARS 4.15  
version delimiter 2.2  
**VIDEORATE** (Function) 4.31  
**VJUSTIFY** (FreeMenu Item Property) D.9

**W**

**warn** (Function) E.10  
WHENCHANGED 4.9  
**WINDOWPROP** (Function) 4.26  
WINDOWPROPS 4.26  
**with-collection** (Macro) 7.6  
**with-input-from-string** (Macro) 7.13  
**with-output-to-string** (Macro) 7.13  
**WITH-READER-ENVIRONMENT** (Macro) 3.23  
**write-string** (Function) 7.12  
**WRITESTRIKEFONTFILE** (Function) 4.22  
writing macros  
    macros for 7.7  
Writing your own SEdit commands B.14

**X**

XCL 3.47  
XCL Compiler 3.31  
XCL Exec 3.7  
XCL readtable 3.21  
**xcl:\*current-condition\*** (Variable) E.8  
**XCL:\*DEBUGGER-PROMPT\*** (Variable) A-19  
**XCL:\*EVAL-FUNCTION\*** (Variable) A-19  
**XCL:\*EXEC-PROMPT\*** (Variable) A-19  
**XCL:\*PER-EXEC-VARIABLES\*** (Variable) A-18  
**XCL:ABORT** (Function) 3.10  
**XCL:ADD-EXEC** (Function) A-18  
**XCL:ARGLIST** (Variable) 3.15  
**XCL:ARRAY-SPACE-FULL** (Error Type) 3.12

**XCL:ATTEMPT-TO-CHANGE-CONSTANT** (Error Type) 3.11-12  
**XCL:ATTEMPT-TO-RPLAC-NIL** (Error Type) 3.11  
**XCL:CATCH-ABORT** 3.10  
**xcl:catch-abort** (Macro) E.21  
**XCL:CONDITION** 3.10  
**xcl:condition-case** (Macro) E.11  
**xcl:condition-handler** (Macro) E.8  
**xcl:condition-reporter** (Macro) E.7  
**XCL:CONTROL-E-INTERRUPT** (Error Type) 3.12  
**XCL:DATA-TYPES-EXHAUSTED** (Error Type) 3.12  
**XCL:DEF-DEFINE-TYPE** (Macro) 3.27-28  
**XCL:DEFCOMMAND** 3.8  
**XCL:DEFCOMMAND** (Macro) A-11  
**XCL:DEFDEFINER** (Function) 3.20  
**XCL:DEFDEFINER** (Macro) 3.29  
**XCL:DEFGLOBALPARAMETER** (Variable) 3.20  
**XCL:DEFGLOBALVAR** (Variable) 3.20  
**XCL:DEFINE-PROCEED-FUNCTION** (Function) 3.20  
**XCL:DEFINELINE** (Function) 3.20  
**XCL:DEFOPTIMIZER** 3.32  
**XCL:DEFOPTIMIZER** (Macro) 3.5  
**XCL:EXEC** (Function) A-18  
**XCL:EXEC-EVAL** (Function) A-19  
**XCL:EXEC-FORMAT** (Function) A-19  
**XCL:FILE-NOT-FOUND** (Error Type) 3.12  
**XCL:FILE-WONT-OPEN** (Error Type) 3.11  
**XCL:FLOATING-OVERFLOW** (Error Type) 3.12  
**XCL:FLOATING-UNDERFLOW** (Error Type) 3.12  
**XCL:FS-PROTECTION-VIOLATION** (Error Type) 3.12  
**XCL:FS-RESOURCES-EXCEEDED** (Error Type) 3.12  
**XCL:HASH-TABLE-FULL** (Error Type) 3.12  
**XCL:INVALID-PATHNAME** (Error Type) 3.12  
**XCL:SET-DEFAULT-EXEC-TYPE** (Function) A-20  
**XCL:SET-EXEC-TYPE** (Function) A-20  
**XCL:SIMPLE-DEVICE-ERROR** (Error Type) 3.11  
**XCL:SIMPLE-TYPE-ERROR** (Error Type) 3.11  
**XCL:STORAGE-EXHAUSTED** (Error Type) 3.12  
**XCL:STREAM-NOT-OPEN** (Error Type) 3.11  
**XCL:SYMBOL-HT-FULL** (Error Type) 3.11  
**XCL:SYMBOL-NAME-TOO-LONG** (Error Type) 3.11  
**XCL:UNDOABLY** (Macro) A-13  
**XCL:UNDOABLY-SETF** (Macro) A-15

**\**

**\#UNDOSAVES** (Variable) A-15  
**\10MBTYPE.3TO10** (Variable) 4.34  
**\10MBTYPE.PUP** (Variable) 4.34

**~**

**~C** (Format directive) 7.13

**!**

**!EVAL** 2.2

**"**

"numeric" print names 3.43

**\***

**\*break-on-warnings\*** (Variable) E.10  
**\*Clear-Linear-On-Completion\*** (Variable) B.14  
**\*Compile-Fn\*** (Variable) B.16  
**\*COMPILED-EXTENSIONS\*** (Variable) 3.25

**\*DEFAULT-CLEANUP-COMPILER\*** (Variable) 3.25  
**\*DEFAULT-MAKEFILE-ENVIRONMENT\*** (Variable) 3.21  
**\*Edit-Fn\*** (Variable) B.16  
**\*ERROR-OUTPUT\*** (Variable) 3.10  
**\*Fetch-Definition-Error-Break-Flag\*** (Variable) B.16  
**\*Getdef-Error-Fn\*** (Variable) B.16  
**\*Getdef-Fn\*** (Variable) B.16  
**\*LAST-CONDITION\*** (Variable) 3.10  
**\*LISPXPRINT\*** (Property) A-18  
**\*NSADDRESS-FORMAT\*** (Variable) 4.32  
**\*PACKAGE\*** (Variable) 3.20,45-46; A-1  
**\*PRINT-ARRAY\*** (Variable) 3.43  
**\*PRINT-BASE\*** (Variable) 3.39,42,44  
**\*PRINT-BASE\*** vs **RADIX** 3.39  
**\*PRINT-CASE\*** (Variable) 3.44  
**\*PRINT-ESCAPE\*** (Variable) 3.41,44  
**\*PRINT-LENGTH\*** (Variable) 4.22  
**\*PRINT-LEVEL\*** (Variable) 4.22  
**\*PRINT-LEVEL\*** & **\*PRINT-LENGTH\*** vs **PRINTLEVEL** 3.39  
**\*PRINT-LEVEL\*** or **\*PRINT-LENGTH\*** is exceeded 3.45  
**\*PRINT-RADIX\*** (Variable) 3.39,44  
**\*READ-BASE\*** (Variable) 3.20,44  
**\*READ-SUPPRESS\*** (Variable) 3.41  
**\*READTABLE\*** (Variable) 3.39,41-42,48  
**\*READTABLE\*** vs **SETREADTABLE** 3.39  
**\*REMOVE-INTERLISP-COMMENTS\*** (Variable) 3.29-30  
**\*STANDARD-INPUT\*** (Variable) 3.37  
**\*STANDARD-INPUT\*** vs **INPUT** 3.39  
**\*STANDARD-OUTPUT\*** (Variable) 3.37  
**\*STANDARD-OUTPUT\*** vs **OUTPUT** 3.39  
**\*Wrap-Parens\*** (Variable) B.13  
**\*Wrap-Search\*** (Variable) B.14

**.FONT** 4.16

**1**  
 10MB Ethernet encapsulation types 4.34  
 1108 User's Guide  
     summary of changes 6.14  
 1186 User's Guide  
     summary of changes 6.16

**3**  
**3STATE** (FreeMenu Item) 4.26; D.11

:  
**:fast-accessors** (Defstruct option) 7.5  
**:inline** (Defstruct option) 7.5  
**:template** (Defstruct option) 7.5  
**:type** (Defstruct option) 7.5

=  
**=** (Event Address) A-5

?  
 ? (Exec Command) A-7  
 ?? (Exec Command) A-7  
**?ACTIVATEFLG** (Variable) A-24

# INDEX

## A

**Abort** (Editor Command) B-7  
**ACCESS** 3-38  
**Add-Command** (Function) B-14  
**add.process** (Function) 4-12; 7-12  
**ADDMENU** (Function) 4-24  
**ADDTOSCRATCHLIST** (Function) 4-1  
**ADVICE** (File Manager Command) 3-15  
**ADVINFOLST** (Variable) 3-14  
**ADVISE** (File Manager Command) 3-15  
**ADVISE** (Function) 3-13,15  
**ADVISEDFNS** (Variable) 3-14  
**ADVISEDUMP** (Function) 3-14  
Advising 3-14; 7-9  
**AFTERDOMAKESYS** 4-7  
**AFTERDOSAVEVM** 4-7  
**AFTERDOSYSOUT** 4-7  
**AFTERLOGOUT** 4-7  
**AFTERLOGOUTFORMS** 4-7  
**AFTERMAKESYS** 4-7  
**AFTERSAVEVM** 4-7  
**AFTERSYSOUT** 4-7  
**AGAIN** (Editor Command) B-8  
**ALL** (Event Address) A-5  
**ALLOWED-LOGINS** 4-6  
**append** (Function)  
    with non-list argument 7-8  
Application Menus D-1  
**APPLY-format input** A-3  
**ARCHIVEFLG** (Variable) 3-9  
**ARCHIVEFN** (Variable) 3-9  
**Arglist** (Editor Command) B-9  
**AROUNDEXITFNS** (Variable) 4-7  
array reference 7-4  
arrays 3-3  
**ASKUSER** (Function) 4-16  
**assert** (Macro) E-10  
**Attach Menu** (Editor Command) B-11  
Attached Windows 4-28  
**AUTHENTICATE** 4-6  
**AUTHENTICATION.NET.HINT** (Variable) 4-33  
**AUTOHARDRESETFLG** 4-5

## B

back-quote facility 3-49  
**BACKGROUND** (FreeMenu Group Property) D-8  
**BACKGROUND** (FreeMenu Item Property) D-10  
**BACKGROUNDDFNS** (Variable) 4-12  
**BACKSPACE** (Editing Command) A-21  
**BCOMPL** (Function) 3-22,25; 4-10  
**BEEPON** (Function) 4-31  
**BEFORELOGOUT** 4-7  
**BEFOREMAKESYS** 4-7  
**BEFORESAVEVM** 4-7  
**BEFORESYSOUT** 4-7  
**BEFORESYSOUTFORMS** 4-7  
**BITMAP** (FreeMenu System Property) D-10  
**BKSYSBUF** (Function) 4-30  
**BKSYSCHARCODE** (Function) 4-30  
**BLOCKRECORD** (Record Type) 4-3

**BOUNDP** (Function) 3-2  
**BOX** (FreeMenu Group Property) D-5,8  
**BOX** (FreeMenu Item Property) D-10  
**BOXSHADE** (FreeMenu Group Property) D-8  
**BOXSHADE** (FreeMenu Item Property) D-10  
**BOXSPACE** (FreeMenu Group Property) D-8  
**BOXSPACE** (FreeMenu Item Property) D-10  
**break** (Function) 3-13; E-13  
break commands 3-13  
Break packages 3-9  
**BREAK0** (Function) 3-13  
**BREAK1** (Function) 3-9  
**BREAKCONNECTION** (Function) 4-14  
**BREAKIN** (Function) 3-13  
breaking 7-9  
**BREAKREGIONSPEC** (Variable) 4-8  
**BRECOMPILE** (Function) 3-22,25  
**BRKINFOLST** (Variable) 3-13  
**BROKENFNS** (Variable) 3-13  
bulk data transfer 4-34

## C

Catch errors 3-10  
**ccase** (Macro) E-10  
**cerror** (Function) E-9  
**Change Print Base** (Editor Command) B-11  
**CHANGEBACKGROUND** (Function) 4-31  
**CHANGEFONT** (Function) 4-23  
**CHANGESLICE** (Function) A-11,17  
**CHANGESTATE** (FreeMenu Item Property) D-11  
changing a standard readtable 3-22  
characters 3-3  
**CHARCODE** (Function) 3-3  
**CHCON** (Function) 3-42  
**check-type** (Macro) E-10  
CL Exec 3-7  
**CL:\*** (Variable) A-10  
**CL:\*\*** (Variable) A-10  
**CL:\*\*\*** (Variable) A-10  
**CL:+** (Variable) A-10  
**CL:++** (Variable) A-10  
**CL:+++** (Variable) A-10  
**CL:-** (Variable) A-10  
**CL:!** (Variable) A-10  
**CL:!!** (Variable) A-10  
**CL:!!!** (Variable) A-10  
**CL:BREAK** (Function) 3-13  
**CL:CATCH** (Function) 3-5  
**CL:CODE-CHAR** (Function) 3  
**CL:COMPILE-FILE** (Function) 3-24-25; 4-10  
**CL:DEFCONSTANT** (Variable) 3-20  
**CL:DEFINE-MODIFY-MACRO** (Function) 3-20  
**CL:DEFMACRO** (Function) 3-20  
**CL:DEFMACRO** (Macro) 3-29  
**CL:DEFPARAMETER** (Macro) 3-26,29  
**CL:DEFPARAMETER** (Variable) 3-20  
**CL:DEFUN** (Function) 3-20  
**CL:DEFUN** (Macro) 3-29  
**CL:DEFVAR** (Macro) 3-29  
**CL:DEFVAR** (Variable) 3-20

**CL:ERROR** 3-10  
**CL:EVAL-WHEN** (*File Package Command*) 3-31  
**CL:GENSYM** (*Function*) 3-2  
**CL:LOAD** (*Function*) 3-24  
**CL:MAKE-HASH-TABLE** (*Function*) 3-4  
**CL:MAPHASH** (*Function*) 3-4  
**CL:PRIN1** (*Function*) 3-41-42  
**CL:PRINC** (*Function*) 3-41  
**CL:READ** (*Function*) 3-40  
**CL:READ-PRESERVING-WHITESPACE** (*Function*) 3-41  
**CL:THROW** (*Function*) 3-5,11  
**CL:UNWIND-PROTECT** 3-6  
**CL:UNWIND-PROTECT** (*Function*) 3-11  
**CL:WITH-INPUT-FROM-STRING** 3-37  
**CL:WRITE** (*Function*) 3-41  
**CLEANUP** (*Function*) 3-25  
cleanup forms 3-6  
**CLEARCLISPARRAY** (*Function*) 4-10  
**CLEARSTK** (*Function*) 4-5  
**CLEARSTKLST** (*Variable*) 4-5  
CLISP infix forms 3-33  
CLISPARRAY 4-2  
**CLOSEALL** (*Function*) 3-38  
closure 7-8  
**coerce** (*Function*) 7-12  
**COERCE-TO-NSADDRESS** (*Function*) 4-33  
**collect** (*Macro*) 7-6  
collecting objects  
  macros for 7-6  
**COLLECTION** (*FreeMenu Item Property*) D-12  
COLLECTION property 4-26  
**COLUMN** (*FreeMenu Group Property*) D-7  
**COLUMNSPACE** (*FreeMenu Group Property*) D-7  
**Comment Out Selection** (*Editor Command*) B-9  
comment treated as declaration 3-32  
Comments  
  in SEdit B-6  
Common Lisp strings 3-3  
Common Lisp Symbols 3-1  
COMMONNUMSYNTAX 3-44  
**compile-definer** (*Definer*) 7-2  
**compile-form** (*Definer*) 7-2  
compiler  
  behavior with FLEd lexical functions 7-12  
  behavior with recursion 7-12  
  ignoring TEdit formatting 7-12  
  retaining special arguments 7-12  
complex numbers 3-4  
coms 7-11  
condition E-3  
**conditions:\*break-on-signals\*** (*Variable*) E-9  
**conditions:abort** (*Function*) E-21  
**conditions:compute-restarts** (*Function*) E-18  
**conditions:continue** (*Function*) E-21  
**conditions:define-condition** (*Macro*) E-5  
**conditions:find-restart** (*Function*) E-19  
**conditions:handler-bind** (*Macro*) E-4,11  
**conditions:handler-case** (*Macro*) E-11  
**conditions:ignore-errors** (*Macro*) E-12  
**conditions:invoke-debugger** (*Function*) E-13  
**conditions:invoke-restart** (*Function*) E-5,20  
**conditions:invoke-restart-interactively** (*Function*) E-20  
**conditions:make-condition** (*Function*) E-6,8  
**conditions:muffle-warning** (*Function*) E-22  
**conditions:restart-bind** (*Macro*) E-17  
**conditions:restart-case** (*Function*) E-5

**conditions:restart-case** (*Macro*) E-13  
**conditions:restart-name** (*Function*) E-19  
**conditions:signal** (*Function*) E-8  
**conditions:store-value** (*Function*) E-22  
**conditions:use-value** (*Function*) E-22  
**conditions:with-simple-restart** (*Macro*) E-20  
**CONN** (*Exec Command*) A-7  
**CONTROL-A** (*Editing Command*) A-21  
**Control-C** (*Editor Command*) B-7  
**Control-L** (*Editor Command*) B-7  
**Control-Meta-;** (*Editor Command*) B-9  
**Control-Meta-F** (*Editor Command*) B-8  
**Control-Meta-O** (*Editor Command*) B-7  
**Control-P** 4-29  
**CONTROL-Q** (*Editing Command*) A-21  
**CONTROL-R** (*Editing Command*) A-21  
**Control-T** 4-29  
**CONTROL-W** (*Editing Command*) A-21  
**Control-W** (*Editor Command*) B-7  
**CONTROL-X** (*Editing Command*) A-21  
**Control-X** (*Editor Command*) B-7  
**Convert Comments** (*Editor Command*) B-9  
**Convert-Upgrade** (*Variable*) B-14  
converting characters 3-3  
Converting old code  
  for use with new Error system E-1  
**COORDINATES** (*FreeMenu Group Property*) D-7  
**COPY** (*Function*) 3-49  
**COPYBYTES** (*Function*) 4-16  
**COPYDEF** (*Function*) 4-4  
**COPYFILE** (*Function*) 3-38  
**COPYREADTABLE** (*Function*) 3-46  
**COS** (*Function*) 4-3  
**COURIER.CALL** (*Function*) 4-34  
**COURIER.OPEN** (*Function*) 4-34  
Creating an Exec process A-18  
Creating conditions E-4  
Creating icons  
  with ICONW C-1  
CTRLUFLG 4-18  
**ctypescase** (*Macro*) E-10  
CUHOTSPOTX 4-30  
CUHOTSPOTY 4-30  
CUIMAGE 4-30  
current package 3-45  
CURSOR 4-30  
Cursor Movement Commands A-22  
CURSORBITMAP 4-30  
**CURSORCREATE** (*Function*) 4-30  
CURSORHOTSPOTX 4-30  
CURSORHOTSPOTY 4-30

## D

**DA** (*Exec Command*) A-7  
**DAUGHTERS** (*FreeMenu Group Property*) D-8  
**DC** (*Function*) 3-18  
Declining by Condition handler E-4  
DEdit 3-15  
Default handlers 3-10  
**Default-Commands** (*Function*) B-15  
**DEFAULT.OSTYPE** (*Variable*) 4-15  
**DEFAULTFONT** (*Variable*) D-7  
**DEFAULTICONFN** (*Variable*) 4-25  
**DEFAULTTEXTICON** (*Variable*) C-3  
**deferredconstant** (*Function*) 7-12  
**define-file-environment** (*Definer*) 7-2  
**define-record** (*Definer*) 7-3  
Defining New Terms A-11



**DEFMACRO** (Macro) 3-5  
**defstruct** (Macro) 7-4  
     warning 7-6  
**DELDEF** (Function) 3-28  
**Delete Selection** (Editor Command) B-7  
**Delete Structure** (Editor Command) B-8  
**Delete Word** (Editor Command) B-7  
**DELFILE** (Function) 3-38  
**DESELECT** (FreeMenu Item Property) D-12  
**DF** (Function) 3-18  
**DFASL** files 2-1  
**DFNFLG** (Variable) 3-27  
**DIR** (Exec Command) A-7  
**DISPLAY** (FreeMenu Item) D-6-7,14  
 Display icons C-1  
 DISPLAY item 4-26  
**DISPLAYFONTDIRECTORIES** (Variable) 4-23  
**DMACRO** (Property) 3-5  
 DMACROS 2-1  
**DO-EVENTS** (Exec Command) A-8  
**DOCOLLECT** (Function) 4-1  
**DOSHAPFN** (Window Property) 4-25  
**DOWNFN** (FreeMenu Mouse Property) D-10  
**DP** (Function) 3-18  
**DRAWARC** (Function) 4-19  
**DRAWLINE** (Function) 4-19  
**DRAWPOLYGON** (Function) 4-20  
**DSPCLEOL** (Function) 4-18  
**DSPFONT** 4-16  
**DSPRUBOUTCHAR** (Function) 4-18  
**DSPSCALE** 4-19  
 dummy definitions 3-17  
**DV** (Function) 3-18  
**DWIMIFYCOMPFLG** (Variable) 3-34

## E

**ecase** (Macro) E-10  
**ECHOCHAR** (FreeMenu Item Property) D-13  
**ED** (Function) 3-16  
**Edit** (Editor Command) B-9  
**EDIT** (FreeMenu Item) 4-27; D-13  
 Edit caret in SEdit B-2  
 Edit Interface 3-18  
**EDITBM** (Function) 4-18  
**EDITCALLERS** (Function) 3-19  
 Editing Exec Input A-20  
 Editing Lisp Code in Memory B-1  
 Editing VALUES 3-18  
**EDITMODE** (Function) 3-16  
**EDITSTART** (FreeMenu Item) 4-27; D-14  
**END-OF-FILE** (Error Type) 3-12  
**ENDCOLLECT** (Function) 4-1  
 Ending an SEdit session B-2  
**ENDOFSTREAMOP** 3-38  
**ENVAPPLY** 3-6  
**ENVEVAL** 3-6  
**EQUAL** (Function) 3-26  
**EQUALALL** (Function) 4-3  
**ERROR** (Function) 3-10  
**error** (Function) E-9  
 Error conditions 3-10  
 error system 3-10  
 Error system  
     differences between old and new E-1  
 Error system proposal E-1  
 Error type mapping 3-11  
 Error type name 3-11  
 Error type number 3-11

**ERROR!** (Function) 3-10  
**ERRORMESS** (Function) 3-10  
**ERRORMESS1** (Function) 3-10  
**ERRORN** (Function) 2-2; 3-10  
 Errors  
     definition of E-3  
**ERRORSET** 3-10  
**ERRORSTRING** (Function) 3-10  
**ERRORTYPELIST** 3-10  
**ERRORTYPELIST** (Variable) 2-2  
**ERSETQ** (Function) 3-10; 4-8  
**ERXM** 3-10  
**ESCAPE** (Editing Command) A-21  
 Escape  
     in SEdit B-6  
 Establishing handlers within dynamic context E-4  
**etypecase** (Macro) E-10  
**Eval** (Editor Command) B-9  
 EVAL-format input A-2  
 Exec Editing Commands A-22  
 Exec type A-4  
**EXEC-EVAL** (Function) 3-9  
**EXPAND** (Editor Command) B-9  
**EXPANDBITMAP** (Function) 4-18  
**EXPANDMACRO** (Function) 3-5  
**EXPANDREGIONFN** (Window Property) 4-24  
**EXPLICIT** (FreeMenu Group Property) D-7  
**export** (Function) 7-9  
**Extract** (Editor Command) B-9

## F

**F** (Event Address) A-5  
 features  
     new Common Lisp 7-1  
**FETCH** 3-33  
 File Manager 3-19  
 file-reading functions 3-20  
**FILEPKGCOM** (Function) 4-9  
**FILEPKGTYPE** (Function) 4-9  
**FILEPKGTYPES** (Variable) 3-16  
**FILEPOS** (Function) 4-16  
**FILERDTBL** 3-22  
 files containing bitmaps 3-31  
**FILES?** (Function) 3-28  
**FILETYPE** (Property) 3-25  
**FILLPOLYGON** (Function) 4-19-20  
**FIND** (Editor Command) B-8  
**Find Gap** (Editor Command) B-8  
**FIND-READTABLE** (Function) 3-45  
**FINDCALLERS** (Function) 3-19  
**FIX** (Exec Command) A-8  
**FIXP** (Predicate) 3-4  
**flet** (Special form) 7-4  
 floating point 3-4  
**FLOATP** (Predicate) 3-4  
**FM.BACKGROUND** (FreeMenu Window Property) D-15  
**FM.CHANGELABEL** (FreeMenu Function) D-16  
**FM.CHANGELABEL** (Function) 4-27-28  
**FM.CHANGESTATE** (FreeMenu Function) D-16  
**FM.CHANGESTATE** (Function) 4-28  
**FM.DONTRESHAPE** (FreeMenu Window Property) D-15  
**FM.EDITITEM** (FreeMenu Function) D-17  
**FM.EDITP** (FreeMenu Function) D-17  
**FM.ENEDIT** (FreeMenu Function) D-17  
**FM.FIXSHAPE** (Function) 4-28  
**FM.FORMATMENU** (Function) 4-26-27

**FM.GETITEM** (Function) 4-27  
**FM.GETITEM** (FreeMenu Function) D-15  
**FM.GETSTATE** (FreeMenu Function) D-16  
**FM.GETSTATE** (Function) 4-27  
**FM.GROUPPROP** (FreeMenu Macro) D-7,18  
**FM.HIGHLIGHTITEM** (FreeMenu Function) D-17  
**FM.HIGHLIGHTITEM** (Function) 4-28  
**FM.ITEMFROMID** (Function) 4-27  
**FM.ITEMPROP** (FreeMenu Macro) D-18  
**FM.MAKEMENU** (Function) 4-26-27  
**FM.MENUPROP** (FreeMenu Macro) D-7,19  
**FM.NWAYPROP** (FreeMenu Macro) D-19  
**FM.NWAYPROPS** (Macro) 4-27  
**FM.PROMPTWINDOW** (FreeMenu Window Property) D-15  
**FM.READSTATE** (Function) 4-27  
**FM.REDISPLAYITEM** (FreeMenu Function) D-18  
**FM.REDISPLAYMENU** (FreeMenu Function) D-18  
**FM.RESETGROUPS** (FreeMenu Function) D-17  
**FM.RESETMENU** (FreeMenu Function) D-17  
**FM.RESETSHAPE** (FreeMenu Function) D-17  
**FM.RESETSHAPE** (Function) 4-28  
**FM.RESETSTATE** (FreeMenu Function) D-17  
**FM.SHADE** (FreeMenu Function) D-18  
**FM.SHADE** (Function) 4-28  
**FM.SHADEITEM** (Function) 4-28  
**FM.SHADEITEMBM** (Function) 4-28  
**FM.SKIPNEXT** (FreeMenu Function) D-17  
**FM.TOPGROUPID** (FreeMenu Function) D-18  
**FM.WHICHITEM** (FreeMenu Function) D-18  
**FONT** (FreeMenu Group Property) D-7  
**FONT** (FreeMenu Item Property) D-9  
font descriptor 4-22  
**FONTCHANGEFLG** (Variable) 4-23  
**FONTCREATE** (Function) 4-22  
**FONTSAVAILABLE** 4-21  
**FOR** 3-33  
**FOR** (Exec Command) A-6  
**FORGET** 4-6  
**FORGET** (Exec Command) A-8  
**FORMAT** (FreeMenu Group Property) D-4,7  
Free Menu  
    How to make a D-1  
Free Menu format D-2  
Free Menu layout D-1  
**FREEMENU** (FreeMenu Function) D-15  
**FREEMENU** (Function) 4-26-27  
**FROM** (Event Address) A-5  
**FULLNAME** (Function) 3-37  
**FUNARG** 4-4

## G

Gaps  
    in SEdit B-4  
garbage collector 4-11  
**gensym** (Function) 3-2; 7-12  
**GET-ENVIRONMENT-AND-FILEMAP** (Function) 3-23  
**Get-Prompt-Window** (Function) B-15  
**Get-Selection** (Function) B-16  
**Get-Window-Region** (Function) B-13  
**GETDEF** (Function) 3-28  
**GETFILEINFO** (Function) 3-38; 4-13  
**GETPROMPTWINDOW** (Function) 4-28  
**GETREADTABLE** (Function) 3-39  
**GETSYNTAX** 3-45  
global macro shadowing 7-4  
**GROUP** (FreeMenu Group Property) D-7

**GROUPID** (FreeMenu System Property) D-10

## H

**handler** (Function) E-4  
Handling conditions E-3  
**HARDCOPYW** (Function) 4-29  
**HARDRESET** (Function) 4-4  
**HASDEF** (Function) 3-26,28; 4-9  
hash arrays 3-4  
HASHARRAY 3-4  
**HASHARRAY** (Function) 4-2  
**HELDFN** (FreeMenu Mouse Property) D-10  
**HELP** (Editor Command) B-9  
**HELP** (Function) 3-10  
Help Menu Commands B-11  
**HIGHLIGHT** (FreeMenu Item Property) D-9,14  
History list A-16  
**HISTORYSAVEFORMS** (Variable) 3-9  
**HJUSTIFY** (FreeMenu Item Property) D-4,9  
HORRIBLEVARS 4-9,15  
**HPRINT** (Function) 4-15

## I

**ICONW** (Function) C-1  
**ICONW** windows  
    from an image defined by a mask C-1  
    with titles C-1  
**ICONW.SHADE** (Function) C-2  
**ICONW.TITLE** (Function) C-2  
**ID** (FreeMenu Group Property) D-7  
**ID** (FreeMenu Item Property) D-9  
**IDLE-PROFILE** 4-6  
**IDLE-RESETVARS** (Variable) 4-6  
**IDLE-SUSPEND-PROCESS.NAMES** (Variable) 4-7  
IEEE 802-3 specification 4-34  
**IF** 3-33  
**IL Exec** 3-7  
**IL:IT** (Variable) A-9  
**IL:LOAD** (Function) 3-24  
**IL:MAPHASH** (Function) 3-4  
**IL:PRIN1** (Function) 3-41  
**IL:PRIN2** (Function) 3-41  
**IL:READ** (Function) 3-40  
**ILLEGAL-GO** (Error Type) 3-11  
**ILLEGAL-RETURN** (Error Type) 3-11  
**ILLEGAL-STACK-ARG** (Error Type) 3-12  
**IN** (Exec Command) A-6  
**in-package** (Function) 7-8  
**INFILEP** (Function) 3-37  
**INFINITETHICK** (FreeMenu Item Property) D-13  
**INITSTATE** (FreeMenu Item Prop) 4-26  
**INITSTATE** (FreeMenu Item Property) D-9,12  
**INPUT** (Function) 3-37  
**INPUTFONT** (Variable) A-10  
**Inspect** (Editor Command) B-10  
**INTEGERLENGTH** (Function) 4-3  
integers 3-4  
Interlisp Compiler 3-31  
**INTERLISP-ERROR** (Error Type) 3-12  
**INTERPRESSFONTDIRECTORIES** (Variable) 4-22  
**INTERRUPTCHAR** (Function) 4-29  
**INVALID-ARGUMENT-LIST** (Error Type) 3-12  
**ITEMS** (FreeMenu Group Property) D-8

## J

**Join** (Editor Command) B-10

**K****Keep-Window-Region** (*Variable*) B-13**KEYACTION** (*Function*) 4-31**KEYDOWNP** (*Function*) 4-31**L****LABEL** (*FreeMenu Item Property*) D-9

LABELS construct

warning 7-10

**LASTC** (*Function*) 4-15

Layout

of Free Menu D-1

LCOM files 2-1

ldflg 7-11

**LEFT** (*FreeMenu Group Property*) D-7**LEFT and BOTTOM** (*FreeMenu Item Property*) D-9

Left mouse button

in SEdit B-3

lexical bindings 3-33

Library modules

summary of changes 5-1

**LIMITCHARS** (*FreeMenu Item Property*) D-3,13**LINKS** (*FreeMenu Item Property*) D-10,15

LISP 3-47

Lisp structures

SEdit gaps for B-4

**LISPSOURCEFILEP** (*Function*) 4-10**LISPXEVAL** (*Function*) 3-9**LISPFNS** (*Variable*) A-15**LISPXHISTORY** (*Variable*) A-16**LISPXHISTORYMACROS** (*Variable*) 3-9

LISPMACROS 3-8

**LISPMACROS** (*Variable*) 3-9**LISPXREADFN** (*Function*) 4-8**LISPXUNREAD** (*Function*) 3-9**LISPXUSERFN** (*Variable*) 3-9**LIST** (*Function*) 3-49

Lists

in SEdit B-5

**LOAD** (*Function*) 3-20loadflg (*Argument*) 7-11

load-time expression 7-4

**LOADCOMP** (*Function*) 3-25**LOADFNS** (*Function*) 3-20,25**LOADFROM** (*Function*) 3-25

loading compiled files 3-32

loading Medley files into Lyric 4-10

**LOADVARS** (*Function*) 3-25

Locally defined handler E-4

LOCALVARS 3-2

LOGIN.TIMEOUT 4-6

**LOGOUT** (*Function*) 4-7**long-site-name** (*Variable*) 7-3**M****MACHINETYPE** (*Function*) 4-7**MAKE-READER-ENVIRONMENT** (*Function*) 3-23**MAKEFILE** (*Function*) 3-20,25,43,49**MAKEFILE-ENVIRONMENT** (*Property*) 3-21**MAKESYS** (*Function*) 4-7

MAKETITLEBARICON 4-25

**map** (*Function*) 7-11**MAPATOMS** (*Function*) 3-2-3**MAX** (*Function*) 4-2**MAX.INTEGER** (*Variable*) 4-2**MAXHEIGHT** (*FreeMenu Item Property*) D-9**MAXREGION** (*FreeMenu System Property*) D-11**MAXWIDTH** (*FreeMenu Item Property*) D-7,9,13

Medley

on Sun workstations 1-1

on Xerox workstations 1-1

Medley compiled files 2-1

Medley enhancements

summary 1-1

**MENU** (*FreeMenu Group Property*) D-7**MENUFONT** (*FreeMenu Item Property*) D-12**MENUITEMS** (*FreeMenu Item Property*) D-6,12**MENUTITLE** (*FreeMenu Item Property*) D-12**MESSAGE** (*FreeMenu Item Property*) D-9**Meta- (** (*Editor Command*) B-10**Meta- )** (*Editor Command*) B-10**Meta- /** (*Editor Command*) B-9**Meta-9** (*Editor Command*) B-10**Meta-;** (*Editor Command*) B-9**Meta-A** (*Editor Command*) B-7**Meta-B** (*Editor Command*) B-11**Meta-Control-C** (*Editor Command*) B-7**Meta-Control-S** (*Editor Command*) B-8**Meta-Control-X** (*Editor Command*) B-7**Meta-E** (*Editor Command*) B-9**Meta-F** (*Editor Command*) B-8**Meta-H** (*Editor Command*) B-9**Meta-I** (*Editor Command*) B-10**Meta-J** (*Editor Command*) B-10**Meta-M** (*Editor Command*) B-11**Meta-N** (*Editor Command*) B-8**Meta-O** (*Editor Command*) B-9**Meta-P** (*Editor Command*) B-11**Meta-R** (*Editor Command*) B-8**Meta-Return** (*Editor Command*) B-10**Meta-S** (*Editor Command*) B-8**Meta-Space** (*Editor Command*) B-10**Meta-U** (*Editor Command*) B-7**Meta-X** (*Editor Command*) B-9**Meta-Z** (*Editor Command*) B-10

Middle mouse button

in SEdit B-3

**MIN** (*Function*) 4-2**MIN.INTEGER** (*Variable*) 4-2

minimum window size 4-24

**MKSTRING** (*Function*) 3-42**MOMENTARY** (*FreeMenu Item*) D-11**MOTHER** (*FreeMenu Group Property*) D-8

Mouse buttons

in SEdit B-3

**MOVD** (*Function*) 4-4**MOVEDFN** (*FreeMenu Mouse Property*) D-10

multiple escape character 3-42

Multiple Execs A-4

multiple streams 3-37

MULTIPLE-ESCAPE 3-45

**Mutate** (*Editor Command*) B-10**N****NAME** (*Exec Command*) A-8**NCHARS** (*Function*) 3-42

NCHOOSE item 4-26

**NDIR** (*Exec Command*) A-8

Nesting Free Menu Groups D-2

**NETWORKOSTYPES** (*Variable*) 4-15**NEW** (*MAKEFILE Option*) 3-21

NLAMBDA 3-5

**NLSETQ** (*Function*) 3-10; 4-8

NOBIND 3-2

**NOCLEARSTKLST** (*Variable*) 4-5

**NODIRCORE** (*File Device*) 4-13  
**Normalize Selection** (*Editor Command*) B-10  
notational conventions 18  
**NSADDRESS** 4-32  
**NSNAME** 4-32  
**NSNET.DISTANCE** (*Function*) 4-35  
**NUMBER** (*FreeMenu Item*) D-14  
**NUMBERP** (*Predicate*) 3-4  
**NUMBERTYPE** (*FreeMenu Item Property*) D-14  
**NWAY** (*FreeMenu Item*) 4-26; D-6; 12  
**NWAYPROPS** (*FreeMenu Item Prop*) 4-27  
**NWAYPROPS** (*FreeMenu Item Property*) D-6,12

## O

**OLD-INTERLISP-FILE** 3-47  
**OLD-INTERLISP-T** 3-48  
**once-only** (*Macro*) 7-7  
**OPENFILE** (*Function*) 3-37  
**OPENFN** (*Window Property*) 4-25  
**OPENP** (*Function*) 3-38  
**OPENSTREAM** (*Function*) 3-11,37  
**OPENSTRINGSTREAM** (*Function*) 3-37; 4-16  
options E-5  
**ORIG** 3-46  
**OUTPUT** (*Function*) 3-37

## P

package delimiter 2-2  
**PACKAGEDELIM** 3-47  
packages 3-19  
**PARSE-NSADDRESS** (*Function*) 4-33  
**PAT** (*Event Address*) A-5  
pattern matching 3-6  
**PEEK** (*Function*) 4-15  
**pkg-goto** (*Function*) 7-8  
**PL** (*Exec Command*) A-8  
**PLVLFILFLG** 3-42  
**PP** (*Exec Command*) A-9  
**PRETTYDEF** (*Function*) 4-9  
**PRIN1** 4-30  
**PRIN2** 4-30  
**PRINT** (*Function*) 3-20,48  
**PRINTLEVEL** 4-29  
**PRINTNUM** (*Function*) 4-15  
**PRINTOUT** 3-43  
**PRINTOUTFONT** (*Variable*) A-11  
**PRINTSERVICE** (*Variable*) 4-19  
process status window 4-12  
**PROCESS.APPLY** (*Function*) 4-12  
**PROCESS.EVAL** (*Function*) 4-12  
Programmer's interface  
    to SEdit B-12  
**PROMPT#FLG** (*Variable*) 3-9  
**PROMPTFONT** (*Variable*) A-10  
**PROMPTCHARFORMS** (*Variable*) 3-9  
**PROTECTION** 4-13  
**PRXFLG** 3-42  
**PUTDEF** (*Function*) 3-28

## Q

**Quote** (*Editor Command*) B-10  
Quoted structures  
    in SEdit B-5

## R

**RADIX** (*Function*) 3-44  
ratios 3-4

**READ** (*Function*) 3-20,48  
read-eval-print A-1  
read/print consistency 3-44  
**READBUF** (*Variable*) 3-9  
**READC** (*Function*) 3-41  
**READER** 4-13  
**READER-ENVIRONMENT** 3-20  
**READLINE** (*Function*) 4-8  
**READMACROS** 4-16  
**READSYS** (*Function*) 4-35  
**READTABLEPROP** (*Function*) 3-45  
**READVISE** (*Function*) 3-14  
**REALFRAMEP** (*Function*) 4-5  
**REBREAK** (*Function*) 3-14  
**RECOMPILE** (*Function*) 3-22,25  
**record-create** (*Macro*) 7-4  
**record-fetch** (*Macro*) 7-4  
**record-ffetch** (*Macro*) 7-4  
**Redisplay** (*Editor Command*) B-7  
**Redo** (*Editor Command*) B-8  
**REDO** (*Exec Command*) A-6  
**REGION** (*FreeMenu Group Property*) D-8  
**REGION** (*FreeMenu System Property*) D-11  
**RELDRAWTO** (*Function*) 4-19  
Release Notes  
    organization of 17  
**REMEMBER** (*Exec Command*) A-8  
**REMPROP** (*Function*) 3-2  
**RENAMEFILE** (*Function*) 3-38  
**REPAINTFN** 4-24  
**REPAINTFN** (*Window Property*) 4-25  
**REPEATUNTIL** 4-3  
**Replace-Selection** (*Function*) B-16  
Reporting a condition or restart E-5  
**Reset** (*Function*) 3-10; B-14  
**Reset-Commands** (*Function*) B-15  
**RESETFORM** 3-40  
**RESETFORM** 3-39  
**RESETFORMS** (*Variable*) 3-9  
**RESETLST** 3-6  
Resetting system state 3-11  
**RESETVARS** 4-6  
**RESHAPEFN** 4-24  
Restart type E-5  
Restarting computations E-3  
Restarting conditions E-5  
**RETAPPLY** 3-6  
**RETEVAL** 3-6  
**RETFROM** 3-6  
**RETFROM** (*Function*) 3-11  
**RETRY** (*Exec Command*) A-6  
**RETTO** 3-6  
**RETURN** 3-13; 4-5  
**Reverse Find** (*Editor Command*) B-8  
Right mouse button  
    in SEdit B-3  
**ROTATE-BITMAP** (*Function*) 4-18  
**ROW** (*FreeMenu Group Property*) D-7  
**row-major-aref** (*Function*) 7-4  
**ROWSPACE** (*FreeMenu Group Property*) D-7  
RS232 or TTY ports 3-38

## S

**Save-Window-Region** (*Function*) B-13  
**SAVEVM** (*Function*) 4-7  
**SCRATCHLIST** 4-1  
SEdit 3-15  
**SEdit** (*Function*) B-16

SEdit Command Menu B-12  
**SEE** (Exec Command) A-9  
**SEE\*** (Exec Command) A-9  
**SELECTEDFN** (FreeMenu Mouse Property) D-10  
**Set Package** (Editor Command) B-11  
**SETERRORN** (Function) 3-10  
**SETFILEINFO** (Function) 3-38; 4-13  
**SETREADTABLE** (Function) 3-48  
**SETSTKARGNAME** (Function) 4-5  
**SETSYNTAX** 3-45,49  
**SHAPEW** (Function) 4-24  
**SHH** (Exec Command) A-8  
**SHIFT-FIND** (Editor Command) B-8  
**short-site-name** (Variable) 7-3  
**SHOULDCOMPILEMACROATOMS** (Variable) 4-4  
**SHOULDNT** (Function) 3-10  
**SHOWPARENFLG** (Variable) A-25  
**SHRINKBITMAP** (Function) 4-18  
**SHRINKFN** (Window Property) 4-24  
 SIDE effects of event A-18  
 Signalling conditions E-3  
**SIN** (Function) 4-3  
 Sketch  
   summary of changes 6-10  
**SKIP-NEXT** (Editor Command) B-8  
**SKREAD** (Function) 3-41  
**SORT** (Function) 4-1  
 Special characters  
   in SEdit B-5  
 Specifying event addresses A-4  
 Specifying Free Menu Items D-2  
 stack manipulations 3-5  
**STACK OVERFLOW** (Error Type) 4-4  
 Stack pointers 3-5  
**STACK-OVERFLOW** (Error Type) 3-11  
**STACK-POINTER-RELEASED** (Error Type) 3-12  
 Starting an SEdit session B-2  
**STATE** 4-26  
**STATE** (FreeMenu Item) D-7,11  
**STATE** (FreeMenu Item Property) D-12  
**STATE** (FreeMenu System Property) D-10  
**STKARG** (Function) 4-5  
**STKNARGS** (Function) 4-5  
**STKPOS** (Function) 4-5  
**STOP** (Function) 4-10  
**STOP-UNDOABLY** (Macro) A-13  
 strings 3-3  
   in SEdit B-6  
**STRINGWIDTH** (Function) 3-42; 4-22  
 Structure caret in SEdit B-2  
 Structure editor 3-15  
**Substitute** (Editor Command) B-8  
**SUCHTHAT** (Event Address) A-5  
**SUSPEND-PROCESS.NAMES** 4-7  
 Switching between editors 3-16  
 Symbols 3-1,6  
   in Error system E-1  
 symbols in the INTERLISP package 3-20  
**SYSDOWNFN** (FreeMenu System Property) D-11  
 sysload 3-24; 7-11  
**SYSMOVEDFN** (FreeMenu System Property) D-11  
**SYSOUT** (Function) 4-7  
**SYSPRETTYFLG** (Variable) 3-9  
**SYSSELECTEDFN** (FreeMenu System Property) D-11

## T

**TABLE** (FreeMenu Group Property) D-7

**TCOMPL** (Function) 3-22,25; 4-10  
 TEdit  
   summary of changes 6-1  
 TeleRaid Library module 4-35  
**TEXTICON** (Function) 4-25; C-3  
**TIME** (Exec Command) A-9  
**TIME** (Function) 3-36  
**TIME** (Macro) 3-36  
**TITLE** (FreeMenu Item) 4-27  
 titled icons 4-25  
**TILEDICONW** (Function) C-1  
**TOGGLE** (FreeMenu Item) D-11  
**TOO-MANY-ARGUMENTS** (Error Type) 3-12  
**TRACE** (Function) 3-13-14  
**TTYBACKGROUNDFN** (Variable) 4-12  
**TTYDISPLAYSTREAM** (Function) 4-25  
 TTYIN display typein editor 4-16  
 TTYIN Editor from Exec A-20  
**TY** (Exec Command) A-9  
**TYPE** (Exec Command) A-9  
**TYPE** (FreeMenu Item Property) D-9

## U

**UGLYVARS** 3-43; 4-9,15  
**UNBOUND-VARIABLE** (Error Type) 3-12  
**UNBREAK** (Function) 3-14  
**UNBREAKIN** (Function) 3-13  
**UNDEFINED-CAR-OF-FORM** (Error Type) 3-12  
**UNDEFINED-FUNCTION-IN-APPLY** (Error Type) 3-12  
**UNDO** (Editor Command) B-7  
**UNDO** (Exec Command) A-4,8,13  
**UNDO key** (Editing Command) A-21  
**UNDOABLY-MAKUNBOUND** (Function) 3-29  
**UNDOABLY-SETQ** (Function) A-15  
 Undoing in Functions A-14  
 Undoing In Programs A-13  
 Undoing out of order A-16  
**UNDOSAVE** (Function) A-15  
**UNIXFTPFLG** (Variable) 4-14  
**UNPACKFILENAME** (Function) 3-37  
**UNSAFEMACROATOMS** (Variable) 4-4  
 UNTIL 4-3  
**USE** (Exec Command) A-6  
**USERDATA** (FreeMenu System Property) D-11  
 USERDATA LIST D-14  
**USEREXEC** (Function) 3-9  
 USERNAME 4-6  
**USERWORDS** (Variable) A-25  
**USESILPACKAGE** 3-45  
 Using Execs 3-7

## V

**VALUEFONT** (Variable) A-11  
**VARS** 4-15  
 version delimiter 2-2  
**VIDEORATE** (Function) 4-31  
**VJUSTIFY** (FreeMenu Item Property) D-9

## W

**warn** (Function) E-10  
**WHENCHANGED** 4-9  
**WINDOWPROP** (Function) 4-26  
**WINDOWPROPS** 4-26  
**with-collection** (Macro) 7-6  
**with-input-from-string** (Macro) 7-13  
**with-output-to-string** (Macro) 7-13

**WITH-READER-ENVIRONMENT** (Macro) 3-23  
**write-string** (Function) 7-12  
**WRITESTRIKEFONTFILE** (Function) 4-22  
 writing macros  
   macros for 7-7  
 Writing your own SEdit commands B-14

**X**

XCL 3-47  
 XCL Compiler 3-31  
 XCL Exec 3-7  
 XCL readtable 3-21  
**xcl:\*current-condition\*** (Variable) E-8  
**XCL:\*DEBUGGER-PROMPT\*** (Variable) A-19  
**XCL:\*EVAL-FUNCTION\*** (Variable) A-19  
**XCL:\*EXEC-PROMPT\*** (Variable) A-19  
**XCL:\*PER-EXEC-VARIABLES\*** (Variable) A-18  
**XCL:ABORT** (Function) 3-10  
**XCL:ADD-EXEC** (Function) A-18  
**XCL:ARGLIST** (Variable) 3-15  
**XCL:ARRAY-SPACE-FULL** (Error Type) 3-12  
**XCL:ATTEMPT-TO-CHANGE-CONSTANT** (Error Type) 3-11-12  
**XCL:ATTEMPT-TO-RPLAC-NIL** (Error Type) 3-11  
**XCL:CATCH-ABORT** 3-10  
**xcl:catch-abort** (Macro) E-21  
**XCL:CONDITION** 3-10  
**xcl:condition-case** (Macro) E-11  
**xcl:condition-handler** (Macro) E-8  
**xcl:condition-reporter** (Macro) E-7  
**XCL:CONTROL-E-INTERRUPT** (Error Type) 3-12  
**XCL:DATA-TYPES-EXHAUSTED** (Error Type) 3-12  
**XCL:DEF-DEFINE-TYPE** (Macro) 3-27-28  
**XCL:DEFCOMMAND** 3-8  
**XCL:DEFCOMMAND** (Macro) A-11  
**XCL:DEFDEFINER** (Function) 3-20  
**XCL:DEFDEFINER** (Macro) 3-29  
**XCL:DEFGLOBALPARAMETER** (Variable) 3-20  
**XCL:DEFGLOBALVAR** (Variable) 3-20  
**XCL:DEFINE-PROCEED-FUNCTION** (Function) 3-20  
**XCL:DEFININE** (Function) 3-20  
**XCL:DEFOPTIMIZER** 3-32  
**XCL:DEFOPTIMIZER** (Macro) 3-5  
**XCL:EXEC** (Function) A-18  
**XCL:EXEC-EVAL** (Function) A-19  
**XCL:EXEC-FORMAT** (Function) A-19  
**XCL:FILE-NOT-FOUND** (Error Type) 3-12  
**XCL:FILE-WONT-OPEN** (Error Type) 3-11  
**XCL:FLOATING-OVERFLOW** (Error Type) 3-12  
**XCL:FLOATING-UNDERFLOW** (Error Type) 3-12  
**XCL:FS-PROTECTION-VIOLATION** (Error Type) 3-12  
**XCL:FS-RESOURCES-EXCEEDED** (Error Type) 3-12  
**XCL:HASH-TABLE-FULL** (Error Type) 3-12  
**XCL:INVALID-PATHNAME** (Error Type) 3-12  
**XCL:SET-DEFAULT-EXEC-TYPE** (Function) A-20  
**XCL:SET-EXEC-TYPE** (Function) A-20  
**XCL:SIMPLE-DEVICE-ERROR** (Error Type) 3-11  
**XCL:SIMPLE-TYPE-ERROR** (Error Type) 3-11  
**XCL:STORAGE-EXHAUSTED** (Error Type) 3-12  
**XCL:STREAM-NOT-OPEN** (Error Type) 3-11  
**XCL:SYMBOL-HT-FULL** (Error Type) 3-11  
**XCL:SYMBOL-NAME-TOO-LONG** (Error Type) 3-11  
**XCL:UNDOABLY** (Macro) A-13

**XCL:UNDOABLY-SETF** (Macro) A-15

**1**

10MB Ethernet encapsulation types 4-34  
 1108 User's Guide  
   summary of changes 6-14  
 1186 User's Guide  
   summary of changes 6-16

**3**

**3STATE** (FreeMenu Item) 4-26; D-11

**\**

**\#UNDOSAVES** (Variable) A-15  
**\10MBTYPE-3TO10** (Variable) 4-34  
**\10MBTYPE-PUP** (Variable) 4-34

**~**

**~C** (Format directive) 7-13

**!**

**!EVAL** 2-2

**\***

**\*break-on-warnings\*** (Variable) E-10  
**\*Clear-Linear-On-Completion\*** (Variable) B-14  
**\*Compile-Fn\*** (Variable) B-16  
**\*COMPILED-EXTENSIONS\*** (Variable) 3-25  
**\*DEFAULT-CLEANUP-COMPILER\*** (Variable) 3-25  
**\*DEFAULT-MAKEFILE-ENVIRONMENT\*** (Variable) 3-21  
**\*Edit-Fn\*** (Variable) B-16  
**\*ERROR-OUTPUT\*** (Variable) 3-10  
**\*Fetch-Definition-Error-Break-Flag\*** (Variable) B-16  
**\*Getdef-Error-Fn\*** (Variable) B-16  
**\*Getdef-Fn\*** (Variable) B-16  
**\*LAST-CONDITION\*** (Variable) 3-10  
**\*LISPXPRT\*** (Property) A-18  
**\*NSADDRESS-FORMAT\*** (Variable) 4-32  
**\*PACKAGE\*** (Variable) 3-20,45-46; A-1  
**\*PRINT-ARRAY\*** (Variable) 3-43  
**\*PRINT-BASE\*** (Variable) 3-39,42,44  
**\*PRINT-BASE\*** vs RADIX 3-39  
**\*PRINT-CASE\*** (Variable) 3-44  
**\*PRINT-ESCAPE\*** (Variable) 3-41,44  
**\*PRINT-LENGTH\*** (Variable) 4-22  
**\*PRINT-LEVEL\*** (Variable) 4-22  
**\*PRINT-LEVEL\*** & **\*PRINT-LENGTH\*** vs PRINTLEVEL 3-39  
**\*PRINT-LEVEL\*** or **\*PRINT-LENGTH\*** is exceeded 3-45  
**\*PRINT-RADIX\*** (Variable) 3-39,44  
**\*READ-BASE\*** (Variable) 3-20,44  
**\*READ-SUPPRESS\*** (Variable) 3-41  
**\*READTABLE\*** (Variable) 3-39,41-42,48  
**\*READTABLE\*** vs SETREADTABLE 3-39  
**\*REMOVE-INTERLISP-COMMENTS\*** (Variable) 3-29-30  
**\*STANDARD-INPUT\*** (Variable) 3-37  
**\*STANDARD-INPUT\*** vs INPUT 3-39  
**\*STANDARD-OUTPUT\*** (Variable) 3-37  
**\*STANDARD-OUTPUT\*** vs OUTPUT 3-39  
**\*Wrap-Parens\*** (Variable) B-13  
**\*Wrap-Search\*** (Variable) B-14

---

:  
:fast-accessors (Defstruct option) 7-5  
:inline (Defstruct option) 7-5  
:template (Defstruct option) 7-5  
:type (Defstruct option) 7-5

=  
= (Event Address) A-5

?  
? (Exec Command) A-7  
?? (Exec Command) A-7  
?ACTIVATEFLG (Variable) A-24

# LIST OF TABLES

---

| Table                                                  | Page |
|--------------------------------------------------------|------|
| 1. TEdit's Abbreviations and their Expanded Characters | 6-1  |



[This page intentionally left blank]

# PREFACE

---

The *Lisp Release Notes* provide current information about the Lisp software development environment. You will find the following information in these *Notes*:

- An overview of significant extensions to the Common Lisp language.
- Descriptions of new features that enhance the integration and implementation of Common Lisp into the Lisp environment.
- A summary of changes made in the Library modules, in the Sketch and TEdit tools, and in the 1108 and 1186 User's Guides.
- Discussions of how specific Common Lisp features have affected the Interlisp-D language.
- Notes reflecting the changes made to Interlisp-D, independent of Common Lisp.
- Known restrictions.

---

## How the Release Notes are Organized

---

The *Lisp Release Notes* are organized as follows:

Chapter 1, Introduction, summarizes the Medley release enhancements.

Chapter 2, Notes and Cautions, highlights significant Medley and Lyric changes in the Lisp environment.

Chapter 3, Common Lisp/Interlisp-D Integration, discusses how the integration of Common Lisp into the Lisp environment affects Interlisp features.

Chapter 4, Changes to Interlisp-D in Lyric/Medley, outlines changes that have taken place in Interlisp-D and its environment during the Lyric and Medley releases. These changes are primarily independent of Common Lisp integration.

Chapters 3 and 4 are organized to parallel the *Interlisp-D Reference Manual* as closely as possible. To make it easy to use these chapters with the *IRM*, the following conventions are used:

Information is organized by *Interlisp-D Reference Manual* volume and section. The *IRM* section level headings are maintained to aid in cross-referencing.

Chapter 5, Library Modules, is a synopsis of the changes to the Lisp Library Modules.

Chapter 6, *User's Guides*, is a collection of release notes on the 1108 and 1186 User's Guides; *A User's Guide to Sketch*, and *A User's Guide to TEdit*.

Chapter 7, *Common Lisp Implementation*, describes improved features that integrate Common Lisp into the environment.

Five Appendices contain documentation of newly integrated system features:

Appendix A, *The Exec*, describes Lisp's Exec.

Appendix B, *SEdit*, describes the Lisp structure editor.

Appendix C, *ICONW*, describes the Lisp feature for building display icons.

Appendix D, *Free Menu*, describes Lisp's flexible menu feature.

Appendix E, *Error System*, describes error conditions and recovery.

---

## Notational Conventions

---

Conventions used in the *Lisp Release Notes* include the following:

Names of Interlisp functions, macros and variables are shown in **BOLD UPPERCASE**; their arguments are in *ITALICS*.

Names of Common Lisp functions, macros and variables are shown in **bold lowercase**; their arguments are in *italics*.

A backslash (\) character preceding a function or variable name signifies that it is a property of the system.

Examples are shown in `terminal 10`.

Text shown with ~~StrikeThru~~ is information that no longer applies.

Text shown with revision bars in the right margin is information that has been added or modified since the last release.

References to the *Interlisp-D Reference Manual*, or *IRM*, are used throughout this manual.

---

## How to Use The Release Notes

---

The *Lisp Release Notes* contain current information on the Lisp environment. The Medley release enhances the Lyric release with new features and corrections to over 450 known Lyric bugs. Because Medley primarily contains additions to Lyric, these *Release Notes* have been written to include Lyric information that applies in Medley.

These *Lisp Release Notes* replace the *Lyric Release Notes*. The descriptions contained within these *Notes* are closely interwoven with functions, variables and other concepts discussed in the *Interlisp-D Reference Manual*, or *IRM*. Chapters 3 and 4 of these *Notes* closely parallel the *IRM*, preserving section headings with respect to order and numbering. You might find it helpful to go through the *Release Notes* and the *IRM* together, marking the *IRM* sections that have new information. Later when you consult the *IRM* you will know which sections require you to read the analogous section of the *Release Notes*.

---

## Related Literature

---

We recommend that you use the *Lisp Release Notes* as a supplement to the following publications:

*Interlisp-D Reference Manual*, Volumes I through III, Koto Release, 1985.

*Common Lisp, the Language*, by Guy L. Steele Jr., Digital Press, 1984.

*Common Lisp Implementation Notes*, Lyric Release, 1987.

*Lisp Documentation Tools*, (includes "A User's Guide to TEdit" and "A User's Guide to Sketch"), Lyric Release, 1987.

*Lisp Library Modules*, Medley Release, 1988.

*Medley 1.0-S User's Guide*, Medley Release, 1988.

[This page intentionally left blank]

**PRINTING SPECIFICATIONS**  
**LISP RELEASE NOTES, MEDLEY RELEASE**  
**AND**  
**LISP LIBRARY MODULES, MEDLEY RELEASE**

***Special Instructions:*** >>3-hole punch (error tolerances: + or - 1/16 inch)<<

**PRINTING**

***Printing Method:*** >>offset<<

***Paper Weight:*** >>60 lb. <<

***Paper Type (Finish):*** >> Matte<<

***Paper Color:*** >>White<<

***Paper Texture:*** >>Smooth<<

***Paper Opacity:*** 92 (no show through)

***Paper Size:*** >>8-1/2 X 11<<

***Exceptions (e.g., oversize diagrams):*** >>none<<

***Special Instructions:*** >><<

***Number of Pages:*** >><<

**PAPER**

The *Lisp Release Notes* contain information from both the Lyric and Medley releases, including descriptions of all Lyric bug fixes. Medley additions are indicated with revision bars in the right margin.

---

## Summary of Medley Changes

---

The Medley release is currently provided on two platforms, Xerox 1100 series workstations using Medley 1.0, and Sun 3 workstations using Medley 1.0-S. Medley 1.0 and Medley 1.0-S are compatible with each other and will let you develop software on either platform. Source and compiled files are transferable between the two platforms. Sysouts developed on Xerox workstations can also be run on the Sun 3. Sysouts made on the Sun 3, however, cannot be run on Xerox workstations.

The Medley release enhances the Lyric release and fixes over 450 known Lyric bugs. Medley adds new features, improves Common Lisp implementation, and improves overall reliability of the Lisp sysout. Specific enhancements include:

- The COMPILER contains many added optimizations and numerous bug fixes.
- The DEBUGGER evaluates lexical variables. Lexical variables can now be contained in interpreted code.
- DFASL files now behave at the level of Interlisp-D compiled files. COMS are contained in DFASLs so that the system loads a DFASL file only once.
- The SEDIT code editor is more robust and better integrated with the environment.
- Common Lisp comments are preserved during loading. During MAKEFILE, comments can be written out with just semicolons.
- The new ERROR SYSTEM is compatible with the most recent standard defined for Common Lisp error systems.
- TEdit contains numerous bug fixes.
- MASTERSCOPE contains Common Lisp query support allowing you to ask questions about Common Lisp code that could previously be asked only of Interlisp-D code. Currently, questions specific to Common Lisp constructs are not supported.
- RS232 contains many bug fixes that improve the reliability of data transfer and the addition of various debugging tools.
- TCP/IP now contains many bug fixes including UNIX file interface and directory enumeration.

- A new System Tool lets you fetch sysouts from TCP hosts.
- NS Random Access is now supported.
- A new File Browser user interface now supports file sorting by dates. The new interface includes the ability to stop in the middle of operations.
- The Medley sysout is about the same size as the Lyric sysout.

In addition, Medley on the Sun 3 workstation offers the following new features:

- The UnixChat library module allows you to communicate with a UNIX shell on your own host.
- The UnixComm library module allows you to start a Unix process on a Sun workstation and provides an interface to the SunOS operating system.
- The ability to suspend Medley and use UNIX as a background process is provided.



[This page intentionally left blank]

## 2. NOTES AND CAUTIONS

---

This section contains notes and cautions that apply in Lyric and Medley. Medley notes are indicated with revision bars in the right margin. Text shown with ~~StrikeThru~~ is that information from the Lyric release that no longer applies.

---

### Changes and Cautions in the Medley Release

---

- The Medley Release is currently provided on two platforms, the Xerox 1100 series workstations and selected Sun workstations. File structure for the 1108/09/86 remains the same. For Sun workstations, UNIX file structure is supported. See the *Medley 1.0-S User's Guide* for details.
- Files compiled in Medley cannot be loaded back into Lyric. Medley-compiled .LCOM and .DFASL files will produce an error message when loaded into Lyric. (Lyric-compiled .LCOM and .DFASL files can be loaded and run in Medley.) If you need to run a Medley file in Lyric, load the source file and use the Lyric compiler.
- SEdit and definers now support four-semicolon and balanced comments. Print support for these new types of comments is also provided. For details, see "TextModules" in the *Lisp Library Modules* manual, and "SEdit" in Appendix B of this manual.
- Medley and Lyric can both be installed on one machine.

---

### Changes and Cautions in the Lyric Release

---

- Koto and Lyric cannot both be supported on one machine.
- You must have Services 10.0 installed on your printers to correctly print TEdit files.
- Interlisp DMACROS are not visible to Common Lisp. If a symbol has both a function definition and a DMACRO property, the compiler assumes that the DMACRO is an optimizer for the old Interlisp compiler and ignores it.
- The Common Lisp functions found in *Common Lisp, The Language*, section 25.4.2, "Other environmental inquiries" (e.g., LISP-IMPLEMENTATION-TYPE) are in the COMMON LISP (CL:) package.
- Both Medley and Lyric use the new type of Executive, and both sysouts contain the ability to spawn multiple executive processes. The default executive is Common Lisp, not

Interlisp. The old Executive (the "Programmer's Assistant") is not available in Medley.

You should be particularly careful in the new Executives when typing file names, as some file name delimiters now have syntactic significance in the new readtables. In particular, the character colon (:) used in NS file server names is a package delimiter in all new Executives, and the version delimiter semi-colon (;) is a comment character in the Common Lisp Executives. If you type a file name in the form of a symbol to an Exec, you must escape the special characters, or use the multiple escape character around the whole name. For example, in a Common Lisp Exec you might type

```
{FS\:Me\:Company}<Fred>Stuff.tedit\;3
```

or

```
|{FS:Me:Company}<Fred>Stuff.tedit;3|,
```

which are equivalent, except that the former is read as all upper case (Common Lisp Exec's read case-insensitively). This caution should also be noted when copy-selecting file names out of a File Browser.

It is recommended that you type file names as strings whenever possible, as virtually all system interfaces accept strings instead of symbols. Two notable exceptions are MAKEFILE and TEDIT, which require symbols when naming files.

These escaping rules apply *only* to file names typed to an Executive (or in general, a Lisp reader). Individual tools that prompt for a file name read the name as a string, so escape characters need not (and should not) be typed. In particular, this is true for the prompt windows of TEdit and File Browser, and the prompt for an Init file when a system with no local Init file is started up.

- A new error system, based on the current Common Lisp proposed error standard, replaces the old Interlisp error system.
- The **!EVAL** debugger command no longer exists and the = and -> break commands are no longer supported..
- The function **ERRORN** no longer exists and **ERRORTYPELIST** is no longer supported. See Chapter 3, Common Lisp /Interlisp Integration, section 14.10 "Error List" for Interlisp errors that are no longer supported.
- The Lyric release contained a new compiler and compiled code format, .DFASL (FASt Loading) files. The old compiler is still available and produces files in the old format, but with extension .LCOM. The old compiler will not be available in future releases.
- Files produced by the Lyric File Manager cannot be loaded into previous releases of the system. Files compiled in Koto cannot be loaded into Lyric.

- SETQ from the exec does not interact with the File Manager, nor does it print (var reset) (except in the "Programmer's Assistant").
- DWIM/CLISP: CLISP infix is no longer fully supported; users should dwimify old Koto code before running it in Lyric. Additionally, WITH constructs using "←" and BIND constructs in the form of an atom A←B need to be dwimified.
- The functions BREAKDOWN and BRKDOWNRESULTS as well as the variables, BRKDWNTYPE and BRKDWNTYPES have been removed from the environment. The Lisp Library Module, SPY supersedes BREAKDOWN.
- The file system supports having multiple streams opened on a single file at one time. This means that the input/output functions accept only streams as arguments, not symbols naming files. This has several implications for Interlisp programmers, one being that the function **CLOSEALL** is no longer implemented. See the Chapter 3, Common Lisp/Interlisp Integration, Streams and Files section, for details.
- Windows cannot be used interchangeably with streams in Common Lisp functions. If you need to use a window in the middle of a Common Lisp function, use (IL:GETSTREAM window) to get the associated display stream.
- Loading CPM-format floppies is very slow in Lyric. CPM-format floppies are not supported in Medley.
- The default Interlisp readtable has been modified for compatibility with Common Lisp. The characters colon (:), hash (#) and vertical bar (|) have different meaning. The File Manager gives a choice of reader environments in which to write files, and remembers which one was used for each file.
- READ/PRINT consistency: Old Interlisp code that used **READ** and **PRINT** without being careful about using a particular readtable may need to be fixed.
- The Interlisp function **SKREAD** defaults its readtable argument to the current readtable, viz., the value of **\*READTABLE\***, rather than **FILERDTBL**.
- FREEMENU and ICONW, formerly Library modules, are included in the Lisp.sysout in Lyric and Medley.
- The Lyric Lisp editor, SEdit, has been modified in Medley. DEdit is now a library module.
- Revised fonts: Lyric revised the naming convention for font files, and printer width files had corrected line leading information. ~~Old Koto fonts can still be used, but you are encouraged to start using the new fonts as soon as practicable.~~ Medley and Lyric fonts are completely compatible.
- Lyric image objects are now stored on files in a way that cannot always be read into Koto. (Lyric, on the other hand, can read Koto image objects.) This means, for example, that you may not be able to share TEdit files or sketches containing image objects between Koto and Lyric.

- The field names for the CURSOR datatype have been changed.
- Masterscope has been removed from the standard environment. If you wish to use it, load the Masterscope Library module.
- Pattern matching is no longer a part of the standard environment. Pattern matching can be found in the Lisp Library Module, Match.
- PRESS fonts are not part of the standard Lisp environment. PRESS is now available as a Library Module.
- In Lyric, the Library module TCP/IP does not work on 1186 workstations that have *both* IOPs with part number 140K03030 *and* "old" ROMs. The problem is not with the IOP board per se, rather it's a problem with the IOP's ROMs. If TCP/IP doesn't work on your 1186 you should check your IOP board revision. If you have the old IOP you may need to replace the ROMs before you can use TCP/IP, contact your service representative.

TCP/IP does work with newer IOPs—part number 140K05560.

If you attempt to Teleraid a Lyric sysout from a Koto sysout, you should be aware of the following:

1. All symbols will be read as if they were in the INTERLISP package and you can only type a subset of the IL symbols to it.
2. Teleraid will not understand certain Common Lisp datatypes, such as CHARACTER and strings.

[This page intentionally blank]

# 3. COMMON LISP/INTERLISP-D INTEGRATION

---

**NOTE:** Chapter 3 is organized to correspond to the original *Interlisp-D Reference Manual*, and explains changes related to how Common Lisp affects Interlisp-D in your Lisp software development environment. To make it easy to use this chapter with the *IRM*, information is organized by *IRM* volume and section numbers. Section headings from the *IRM* are maintained to aid in cross-referencing.

Lyric information as well as Medley release enhancements are included. Medley additions are indicated with revision bars in the right margin.

## VOLUME I—LANGUAGE

---

### Chapter 2 Litatoms

---

#### (2.1)

What Interlisp calls a "LITATOM" is the same as what Common Lisp calls a "SYMBOL." Symbols are partitioned into separate name spaces called packages. When you type a string of characters, the resulting symbol is searched for in the "current package." A colon in the symbol separates a package name from a symbol name; for example, the string of characters "CL:AREF" denotes the symbol AREF accessible in the package CL. For a full discussion, see Guy Steele's *Common Lisp, the Language*.

All the functions in this section that create symbols do so in the INTERLISP package (IL), which is also where all the symbols in the *Interlisp-D Reference Manual* are found. Note that this is true even in cases where you might not expect it. For example, U-CASE returns a symbol in the INTERLISP package, even when its argument is in some other package; similarly with L-CASE and SUBATOM. In most cases, this is the right thing for an Interlisp program; e.g., U-CASE in some sense returns a "canonical" symbol that one might pass to a SELECTQ, regardless of which executive it was typed in. However, to perform symbol manipulations that preserve package information, you should use the appropriate Common Lisp functions (See *Common Lisp the Language*, Chapter 11, Packages and Chapter 18, Strings).

Symbols read under an old Interlisp readtable are also searched for in the INTERLISP package. See Section 25.8, Readtables, for more details.

## Section 2.1 Using Litatoms as Variables

---

(I:2.3)

(**BOUND** *VAR*)

[Function]

The Interlisp interpreter has been modified to consider any symbol bound to the distinguished symbol **NOBIND** to be unbound. It will signal an UNBOUND-VARIABLE condition on encountering references to such symbols. In prior releases, the interpreter only considered a symbol unbound if it had no dynamic binding and in addition its top-level value was **NOBIND**.

For most user code, this change has no effect, as it is unusual to bind a variable to the particular value **NOBIND** and still deliberately want the variable to be considered bound. However, it is a particular problem when an interpreted Interlisp function is passed to the function **MAPATOMS**. Since **NOBIND** is a symbol, it will eventually be passed as an argument to the interpreted function. The first reference to that argument within the function will signal an error.

A work-around for this problem is to use a Common Lisp function instead. Calls to this function will invoke the Common Lisp interpreter which will treat the argument as a local, not special, variable. Thus, no error will be signaled. Alternatively, one could include the argument to the Interlisp function in a **LOCALVARS** declaration and then compile the function before passing it to **MAPATOMS**. This has the advantage of significantly speeding up the **MAPATOMS** call.

## Section 2.3 Property Lists

---

(I:2.6)

The value returned from the function **REMPROP** has been changed in one case:

(**REMPROP** *ATM PROP*)

[Function]

Removes all occurrences of the property *PROP* (and its value) from the property list of *ATM*. Returns *PROP* if any were found (**T** if *PROP* is **NIL**), otherwise **NIL**.

## Section 2.4 Print Names

---

(I:2.7)

The print functions now qualify the name of a symbol with a package prefix if the symbol is not accessible in the current package. The Interlisp "PRIN1" print name of a symbol does not include the package name.

(I:2.10)

The **GENSYM** function in Interlisp creates symbols interned in the INTERLISP package. The Common Lisp **CL:GENSYM** function creates uninterned symbols.



**(l:2.11)****(MAPATOMS FN)****[Function]**See the note for **BOUNDP** above.

---

## Section 2.5 Characters

---

A "character" in Interlisp is different from the type "character" in Common Lisp. In Common Lisp, "character" is a distinguished data type satisfying the predicate **CL:CHARACTERP**. In Interlisp, a "character" is a single-character symbol, not distinguishable from the type symbol (litatom). Interlisp also uses a more efficient object termed "character code", which is indistinguishable from the type integer.

Interlisp functions that take as an argument a "character" or "character code" do not in general accept Common Lisp characters. Similarly, an Interlisp "character" or "character code" is not acceptable to a Common Lisp function that operates on characters. However, since Common Lisp characters are a distinguished datatype, Interlisp string-manipulation functions are willing to accept them any place that a "string or symbol" is acceptable; the character object is treated as a single-character string.

To convert an Interlisp character code *n* to a Common Lisp character, evaluate (**CL:CODE-CHAR** *n*). To convert a Common Lisp character to an Interlisp character code, evaluate (**CL:CHAR-CODE** *n*). For character literals, where in Interlisp one would write (**CHARCODE** *x*), to get the equivalent Common Lisp character one writes *#x*. In this syntax, *x* can be any character or string acceptable to **CHARCODE**; e.g., *#\GREEK-A*.

---

## Chapter 4 Strings

---

**(l:4.1)**

Interlisp strings are a subtype of Common Lisp strings. The functions in this chapter accept Common Lisp strings, and produce strings that can be passed to Common Lisp string manipulation functions.

---

## Chapter 5 Arrays

---

Interlisp arrays and Common Lisp arrays are disjoint data types. Interlisp arrays are not acceptable arguments to Common Lisp array functions, and vice versa. There are no functions that convert between the two kinds of arrays.

## Chapter 6 Hash Arrays

---

Interlisp hash arrays and Common Lisp hash tables are the same data type, so Interlisp and Common Lisp hash array functions may be freely intermixed. However, some of the arguments are different; e.g., the order of arguments to the map functions in **IL:MAPHASH** and **CL:MAPHASH** differ. The extra functionality of specifying your own hashing function is available only from Interlisp **HASHARRAY**, not **CL:MAKE-HASH-TABLE**, though the latter does supply the three built-in types specified by *Common Lisp, the Language*.

## Chapter 7 Numbers and Arithmetic Functions

---

### (I:7.2)

The addition of Common Lisp data structures within the Lisp environment means that there are some invariants which used to be true for anything in the environment that are no longer true.

For example, in Interlisp, there were two kinds of numbers: integer and floating. With Common Lisp, there are additional kinds of numbers, namely ratios and complex numbers, both of which satisfy the Interlisp predicate **NUMBERP**. Thus, **NUMBERP** is no longer the simple union of **FIXP** and **FLOATP**. It used to be that a program containing

```
(if (NUMBERP X)
 then (if (FIXP X)
 then ...assume X is an integer ...
 else ...can assume X is floating point...))
```

would be correct in Interlisp. However, this is no longer true; this program will not deal correctly with ratios or complex numbers, which are **NUMBERP** but neither **FIXP** nor **FLOATP**.

### Section 7.2 Integer Arithmetic

---

When typing to a *new* Interlisp Executive, the input syntax for integers of radix other than 8 or 10 has been changed to match that of Common Lisp. Use # instead of |, e.g., #b10101 is the new syntax for binary numbers, #x1A90 for hexadecimal, etc. Suffix Q is still recognized as specifying octal radix, but you can also use Common Lisp's #o syntax.

### (I:7.4)

In the Lyric release, the FASL machinery would handle some positive literals incorrectly, reading them back as negative numbers. The numbers handled incorrectly were those numbers  $x$  greater than  $2^{31}-1$  for which  $(\text{mod}(\text{integer-length } x) 8)$  was zero. The Medley release fixes this situation. Any files containing such numbers should be recompiled.

---

## Chapter 10 Function Definition, Manipulation, and Evaluation

---

### Section 10.1 Function Types

---

All Interlisp **NLAMBDAs** appear to be macros from Common Lisp's point of view. This is discussed at greater length in *Common Lisp Implementation Notes*, Chapter 8, Macros.

### Section 10.6 Macros

---

(**EXPANDMACRO** *EXP QUIETFLG* — — )

[Function]

**EXPANDMACRO** only works on Interlisp macros, those appearing on the **MACRO**, **BYTEMACRO** or **DMACRO** properties of symbols. Use **CL:MACROEXPAND-1** to expand Common Lisp macros and those Interlisp macros that are visible to the Common Lisp compiler and interpreter.

#### Section 10.6.1 DEFMACRO

---

(*l:10.24*)

Common Lisp does not permit a symbol to simultaneously name a function and a macro. In Lyric, this restriction also applies to Interlisp macros defined by **DEFMACRO**. That is, evaluating **DEFMACRO** for a symbol automatically removes any function definition for the symbol. Thus, if your purpose for using a macro is to make a function compile in a special way, you should instead use the new form **XCL:DEFOPTIMIZER**, which affects only compilation. The *Xerox Common Lisp Implementation Notes* describe **XCL:DEFOPTIMIZER**.

Interlisp **DMACRO** properties have typically been used for implementation-specific optimizations. They are not subject to the above restriction on function definition. However, if a symbol has both a function definition and a **DMACRO** property, the Lisp compiler assumes that the **DMACRO** was intended as an optimizer for the old Interlisp compiler and ignores it.

---

## Chapter 11 Stack Functions

---

### Section 11.1 The Spaghetti Stack

---

Stack pointers now print in the form

#<Stackp address/frame-name>.

Some restrictions were placed on spaghetti stack manipulations in order to integrate reasonably with Common Lisp's **CL:CATCH** and **CL:THROW**. In Lyric, it is an error to return to the same frame twice, or to return to a frame that has been unwound through. This means, for example, that if you save a stack pointer to one of your ancestor frames, then perform a **CL:THROW** or **RETFROM** that returns "around" that frame, i.e., to an ancestor of that frame, then

the stack pointer is no longer valid, and any attempt to use it signals an error "Stack Pointer has been released". It is also an error to attempt to return to a frame in a different process, using **RETFROM**, **RETTO**, etc.

The existence of spaghetti stacks raises the issue of under what circumstances the cleanup forms of **CL:UNWIND-PROTECT** are performed. In Lisp, **CL:THROW** always runs the cleanup forms of any **CL:UNWIND-PROTECT** it passes. Thanks to the integration of **CL:UNWIND-PROTECT** with **RESETLST** and the other Interlisp context-saving functions, **CL:THROW** also runs the cleanup forms of any **RESETLST** it passes. The Interlisp control transfer constructs **RETFROM**, **RETTO**, **RETEVAL** and **RETAPPLY** also run the cleanup forms in the analogous case, viz., when returning to a direct ancestor of the current frame. This is a significant improvement over prior releases, where **RETFROM** never ran any cleanup forms at all.

In the case of **RETFROM**, etc, returning to a non-ancestor, the cleanup forms are run for any frames that are being abandoned as a result of transferring control to the other stack control chain. However, this should not be relied on, as the frames would not be abandoned at that time if someone else happened to retain a pointer to the caller's control chain, but subsequently never returned to the frame held by the pointer. Cleanup forms are *not* run for frames abandoned when a stack pointer is released, either explicitly or by being garbage-collected. Cleanup forms are also not run for frames abandoned because of a control transfer via **ENVEVAL** or **ENVAPPLY**. Callers of **ENVEVAL** or **ENVAPPLY** should consider whether their intent would be served as well by **RETEVAL** or **RETAPPLY**, which *do* run cleanup forms in most cases.

---

## Chapter 12 Miscellaneous

---

### Section 12.4 System Version Information

---

All the functions listed on page 12.12 in the *Interlisp-D Reference Manual* have had their symbols moved to the LISP (CL) package. They are *not* shared with the INTERLISP package and any references to them in your code will need to be qualified i.e., **CL:name**.

### Section 12.8 Pattern Matching

---

Pattern matching is no longer a standard part of the environment. The functionality for Pattern matching can be found in the Lisp Library Module called **MATCH**.

[This page intentionally left blank]



## VOLUME II—ENVIRONMENT

---

### Chapter 13 Interlisp Executive

---

[This chapter of the *Interlisp-D Reference Manual* has been renamed Chapter 13, Executives.]

Lisp has a new kind of Executive (or Exec), designed for use in an environment with both Interlisp and Common Lisp. This executive is available in three standard modes, distinguished by their default settings for package and readtable:

|     |                                                       |
|-----|-------------------------------------------------------|
| XCL | New Exec. Uses XCL readtable, XCL-USER package        |
| CL  | New Exec. Uses LISP readtable, USER package           |
| IL  | New Exec. Uses INTERLISP readtable, INTERLISP package |

In addition, the old Interlisp executive, the "Programmer's Assistant", is still available in this release for the convenience of Koto users:

|               |                                                                                                                                                                  |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| OLD-INTERLISP | Old "Programmer's Assistant" Exec. Uses OLD-INTERLISP-T readtable, INTERLISP package. It is likely that this executive will not be supported in future releases. |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|

When Lisp starts, it is running a single executive, the XCL Exec. You can spawn additional executives by selecting EXEC from the background menu. The type of an executive is indicated in the title of its window; e.g., the initial executive has title "Exec (XCL)". Each executive runs in its own process; when you are finished with an executive, you can simply close its window, and the process is killed.

The new executive is modeled, somewhat, on the old "Programmer's Assistant" executive and, to a first approximation, you can type to it just as you did in past releases. You should note, however, that the default executive (XCL) expects Common Lisp input syntax, and reads symbols relative to the XCL-USER package. This means that to type Interlisp symbols, you must prefix the symbol with the characters "IL:" (in upper or lower case). And even in the new IL executive, the readtable being used is the new INTERLISP readtable, in which the characters colon (:), vertical bar (|) and hash (#) all have different meanings than in Koto.

The OLD-INTERLISP exec, with one exception, uses exactly the same input syntax as in Koto; this means in particular that colon cannot be used to type package-qualified symbols, since colon is an ordinary character there. The one exception is that there *is* a package delimiter character in the OLD-INTERLISP readtable, should you have a need to use it—Control-↑, which usually echoes as "↑↑", though it may appear as a black rectangle in some fonts.

The new executive does differ from the old one in several respects, especially in terms of its programmatic interface. Complete details

of the new executive can be found in Appendix A. The Exec. Some of the important differences are:

- Executives are numbered

Executives, other than the first one, are labeled with a distinct number. This number appears in the exec window's title, and also in its prompt, next to the event number. The OLD-INTERLISP executive does not include this exec number.

- Event number allocation

The numbers for events are allocated at the time the prompt for the event is printed, but all execs still share a common event number space and history list. This means that ?? shows all events that have occurred in *any* executive, though not necessarily in the order in which the events actually occurred (since it is the order in which the event numbers were allocated). Events for which the type-in has not been completed are labeled "<in progress>" in the ?? listing. In the old executive, event numbers are not allocated until type-in is complete, which means that the number printed next to the prompt is not necessarily the number associated with the event, in the case that there has been activity in other executives.

In the new executive, relative event specifications are local to the exec; e.g., **-1** refers to the most recent event in that specific exec. In the old executive, **-1** referred to the immediately preceding event in *any* executive.

- New facility for commands

The old Executive has commands based on **LISPMACROS**. The new Executive has its own command facility, **XCL:DEFCOMMAND**, which allows commands to be named without regard to package, and to be written with familiar Common Lisp style of argument list.

- Commands are typed *without* parentheses

In the old executive, a command could be typed with or without enclosing parentheses. In the new executive, a parenthesized form is always interpreted as an EVAL-style input, never a command.

- **SETQ** does not interact with the File Manager

In the Koto release, when you typed in the Exec

**(SETQ FOO some-new-value-for-FOO)**

the executive responded (**FOO reset**), and the file package was told that **FOO**'s value changed. Any files on which **FOO** appeared as a variable would then be marked as needing to be cleaned up. If **FOO** appeared on no file, you'd be prompted to put it on one when you ran (**FILES?**).

This is still the case in the old executive. However, it is no longer the case in the new executive. If you are setting a variable that is significant to a program and you want to save it on a file, you should use the Common Lisp macro **CL:DEFPARAMETER** instead of **SETQ**. This will give the symbol a definition of type **VARIABLES** (rather than **VAR**), and it will be noticed by the File manager. If you want to change the value of the variable, you must either use



**CL:DEFPARAMETER** again, or edit the variable using **ED** (not **DV**).

- Programmatic interface completely different

As a first approximation, all the functions and variables in *IRM* Sections 13.3 (except the **LISPXPRINT** family) and 13.6 apply only to the Old Interlisp Executive, unless specified otherwise in Appendix A. In particular, the variables **PROMPT#FLG**, **PROMTPCHARFORMS**, **SYSPRETTYFLG**, **HISTORYSAVEFORMS**, **RESETFORMS**, **ARCHIVEFN**, **ARCHIVEFLG**, **LISPXUSERFN**, **LISPXMACHOS**, **LISPXHISTORYMACROS** and **READBUF** are not used by the new Exec. The function **USEREXEC** invokes an old-style Executive, but uses the package and readtable of its caller. The function **LISPXUNREAD** has no effect on the new Exec. Callers of **LISPXEVAL** are encouraged to use **EXEC-EVAL** instead.

Some subsystems still use the old-style Executive—in particular, the tty structure editor.

---

## Chapter 14 Errors and Breaks

---

Lisp extends the Interlisp break package to support multiple values and the Common Lisp lambda syntax. Interlisp errors have been converted to Common Lisp conditions.

Note that Sections 14.2 through 14.6 in the *Interlisp-D Reference Manual* have been replaced by new Debugger information (see *Common Lisp Implementation Notes*).

---

### Section 14.3 Break Commands

---

(II:14.6)

The **!EVAL** debugger command no longer exists.

(II:14.10-11)

The Break Commands = and -> are no longer supported.

---

### Section 14.6 Creating Breaks with **BREAK1**

---

While the function **BREAK1** still exists, broken and traced functions are no longer redefined in terms of it. More primitive constructs are used.

---

### Section 14.7 Signalling Errors

---

Interlisp errors now use the new XCL error system. Most of the functions still exist for compatibility with existing Interlisp code, but the underlying machinery is different. There are some incompatible differences, however, especially with respect to error numbers.

The old Interlisp error system had a set of registered error numbers for well known error conditions, and all other errors were identified

by a string (an error message). In the new Lisp error system, all errors are handled by signalling an object of type **XCL:CONDITION**. The mapping from Interlisp error numbers to Lisp conditions is given below in Section 14.10.

Since one cannot in general map a condition object to an Interlisp error number, the function **ERRORN** no longer exists. The equivalent functionality exists by examining the special variable **\*LAST-CONDITION\***, whose value is the condition object most recently signaled.

**(ERRORX ERXM)** calls **CL:ERROR** after first converting **ERXM** into a condition in the following way: If **ERXM** is **NIL**, the value of **\*LAST-CONDITION\*** is used; if **ERXM** is an Interlisp error descriptor, it is first converted to a condition; finally, if **ERXM** is already a condition, it is passed along unchanged. **ERRORX** also sets up a proceed case for **XCL:PROCEED**, which will attempt to re-evaluate the caller of **ERRORX**, much as **OK** did in the old Interlisp break package.

**ERROR**, **HELP**, **SHOULDNT**, **RESET**, **ERRORMESS**, **ERRORMESS1**, and **ERRORSTRING** work as before. All output is directed to **\*ERROR-OUTPUT\***, initially the terminal.

**ERROR!** is equivalent to the new error system's **XCL:ABORT** proceed function, except that if no **ERRORSET** or **XCL:CATCH-ABORT** is found, it unwinds all the way to the top of the process.

**SETERRORN** converts its arguments into a condition, then sets the value of **\*LAST-CONDITION\*** to the result.

---

## Section 14.8 Catching Errors

**ERRORSET**, **ERSETQ** and **NLSETQ** have been reimplemented in terms of the new error system, but their behavior is essentially the same as before. **NLSETQ** catches all errors (conditions of type **CL:ERROR** and its descendants), and sets up a proceed case for **XCL:ABORT** so that **ERROR!** will return from it. **ERSETQ** also sets up a proceed case for **XCL:ABORT**, though it does not catch errors.

One consequence of the new implementation is that there are no longer frames named **ERRORSET** on the stack; programs that explicitly searched for such frames will have to be changed.

**ERRORTYPELIST** is no longer supported. The equivalent functionality is provided by default handlers. Although condition handlers provide a more powerful mechanism for programmatically responding to an error condition, old **ERRORTYPELIST** entries generally cannot be translated directly. Condition handlers that want to resume a computation (rather than, say, abort from a well-know stack location) generally require the cooperation of a proceed case in the signalling code; there is no easy way to provide a substitute value for the "culprit" to be re-evaluated in a general way.

One important difference between **ERRORTYPELIST** and condition handlers is their behavior with respect to **NLSETQ**. In Koto, the relevant error handler on **ERRORTYPELIST** would be tried, even for errors occurring underneath an **NLSETQ**. In Lyric, the **NLSETQ** traps all errors before the default condition handlers can see the

error. This means, for example, that the behavior of **(NLSETQ (OPENSTREAM --))** is now different if the **OPENSTREAM** causes a file not found error—in Koto, the system would search **DIRECTORIES** for the file; in Lyric, the **NLSETQ** returns **NIL** immediately without searching, since the default handler for **XCL:FILE-NOT-FOUND** is not invoked.

## Section 14.9 Changing and Restoring System State

---

The special forms **RESETLST**, **RESETSAVE**, **RESETVAR**, **RESETVARS** and **RESETFORM** still exist, but are implemented by a new mechanism that also supports Common Lisp's **CL:UNWIND-PROTECT**. Common Lisp's **CL:THROW** and (in most cases) Interlisp's **RETFROM** and related control transfer constructs cause the cleanup forms of both **CL:UNWIND-PROTECT** and **RESETLST** (etc) to be performed. This is discussed in more detail in the notes for Chapter 11, the stack.

## Section 14.10 Error List

---

Most of the Interlisp errors are mapped into condition types in Lisp. Some are no longer supported. Following is the list of error type mappings. The first name is the condition type that the error descriptor turns into. If there is a second name, it is the slot whose value is set to **CADR** of the error descriptor. Any additional pairs of items are the values of other slots set by the mapping. Attempting to use an unsupported error type number will result in a simple error to that effect.

- 0    Obsolete
- 1    Obsolete
- 2    **STACK-OVERFLOW**
- 3    **ILLEGAL-RETURN**
- 4    **XCL:SIMPLE-TYPE-ERROR** *CULPRIT* :EXPECTED-TYPE 'LIST
- 5    **XCL:SIMPLE-DEVICE-ERROR** *MESSAGE*
- 6    **XCL:ATTEMPT-TO-CHANGE-CONSTANT**
- 7    **XCL:ATTEMPT-TO-RPLAC-NIL** *MESSAGE*
- 8    **ILLEGAL-GO TAG**
- 9    **XCL:FILE-WONT-OPEN** *PATHNAME*
- 10   **XCL:SIMPLE-TYPE-ERROR** *CULPRIT* :EXPECTED-TYPE 'CL:NUMBER
- 11   **XCL:SYMBOL-NAME-TOO-LONG**
- 12   **XCL:SYMBOL-HT-FULL**
- 13   **XCL:STREAM-NOT-OPEN** *STREAM*
- 14   **XCL:SIMPLE-TYPE-ERROR** *CULPRIT* :EXPECTED-TYPE 'CL:SYMBOL
- 15   Obsolete
- 16   **END-OF-FILE** *STREAM*
- 17   **INTERLISP-ERROR** *MESSAGE*
- 18   Not supported (control-B interrupt)

- 19 **ILLEGAL-STACK-ARG** *ARG*
- 20 Obsolete
- 21 **XCL:ARRAY-SPACE-FULL**
- 22 **XCL:FS-RESOURCES-EXCEEDED**
- 23 **XCL:FILE-NOT-FOUND** *PATHNAME*
- 24 Obsolete
- 25 **INVALID-ARGUMENT-LIST** *ARGUMENT*
- 26 **XCL:HASH-TABLE-FULL** *TABLE*
- 27 **INVALID-ARGUMENT-LIST** *ARGUMENT*
- 28 **XCL:SIMPLE-TYPE-ERROR** *CULPRIT* **:EXPECTED-TYPE**  
**'ARRAYP**
- 29 Obsolete
- 30 **STACK-POINTER-RELEASED** *NAME*
- 31 **XCL:STORAGE-EXHAUSTED**
- 32 Not supported (attempt to use item of incorrect type)
- 33 Not supported (illegal data type number)
- 34 **XCL:DATA-TYPES-EXHAUSTED**
- 35 **XCL:ATTEMPT-TO-CHANGE-CONSTANT**
- 36 Obsolete
- 37 Obsolete
- 38 **XCL:SIMPLE-TYPE-ERROR** *CULPRIT* **:EXPECTED-TYPE**  
**'READTABLEP**
- 39 **XCL:SIMPLE-TYPE-ERROR** *CULPRIT* **:EXPECTED-TYPE**  
**'TERMTABLEP**
- 40 Obsolete
- 41 **XCL:FS-PROTECTION-VIOLATION**
- 42 **XCL:INVALID-PATHNAME** *PATHNAME*
- 43 Not supported (user break)
- 44 **UNBOUND-VARIABLE** *NAME*
- 45 **UNDEFINED-CAR-OF-FORM** *FUNCTION*
- 46 **UNDEFINED-FUNCTION-IN-APPLY**
- 47 **XCL:CONTROL-E-INTERRUPT**
- 48 **XCL:FLOATING-UNDERFLOW**
- 49 **XCL:FLOATING-OVERFLOW**
- 50 Not supported (integer overflow)
- 51 **XCL:SIMPLE-TYPE-ERROR** *CULPRIT* **:EXPECTED-TYPE**  
**'CL:HASH-TABLE**
- 52 **TOO-MANY-ARGUMENTS** *CALLEE* **:MAXIMUM CL:CALL-  
ARGUMENTS-LIMIT**

Note that there are many other condition types in Lisp; see the error system documentation in the *Common Lisp Implementation Notes* for details.

## Chapter 15 Breaking Functions and Debugging

---

In Lyric the uses of **BREAK**, **TRACE**, and **ADVISE** are unchanged, from the user's point of view, but the internals of their implementation are quite different.

For complete documentation on the new implementation of breaking, tracing and advising, see the *Common Lisp Implementation Notes*, Section 25.3.

In particular, you should note the following differences:

- The variable **BRKINFOLST** no longer exists and the format of the value of the variable **BROKENFNFS** has changed. In addition, the **BRKINFO** property is no longer used.
- **BREAK** and **TRACE** no longer work on CLISP words.
- The **BREAKIN** and **UNBREAKIN** functions no longer exist. No comparable facility exists in Lisp. The user can manually insert calls to the Common Lisp function **CL:BREAK** in order to create a breakpoint at that point in the function.

Please note the following additional changes to breaking functions:

### Section 15.1 Breaking Functions and Debugging

---

**(BREAK0 FN WHEN COMS — —)**

[Function]

The function **BREAK0** now works when applied to an undefined function. This allows you to use the breaking facility to create "stubs" that generate a breakpoint when called. You can then examine the arguments passed and use the **RETURN** command in the debugger to return the proper result(s).

The "break commands" facility (the *COMS* argument) is no longer supported. **BREAK0** now signals an error when supplied with a non-**NIL** third argument. If you need finer control over the functioning of breakpoints you are directed to the **ADVISE** facility; it offers complete control of how and when the given function is evaluated.

Passing a non-atomic argument in the form **(FN1 IN FN2)** as the first argument to **BREAK0** still has the effect of creating a breakpoint wherever *FN2* calls *FN1*. However, it no longer creates a function named *FN1-IN-FN2* to do so. In addition, the format of the value of the **NAMESCHANGED** property has changed and the **ALIAS** property is no longer used.

**(TRACE X)**

[Function]

**TRACE** is no longer a special case of **BREAK**, though the functions **UNBREAK** and **REBREAK** continue to work on traced functions.

In addition, the function **TRACE** no longer calls **BREAK0** in order to do its job. Also, non-atomic arguments to **TRACE** no longer specify forms the user wishes to see in the tracing output.

**(UNBREAK X)**

[Function]

The function **UNBREAK** is no longer implemented in terms of **UNBREAK0**, although that function continues to exist.

## Section 15.2 Advising

The implementation of advising has been completely reworked. While the semantics implied by the code shown in Section 15.2.1 of the *Interlisp-D Reference Manual* is still supported, the details are quite different. In particular, it is now possible to advise functions that return multiple values and for **AFTER**-style advice to access those values. Also, all advice is now compiled, rather than interpreted. The advising facility no longer makes use of the special forms **ADV-PROG**, **ADV-RETURN**, and **ADV-SETQ**.

You should also note the following changes to the advise facility:

- The editing of advice has changed slightly. In previous releases, the advice and original function-body were edited simultaneously. In Lyric, they can only be edited separately. When you finish editing the advice for a function, that function is automatically re-advised using the new advice.
- The variable **ADVINFOLST** no longer exists and the format of the value of the variable **ADVISEDFN**s has changed. In addition, the properties **ADVICE** and **READVICE** are no longer used, except in the handling of advice saved on files from previous releases. Advice saved in Lyric does not use the **READVICE** property.
- The function **ADVISEDUMP** no longer exists.
- Advice saved on files in previous releases can, in general, be loaded into the Lyric system compatibly. A known exception is the case in which a list of the form **(FN1 IN FN2)** was given to the **ADVICE** or **ADVISE** file package commands. When **READVICE** is called on such a name, the old-style advice, on the **READVICE** property of the symbol **FN1-IN-FN2**, will not be found. This will eventually lead to an **XCL:ATTEMPT-TO-RPLAC-NIL** error. The user should evaluate the form  
**(RETFROM 'READVICE1)**  
in the debugger to proceed from the error and later evaluate  
**(READVICE FN1-IN-FN2)**  
by hand to install the advice.

- The **ADVISE** and **ADVISE** File Manager commands now accept three kinds of arguments:
  - a symbol, naming an advised function,
  - a list in the form **(FN1 :IN FN2)**, and
  - a symbol of the form **FN1-IN-FN2**.Arguments of the form **(FN1 IN FN2)** are not acceptable any longer. Arguments of the form **FN1-IN-FN2** should be converted into the equivalent form **(FN1 :IN FN2)**.

**(ADVISE WHO WHEN WHERE WHAT)**

[Function]

In the Lyric release of Lisp, **ADVISE** has some changes in the way arguments are treated and the possible values for those arguments. Most notably:

- In earlier releases, you could call **ADVISE** with only one argument, the name of a function. In this case, **ADVISE** "set up" the named function for advising, but installed no advice. This usage is no longer supported.
- Previously, an undocumented value of **BIND** was accepted for the **WHEN** argument to **ADVISE**. This kind of advice is no longer supported. It can be adequately simulated using **AROUND** advice.

In addition, advising Common Lisp functions works somewhat differently with respect to a function's arguments. The arguments are not available by name. Instead, the variable **XCL:ARGLIST** is bound to a list of the values passed to the function and may be changed to affect what will be passed on.

As with the breaking facility (see above), **ADVISE** no longer creates a function named **FN1-IN-FN2** as a part of advising **(FN1 IN FN2)**.

---

## Chapter 16 List Structure Editor

---

The list structure editor, DEdit, is not part of the Lisp environment. It is now a Lisp Library Module. Chapter 16 has been renamed Structure Editor.

SEdit, the new Lisp editor, replaced DEdit in the Lyric release. The description of SEdit may be found in Appendix B of this volume. The commands used to invoke both SEdit and DEdit are the same.

Following is a description of the interface to the Lisp editor.

## Switching Between Editors

If you have both SEdit and DEdit loaded, you can switch between them by calling: **(EDITMODE 'EDITORNAME)** where *EDITORNAME* is one of the symbols SEdit or DEdit.

## Packages

The **ED** editor interface accepts TYPE information from the Interlisp or Common Lisp packages.

## Starting a Lisp Editor

In the XCL environment, calling ED with a pathname will start the editor on the coms of the file (as if DC had been called).

**(ED NAME &OPTIONAL OPTIONS)** [Function]

This function starts the Lisp editor. **ED** is the default interface to the editor. SEdit is the default Lisp editor. The same symbol, **ED**, is exported in both the IL and CL packages.

*NAME* is the name of any File Manager object.

*OPTIONS* is either a single symbol or a list of symbols, each of which is either a File Manager type or one or more of the keywords **:DISPLAY**, **:DONTWAIT**, **:CURRENT**, **:COMPILE-ON-COMPLETION**, **:CLOSE-ON-COMPLETION**, or **:NEW**. If exactly one File Manager type is given, **ED** tries to edit that type of definition for *NAME*. If more than one type is given in *OPTIONS*, **ED** will determine for which of them *NAME* has a definition. If a definition exists for more than one of the types, **ED** gives you a choice of which one to edit. If no File Manager types are given, **ED** treats *OPTIONS* as a list of all of the existing types; thus you are given a choice of all of the existing definitions of *NAME*.

The variable **FILEPKGTYPES** contains a complete list of the currently-known manager types.

If the keyword **:DISPLAY** is included in *OPTIONS*, **ED** uses menus for any prompting, (e.g., to choose one of several possible definitions to edit). If **:DISPLAY** is not included, **ED** prints its queries to and reads the user's replies from **\*QUERY-IO\*** (usually the Exec in which you are typing). Thus all of the following are correct ways to call the editor:

```
(ED 'NAME :DISPLAY)
(ED 'NAME 'FUNCTIONS)
(ED 'NAME '(:DISPLAY))
(ED 'NAME '(FUNCTIONS :DISPLAY))
(ED 'NAME '(FUNCTIONS VARIABLES :DISPLAY))
```

The other keywords are interpreted as follows:



**:CURRENT**

This is a new option with Medley that causes ED to call TYPESOF with SOURCE=CURRENT. This prevents TYPESOF from searching FILECOMS and from looking in WHERE-IS databases. The **CURRENT** option looks only for definitions that are currently loaded. When you know that the definition is loaded, use of the **CURRENT** option results in ED being significantly faster.

**:DONTWAIT**

Lets the edit interface return right away, rather than waiting for the edit to be complete. **DF**, **DV**, **DC**, and **DP** specify this option now, so editing from the exec will not cause the exec to wait.

**:NEW**

Lets you install a new definition for the name to be edited. You will be asked what type of dummy definition you wish to install based on which file manager types were included in *OPTIONS*.

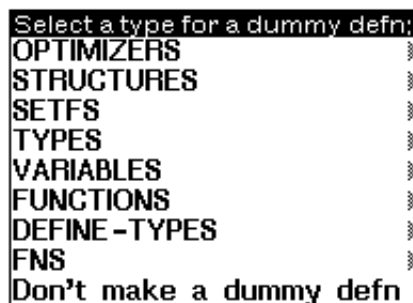
**:COMPILE-ON-COMPLETION**

This option specifies that the definition being edited should be compiled upon completion regardless of the completion command used.

**:CLOSE-ON-COMPLETION**

Tells the editor that it must close the editor window after the first completion. So in SEdit, CONTROL-X will close the window; shrinking the window is not allowed. Editor windows opened by the exec command **FIX** specify this option.

If *NAME* does not have a definition of any of the given types, **ED** can create a dummy definition of any of those types. If **:DISPLAY** is provided in *OPTIONS*, **ED** will pop-up the following menu asking you which type of definition to install. Select the template for the type of definition you wish to create from the DEFN menus and submenus:



```
Select a type for a dummy defn:
OPTIMIZERS >>>
STRUCTURES >>>
SETFS >>>
TYPES >>>
VARIABLES >>>
FUNCTIONS >>>
DEFINE-TYPES >>>
FNS >>>
Don't make a dummy defn
```

New kinds of dummy definitions can be added to the system through the use of the **:PROTOTYPE** option to **XCL:DEFDEFINER**.

## Mapping the Old Edit Interface to ED

The old functions for starting the Lisp editor (**DF**, **DV**, **DP**, and **DC**) have been reimplemented so that they work with Common Lisp. The old edit commands map to the new editor function (ED) as follows:

```
DF NAME ⇒ (ED 'NAME' (FUNCTIONS FNS :DONTWAIT))
DV NAME ⇒ (ED 'NAME' (VARIABLES VARS :DONTWAIT))
DP NAME ⇒ (ED 'NAME' (PROPERTY-LIST :DONTWAIT))
DP NAME MYPROP ⇒ (ED '(NAME MYPROP)' (PROPS :DONTWAIT))
DC NAME ⇒ (ED 'NAME' (FILES :DONTWAIT))
```

Thus, for example, when **DF** is invoked it looks first for Common Lisp **FUNCTIONS** and then for Interlisp **FNS**. **DV**, **DP** and **DC** operate in an analogous fashion.

## Editing Values Directly

The **TYPE** you specify for the object you want to edit determines how that object is edited, i.e. by **DEFINITION** or **VALUE**. Normally you want to edit the **DEFINITION** (this is the default). For example, suppose **FOO** is defined as a variable; to start the editor on the **DEFINITION** of **FOO**, use the form:

```
(ED 'FOO) or (ED 'FOO 'VARIABLES)
```

There may be times when you do not have access to the **DEFINITION** of an object that you need to edit. This can occur when you do not have the source code loaded. You can edit its **VALUE** directly using the form:

```
FOR VARIABLES: ⇒ (ED 'NAME 'IL:VARS)
```

```
FOR FUNCTIONS: ⇒ (ED 'NAME 'IL:FNS)
```

By starting the editor on the **VALUE** of an object, you can change its value without changing its definition. (AR 8971)

To start the editor on the **VALUE** of **FOO**, for example, use the form:

```
(ED 'FOO 'VARS)
```

### EXAMPLE:

When you load a compiled file, the **DEFINITION** of an object is not loaded. Only the **VALUE** is loaded. The compiler does not store the defining forms for objects. Suppose you have compiled code for a system file loaded, but you do not have access to the sources that contain the **DEFINITIONS**, and you need to change the value of a system variable, say **NETWORKLOGINFO**. This variable has a defining form and the system knows this, but the form is not loaded and is not available. You can edit the **VALUE** of the variable directly using:

```
(ED 'NETWORKLOGINFO 'IL:VARS)
```

An editor window opens displaying the **VALUE** of **NETWORKLOGINFO**:

```
SEdit NETWORKLOGINFO Package: INTERLISP
((TENEX (LOGIN "LOGIN " USERNAME " " PASSWORD " ↑M")
 (ATTACH "ATTACH " USERNAME " " PASSWORD " ↑M")
 (WHERE "WHERE " USERNAME CR
 "ATTACH " USERNAME
 " " PASSWORD CR))
 (TOPS20 (LOGIN "LOGIN " USERNAME CR PASSWORD CR)
 (ATTACH "ATTACH " USERNAME "lama " CR PASSWORD CR)
 (WHERE "LOGIN " USERNAME CR PASSWORD CR))
 (UNIX (LOGIN WAIT CR WAIT USERNAME CR WAIT PASSWORD CR))
 (IFS (LOGIN "Login " USERNAME " " PASSWORD CR) (ATTACH))
 (NS (LOGIN "Logon" CR USERNAME CR PASSWORD CR))
 (VMS (LOGIN USERNAME CR PASSWORD CR)))
```

---

## Section 16.18 Editor Functions

### (II:16.74)

The function **FINDCALLERS** has the following limitations in Lisp:

1. **FINDCALLERS** only identifies by name the occurrences inside of Interlisp FNS, not Common Lisp FUNCTIONS.
2. Because **FINDCALLERS** uses a textual search, it may report more occurrences of the specified **ATOMS** than there actually are, if the file contains symbols by the same name in another package, or symbols with the same p-name but different alphabetic case. **EDITCALLERS** still edits only the actual occurrences, since it reads the functions and operates on the real Lisp structure, not its printed representation.

---

## Chapter 17 File Package

The Interlisp-D File Package has been renamed the File Manager. Its operation is unchanged; however, it has been extended to manipulate, load and save Common Lisp functions, variables, etc. It also allows specification of the reader environment (package and readtable) to use when writing and reading a file, solving the problem of compatibility between old and new (Common Lisp) syntax.

Note that although source files from earlier releases can be loaded into Lyric, files produced by the File Manager in the Lyric release cannot be loaded into previous releases. This is true for several reasons, the most important being that previous releases did not have packages, so symbols cannot be read back consistently.

The new File Manager includes several new types to deal with the various definition forms supported in Xerox Common Lisp. The following table associates each new type with the forms that produce definitions of that type:

|              |                                                                                                                   |
|--------------|-------------------------------------------------------------------------------------------------------------------|
| FUNCTIONS    | <b>CL:DEFUN, CL:DEFMACRO, CL:DEFINE-MODIFY-MACRO, XCL:DEFINLINE, XCL:DEFDEFINER, XCL:DEFINE-PROCEED-FUNCTION.</b> |
| VARIABLES    | <b>CL:DEFCONSTANT, CL:DEFVAR, CL:DEFPARAMETER, XCL:DEFGLOBALVAR, XCL:DEFGLOBALPARAMETER</b>                       |
| STRUCTURES   | <b>CL:DEFSTRUCT, XCL:DEFINE-CONDITION</b>                                                                         |
| TYPES        | <b>CL:DEFTYPE</b>                                                                                                 |
| SETFS        | <b>CL:DEFSETF, CL:DEFINE-SETF-METHOD</b>                                                                          |
| DEFINE-TYPES | <b>XCL:DEF-DEFINE-TYPE</b>                                                                                        |
| OPTIMIZERS   | <b>XCL:DEFOPTIMIZER</b>                                                                                           |
| COMMANDS     | <b>XCL:DEFCOMMAND</b>                                                                                             |

Note that the types listed above, as well as all the old File Manager types, are symbols in the INTERLISP package. In addition, the "filecoms" variable of a file and its rootname are also both in the INTERLISP package. You should be careful when typing to a Common Lisp exec to qualify all such symbols with the prefix **IL:**; e.g.,

```
3>(setq il:foocoms '((il:functions bar) (il:prop il:filetype il:foo)))
```

to indicate you want the function BAR (in the current package) to live on a file with rootname FOO, and also that FOO's FILETYPE property should be saved.

## Reader Environments and the File Manager

### (II:17.1)

In order for **READ** to correctly read back the same expression that **PRINT** printed, it is necessary that both operations be performed in the same reader environment, i.e., the collection of parameters that affect the way the reader interprets the characters appearing on the input stream. In previous releases of Interlisp there was, for all practical purposes, a single such environment, defined entirely by the readtable *FILERDTBL*. In the Lyric release of Lisp there are two significantly different readtables in which to read (Common Lisp and Interlisp). In addition, there are more parameters than just the readtable that can potentially affect **READ**: the current package and the read base (the bindings of **\*PACKAGE\*** and **\*READ-BASE\***).

To handle this diversity, a new type of object is introduced, the **READER-ENVIRONMENT**, consisting of a readtable, a package, and a read/print base. Every file produced by the File Manager has a header at the beginning specifying the reader environment for that file. **MAKEFILE** and the compiler produce this header, while **LOAD**, **LOADFNS**, and other file-reading functions read the header in order to set their reading environment correctly. Files written in older releases of Lisp lack this header and are interpreted as

having been written in the environment consisting of the readtable *FILERDTBL* and the package *INTERLISP*. Thus, you need take no special action to be able to load Koto source files into Lyric; characters that are "special" in Common Lisp, such as colon, semi-colon and hash, are interpreted as the "ordinary" characters they were in Koto.

The File Manager's reader environments are specified as a property list of alternating keywords and values of the form **(:READTABLE *readtable* :PACKAGE *package* :BASE *base*)**. The **:BASE** pair is optional and defaults to 10. The values for *readtable* and *package* should either be strings naming a readtable and package, or expressions that can be evaluated to produce a readtable and package. In the former case, the readtable or package *must* be one that already exists in a virgin Lisp sysout (or at least in any Lisp image in which you might attempt *any* operation that reads the file). If an expression is used, care should be exercised that the expression can be evaluated in an environment where no packages or readtables, other than the documented ones, are presumed to exist. For hints and guidelines on writing the *package* expression for files that create or use their own private packages, please see Chapter 11 of the *Common Lisp Implementation Notes*.

When **MAKEFILE** is writing a source file, it uses the following algorithm to determine the reading environment for the new file:

1. If the root name for the file has the property **MAKEFILE-ENVIRONMENT**, the property's value is used. It should be in the form described above. Note that if you want the file always to be written in this environment, you should save the **MAKEFILE-ENVIRONMENT** property itself on the file, using a **(PROP MAKEFILE-ENVIRONMENT *file*)** command in the filecoms.
2. If a previous version of the file exists, **MAKEFILE** uses the previous version's environment. **MAKEFILE** does this even when given option **NEW** or the previous version is no longer accessible, assuming it still has the previous version's environment in its cache. If the previous version was written in an older release, and hence has no explicit reader environment, **MAKEFILE** uses the environment **(:READTABLE "INTERLISP" :PACKAGE "INTERLISP" :BASE 10)**.
3. If no previous version exists (this is a new file), **MAKEFILE** uses the value of **\*DEFAULT-MAKEFILE-ENVIRONMENT\***, initially **(:READTABLE "XCL" :PACKAGE "INTERLISP" :BASE 10)**.

Note that changing the value of **\*DEFAULT-MAKEFILE-ENVIRONMENT\*** only affects new files. If you decide you don't like the environment in which an existing file is written, you must give the file a **MAKEFILE-ENVIRONMENT** property to override any prior default.

Since the XCL readtable is case-insensitive, you should avoid using it for files that contain many mixed-case symbols or old-style Interlisp comments, as these will be printed with many escape

delimiters. This is why the default for reprinted Koto sources is the INTERLISP readtable.

The readtable named LISP (the pure Common Lisp readtable) should ordinarily not be used as part of a **MAKEFILE** environment. It exists solely for the use of "pure" Common Lisp (as in the CL Exec), and thus has no provision for font escapes (inserted by the Lisp prettyprinter) to be treated as whitespace. Most users will want to use either XCL or INTERLISP as the readtable for files.

If the environment for the new version of the file differs from that of the previous version, **MAKEFILE** copies unchanged FNS definitions by actually reading from the old file, rather than just copying characters as it otherwise would. Similarly, when **RECOMPILE** or **BRECOMPILE** attempt to recompile a file for which the previous compiled version's reader environment is different, they must compile afresh all the functions on the file, i.e., they behave like **TCOMPL** or **BCOMPL**.

---

## Modifying Standard Readtables

In the past, programmers have been periodically tempted to change standard readtables, such as **T** and **FILERDTBL**, typically by adding macros to read certain objects in a convenient way. For example, the PQUOTE LispUsers module defined single quote as a macro in **FILERDTBL**. Unfortunately, changing a standard readtable means that unless you are very careful, you cannot read other users' files that were not written with your change, and they cannot read your files without obtaining your macro. Furthermore, the effects are often subtle. Rather than breaking, the system merely reads the file incorrectly. For example, reading a file written with PQUOTE in an environment lacking PQUOTE produces many symbols with a single quote packed on the front.

This confusion can be avoided with **MAKEFILE** reader environments. To add your own special macro:

1. Copy some standard readtable; e.g., (COPYRDTBL "INTERLISP").
2. Give it a distinguished name of its own, by using (READTABLEPROP *rdtbl* 'NAME "yourname").
3. Make your change in the copied readtable.
4. Use your new private readtable to write your files: use its name ("yourname") in the **MAKEFILE-ENVIRONMENT** property of selected files and/or change **\*DEFAULT-MAKEFILE-ENVIRONMENT\*** to affect all your new files.
5. Make sure to save your new readtable. It is usually most convenient to include the code to create it (steps 1-3) in your system initialization, but you could even write a self-contained expression to use in a single file's **MAKEFILE-ENVIRONMENT** property.

With this strategy, your system will read all files in the proper environment—your own files with your private readtable and other users' files in their environments, including the standard environments, which you have carefully avoided polluting. If

another user tries to load one of your files into an environment that doesn't know about your private readtable, **LOAD** will give an error immediately (readtable not found), rather than loading the file quietly but incorrectly.

## Programmer's Interface to Reader Environments

---

The following function and macro are available for programmers to use. Note that reader environments only control the parameters that determine read/print consistency. There are other parameters, such as **\*PRINT-CASE\***, that affect the appearance of the output without affecting its ability to be read. Thus, reader environments are not sufficient to handle problems of, for example, repainting expressions on the display in exactly the same total environment in which they were first written.

**(MAKE-READER-ENVIRONMENT PACKAGE READTABLE BASE)** [Function]

Creates a **READER-ENVIRONMENT** object with the indicated components. The arguments must be valid values for the variables **\*PACKAGE\***, **\*READTABLE\*** and **\*PRINT-BASE\***; names are not sufficient. If any of the arguments is **NIL**, the current value of the corresponding variable is used. Thus **(MAKE-READER-ENVIRONMENT)** returns an object that captures the current environment.

**(WITH-READER-ENVIRONMENT ENVIRONMENT . FORMS)** [Macro]

Evaluates each of the *FORMS* with **\*PACKAGE\***, **\*READTABLE\***, **\*PRINT-BASE\*** and **\*READ-BASE\*** bound to the values in the *ENVIRONMENT* object. Both **\*PRINT-BASE\*** and **\*READ-BASE\*** are bound to the single *BASE* value in the environment.

**(GET-ENVIRONMENT-AND-FILEMAP STREAM DONTCACHE)** [Function]

Parses the header of a file produced by the File Manager and returns up to four values:

1. The reader environment in which the file was written;
2. The file's "filemap", used to locate functions on the file;
3. The file position where the FILECREATED expression starts;  
and
4. A value used internally by the File Manager.

*STREAM* can be a full file name, in which case this function returns **NIL** unless the information was previously cached. Otherwise, *STREAM* is a stream open for input on the file. It must be randomly accessible (unless information is available from the cache). If the file is in Common Lisp format (it begins with a comment), then value 1 is the default Common Lisp reader environment (readtable **LISP**, package **USER**) and the other values are **NIL**. Otherwise, if the file is not in File Manager format, values 1 and 2 are **NIL**, 3 is zero.

If *DONTCACHE* is true, the function does not cache any information it learns about File Manager files; otherwise, the information is cached to speed up future inquiries.

## Section 17.1 Loading Files

(II:17.5)

### Integration of Interlisp and Common Lisp **LOAD** functions

There are four kinds of files that can be loaded in Lisp:

1. Interlisp and Common Lisp source files produced by the File Manager using, for example, the **MAKEFILE** function.
2. Standard Common Lisp source files produced with a text editor either in Lisp or from some other Common Lisp implementation.
3. DFASL files of compiled code, produced by the new XCL Compiler, **CL:COMPILE-FILE** (extension DFASL)
4. LCOM files of compiled code, produced by the old Interlisp Compiler (**BCOMPL**, **TCOMPL**).

Types 1 and 4 were the only kind of files that you could load in Koto; types 2 and 3 are new with Lyric. Both **IL:LOAD** and **CL:LOAD** are capable of loading all four kinds of files. However, they use the following rules to make the types of files unambiguous so that they can be loaded in the correct reader environment.

- If the file begins with an open parenthesis (possibly after whitespace and font switch characters), it is assumed to be of type 1 or 4: files produced by the File Manager. The first expression on the file (at least) is assumed to be written in the old **FILERDTBL** environment; for new Lyric files this expression defines the reader environment for the remainder of the file. See the section, Reader Environments and File Manager for details.
- If the file begins with the special FASL signature byte (octal 221), it is assumed to be a compiled file in FASL format, and is processed by the FASL loader. The FASL loader ignores the **LDFLG** argument to **IL:LOAD**, treating all files as though **LDFLG** were **SYSLOAD** (redefinition occurs, is not undoable, and no File Manager information is saved).
- If the file begins with a semicolon, it is assumed to be a pure Common Lisp file. The expressions on the file are read with the standard Common Lisp readtable and in package **USER** (unless a package argument was given to **LOAD**; see below).
- If the file begins with any other character, **LOAD** doesn't know what to do. Currently, it treats the file as a pure Common Lisp file (as if it started with a comment).

Thus, if you prepare Common Lisp text files you should be sure to begin them with a comment so that **LOAD** can tell the file is in Common Lisp syntax.

The function **CL:LOAD** accepts an additional keyword **:PACKAGE**, whose value must be a package object; the function **IL:LOAD** similarly has an optional fourth argument **PACKAGE**. If a package argument is given, then **LOAD** reads Common Lisp text files (type 2 above) with **\*PACKAGE\*** bound to the specified package. In the



case of File Manager files (types 1 and 4), the package argument overrides the package specified in the file's reader environment.

#### (II:17.6-17.8)

The Interlisp functions **LOADFNS**, **LOADFROM**, **LOADVARS** and **LOADCOMP** do not work on FASL files. They do still work on files produced by the old compiler (extension LCOM).

#### (II:17.9)

**FILESLOAD** (also used by the File Manager's **FILES** command) now searches for compiled files by looking for a file by the specified name whose extension is in the list **\*COMPILED-EXTENSIONS\***. The default value for **\*COMPILED-EXTENSIONS\*** in the Lyric release is (DFASL LCOM). It searches the list of extensions in order for each directory on the search path. This means that FASL files are loaded in preference to old-style compiled files.

## Section 17.2 Storing Files

---

The Lyric release contains two different compilers, the Interlisp Compiler that was present in Koto and previous releases, and the new XCL Compiler (see the next section, Chapter 18 Compiler). With more than one compiler available, the question arises as to which compiler will be used by the functions **CLEANUP** and **MAKEFILE**. The default behavior of these functions in Lyric is to always use the new XCL Compiler. This default can be changed, either on a file-by-file basis or system-wide. Most users, however, will have no need to change the default.

When the **C** or **RC** option has been given to **MAKEFILE**, the system first looks for the value of the **FILETYPE** property on the symbol naming the file. For example, for the file "{DSK}<LISPPFILES>MYFILE", the property list of the symbol **MYFILE** would be examined.

The **FILETYPE** property should be either a symbol from the list below or a list containing one of those symbols. The following symbols are allowed and have the given meanings:

- |                      |                                                                                                                                                                                                             |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>:TCOMPL</b>       | Compile this file by calling either <b>TCOMPL</b> or <b>RECOMPILE</b> , depending upon which of the <b>C</b> or <b>RC</b> options was passed to <b>MAKEFILE</b> .                                           |
| <b>:BCOMPL</b>       | Compile this file by calling either <b>BCOMPL</b> or <b>BRECOMPILE</b> , depending upon which of the <b>C</b> or <b>RC</b> options was passed to <b>MAKEFILE</b> . This is equivalent to the Koto behavior. |
| <b>:COMPILE-FILE</b> | Compile this file by calling <b>CL:COMPILE-FILE</b> , regardless of which option was passed to <b>MAKEFILE</b> .                                                                                            |

If no **FILETYPE** property is found, then the function whose name is the value of the variable **\*DEFAULT-CLEANUP-COMPILER\*** is used. The only legal values for this variable are **TCOMPL**, **BCOMPL**, and **CL:COMPILE-FILE**. Initially, **\*DEFAULT-CLEANUP-COMPILER\*** is set to **CL:COMPILE-FILE**.

If you choose to set the **FILETYPE** property of file name, you should take care that the filecoms for that file saves the value of that property on the file. This will ensure that the same compiler

will be used every time the file is loaded. To save the value of the property, you should include a line in the coms like the following:

```
(PROP FILETYPE MYFILE)
```

where MYFILE is the symbol naming your file.

---

## Section 17.8.2 Defining New File Manager Types

---

### (II:17.30)

The File Manager has been extended to allow File Manager types that accept any Lisp object as a name. A consequence of this is that any user-defined type's **HASDEF** function should be prepared to accept objects other than symbols as the *NAME* argument. Names are compared using **EQUAL**.

---

## Definers: A New Facility for Extending the File Manager

---

The Definer facility is provided to make the process of adding a certain common kind of File Manager type easy. All of the new File Manager types in the Lyric release (including **FUNCTIONS**, **VARIABLES**, **STRUCTURES**, etc.) and almost all of the new defining macros (including **CL:DEFUN**, **CL:DEFPARAMETER**, **CL:DEFSTRUCT**, etc.) were themselves created using the Definer facility.

In previous releases, adding new types and commands to the File Manager involved deeply understanding the way in which it worked and defining a number of functions to carry out certain operations on the new type/command. Further, making functions and macros save away definitions of the new type was similarly subtle and generally difficult or complicated to do. With the addition of Common Lisp, it was realized that a large number of new types and commands would be added, all needing essentially the same implementation of the various operations. In addition, many new defining macros were to be added and all of them needed to save definitions.

As an explanation of the Definer facility, we will describe how **VARIABLES** and **CL:DEFPARAMETER** could be added into the system, if they were not already there.

First, a little background about our example. The macro **CL:DEFPARAMETER** is used in Common Lisp to globally declare a given variable to be special and to give it an initial value. (For the purposes of this example, we will ignore the documentation-string given to real **CL:DEFPARAMETER** forms.) The value of a call to the macro should be the name of the variable being defined. An acceptable definition of this macro might appear as follows:

```
(DEFMACRO CL:DEFPARAMETER (SYMBOL EXPRESSION)
 `(PROGN
 (CL:PROCLAIM ' (CL:SPECIAL ,SYMBOL))
 (SETQ ,SYMBOL ,EXPRESSION)
 ' ,SYMBOL))
```

There are some problems with using such a simple definition in the Lisp environment, however. For example, if a call to this macro were typed to the Exec, the File Manager would not be told to notice it. Thus, there would be no convenient way to remember to

add the form to the filecoms of some file and thus to save it away. Also, note that the macro does not pay attention to the **DFNFLG** variable; thus, loading a file containing a **CL:DEFPARAMETER** form would always set the variable to the value of the initial expression, even when **DFNFLG** was set to **ALLPROP**. This could make editing code using this variable difficult.

We will now proceed to fix these problems by getting the Definer facility involved. There are two steps involved in using Definers:

- Unless one of the currently-existing File Manager types is appropriate for definitions using the new macro, a new type must be created. The macro **XCL:DEF-DEFINE-TYPE** is used for this purpose.
- The macro must be defined in such a way that the File Manager can tell that it should notice and save uses of the macro and under which File Manager type the uses should be saved. The macro **XCL:DEFDEFINER** is used for this purpose.

Since we are pretending for the example that the File Manager type **VARIABLES** is not defined, we decide that definitions using **CL:DEFPARAMETER** should not be given any of the already-existing types. We must define a type, therefore, and we decide to call it **VARIABLES**. The following **XCL:DEF-DEFINE-TYPE** form will do the trick:

```
(XCL:DEF-DEFINE-TYPE VARIABLES "Common Lisp
variables")
```

The first argument to **XCL:DEF-DEFINE-TYPE** is the name for the new type. The second argument is a descriptive string, to be used when printing out messages about the type.

With the new type thus created, we can now use **XCL:DEFDEFINER** to redefine the macro. Simply changing the word **DEFMACRO** into **XCL:DEFDEFINER** and adding an argument specifying the new type suffices to change our earlier definition into a use of the Definer facility:

```
(XCL:DEFDEFINER CL:DEFPARAMETER VARIABLES
 (SYMBOL EXPRESSION)
 ` (PROGN
 (CL:PROCLAIM ' (CL:SPECIAL ,SYMBOL))
 (SETQ ,SYMBOL ,EXPRESSION)
 ' ,SYMBOL))
```

(In fact, we could also remove the final **' ,SYMBOL**; **XCL:DEFDEFINER** automatically arranges for the new macro to return the name of the new definition.) Now, if we were to type the form

```
(CL:DEFPARAMETER *FOO* 17)
```

into the Exec and then call the function **FILES?**, we would be presented with something like the following:

```
24> (FILES?)
the Common Lisp variables: *FOO*
...to be dumped. want to say where the above go?
```

As with other File Manager types, our definitions are being kept track of. If we answer Yes to the above question and specify a file

in which to save the definition, a command like the following will be added to the filecoms:

```
(VARIABLES *FOO*)
```

Actually, the output from **FILES?** as shown above is not quite accurate. In reality, we would also be asked about the following:

```
the Common Lisp functions/macros: CL:DEFPARAMETER
the Definition types: VARIABLES
```

The File Manager is also watching for new types and new Definers being created and will let us save those definitions as well. These would be listed in the filecoms as follows:

```
(DEFINE-TYPES VARIABLES)
(FUNCTIONS CL:DEFPARAMETER)
```

All of these definitions are full-fledged File Manager citizens. The functions **GETDEF**, **HASDEF**, **PUTDEF**, **DELDEF**, etc. all work with the new type. We can edit the definition of **\*FOO\*** above simply by specifying the type to the **ED** function:

```
(ED '*FOO*' 'VARIABLES)
```

When we exit the editor, the new definition will be saved and, unless **DNFLG** is set to **PROP** or **ALLPROP**, evaluated.

It is now time to fully describe the macros **XCL:DEF-DEFINE-TYPE** and **XCL:DEFDEFINER**.

**XCL:DEF-DEFINE-TYPE** *NAME DESCRIPTION &KEY :UNDEFINER* [Macro]

Creates a new File Manager type and command with the given *NAME*. The string *DESCRIPTION* will be used to describe the type in printed messages. The new type implements **PUTDEF** operations by evaluating the definition form, **GETDEF** and **HASDEF** by looking up the given name in an internal hash-table, using **EQUAL** as the equality test on names, and **DELDEF** by removing any named definition from the hash-table. If the **:UNDEFINER** argument is provided, it should be the name of a function to be called with the *NAME* argument to any **DELDEF** operations on this type. The **:UNDEFINER** function can perform any other operations necessary to completely delete a definition.

**XCL:DEF-DEFINE-TYPE** forms are File Manager definitions of type **DEFINE-TYPES**.

As an example of the full use of **XCL:DEF-DEFINE-TYPE**, here is the complete definition of the type **VARIABLES** as it exists in the Lyric release:

```
(XCL:DEF-DEFINE-TYPE VARIABLES "Common Lisp variables"
:UNDEFINER UNDOABLY-MAKUNBOUND)
```

The function **UNDOABLY-MAKUNBOUND** is described in Appendix D of these Release Notes.

**XCL:DEFDEFINER** *{NAME | (NAME {OPTION}\*)} TYPE ARG-LIST &BODY BODY* [Macro]

Creates a macro named *NAME*, calls to which are seen as File Manager definitions of type *TYPE*. *TYPE* must be a File Manager type previously defined using **XCL:DEF-DEFINE-TYPE**. *ARG-LIST* and *BODY* are precisely as in **DEFMACRO**. A macro defined using

**XCL:DEFDEFINER** differs from one defined using **DEFMACRO** in the following ways:

- *BODY* will be evaluated if and only if the value of **DFNFLG** is not one of **PROP** or **ALLPROP**.
- The form returned by *BODY* will be evaluated in a context in which the File Manager has been temporarily disabled. This allows Definers to expand into other Definers without the subordinate ones being noticed by the File Manager.
- Calls to Definers return the name of the new definition (as, for example, **CL:DEFUN** and **CL:DEFPARAMETER** are defined to do).
- Calls to Definers are noticed and remembered by the File Manager, saved as a definition of type *TYPE*.
- SEdit- and Interlisp-style comment forms (those with a CAR of IL:\*) are stripped from the macro call before it is passed to *BODY*. (This comment-removal is partially controlled by the value of the variable **\*REMOVE-INTERLISP-COMMENTS\***, described below.)

The following *OPTIONS* are allowed:

( :UNDEFINER *FN* )

If **DELDEF** is called on a name whose definition is a call to this Definer, *FN* will be called with one argument, the name of the definition. This option allows for Definer-specific actions to be taken at **DELDEF** time. This is useful when more than one Definer exists for a given type. *FN* should be a form acceptable as the argument to the **FUNCTION** special form.

( :NAME *NAME-FN* )

By default, the Definer facility assumes that the first argument to any macro defined using **XCL:DEFDEFINER** will be the name under which the definition should be saved. This assumption holds true for almost all Common Lisp defining macros, including **CL:DEFUN**, **CL:DEFMACRO**, **CL:DEFPARAMETER** and **CL:DEFVAR**. It doesn't work, however, for a few other forms, such as **CL:DEFSTRUCT** and **XCL:DEFDEFINER** itself. When defining a macro for which that assumption is false, the **:NAME** option should be used. *NAME-FN* should be a function of one argument, a call to the Definer. It should return the Lisp object naming the given definition (most commonly a symbol, but any Lisp object is permissible). For example, the **:NAME** option in the definitions of **CL:DEFSTRUCT** and **XCL:DEFDEFINER** is as follows:

```
(:NAME (LAMBDA (FORM)
 (LET ((NAME (CADR FORM)))
 (COND ((LITATOM NAME)
 NAME)
 (T (CAR NAME))))))
```

*NAME-FN* should be a form acceptable as the argument to the **FUNCTION** special form (i.e., a symbol naming a function or a LAMBDA-form).

```
(:PROTOTYPE DEFN-FN)
```

When the editor function **ED** is passed a name with no definitions, the user is offered a choice of several ways to create a prototype definition. Those choices are specified with the **:PROTOTYPE** option to **XCL:DEFDEFINER**. **DEFN-FN** should be a function of one argument, the name to be defined using this Definer. **DEFN-FN** should return either NIL, if no definition of that name can be created with this Definer, or a form that, when evalauted, would create a definition of that name. For example, the **:PROTOTYPE** option for **CL:DEFPARAMETER** might look as follows:

```
(:PROTOTYPE (LAMBDA (NAME)
 (AND (LITATOM NAME)
 `(CL:DEFPARAMETER ,NAME "Value"))))
```

An example using all of the features of **XCL:DEFDEFINER** is the definition of **XCL:DEFDEFINER** itself, which begins as follows:

```
(XCL:DEFDEFINER (XCL:DEFDEFINER
 (:UNDEFINER \DELETE-DEFINER)
 (:NAME
 (LAMBDA (FORM)
 (LET ((NAME (CADR FORM)))
 (COND ((LITATOM NAME)
 NAME)
 (T (CAR NAME)))))))
 (:PROTOTYPE
 (LAMBDA (NAME)
 (AND (LITATOM NAME)
 `(XCL:DEFDEFINER ,NAME "Type"
 ("Arg List"
 "Body")))))
 FUNCTIONS
 (NAME-AND-OPTIONS TYPE ARG-LIST &BODY BODY)
 ...)
```

The following variable is used in the process of removing SEdit- and Interlisp-style comments from Definer forms:

**\*REMOVE-INTERLISP-COMMENTS\*** [Variable]

Interlisp-style comments are forms whose **CAR** is the symbol **IL:\***. It is possible for certain lists in Lisp code to begin with **IL:\*** but not be a comment (for example, a **SELECTQ** clause). When such a list is discovered, the value of **\*REMOVE-INTERLISP-COMMENTS\*** is examined. If it is **T**, the list is assumed to be a comment and is removed without comment. If it is **:WARN**, a warning message is printed, saying that a possible comment was not stripped from the code. If **\*REMOVE-INTERLISP-COMMENTS\*** is **NIL**, the list is not removed, but no warning is printed. This variable is initially set to **:WARN**.

**(CL:EVAL-WHEN WHEN COM ... COM )** [File Package Command]

Interprets each of the commands **COM ... COM** as a file package command, but output is wrapped in **CL:EVAL-WHEN**.

#### EXAMPLE:

```
(CL:EVAL-WHEN (CL:EVAL CL:COMPILE)
 (OPTIMIZERS FOO))
```

will cause the following to be written to the file:

```
(CL:EVAL-WHEN (CL:COMPILE)
 (DEFOPTIMIZER FOO <optimizer for FOO>))
```

---

## Chapter 18 Compiler

---

The Lyric release contains two distinct Lisp compilers:

- The Interlisp Compiler, described in detail in Section 18 of the *IRM*,
- The new XCL Compiler, described in the *Common Lisp Implementation Notes*.

The Interlisp Compiler provides compatibility with previous releases of Interlisp-D. It continues to work in very much the same way as it did in Koto; as before, it compiles all of the Interlisp language. The Interlisp Compiler does not, however, compile the Common Lisp language and will not be extended to do so. The Lyric release is the last release to contain the Interlisp Compiler as a component; future releases will have only the new XCL Compiler. The XCL Compiler is designed to handle both Interlisp and Common Lisp.

Several incompatible changes have been made in the compiled object code produced by the Interlisp Compiler. This means that *all user code must be recompiled in Lyric*. Code compiled in Koto or previous releases will not load into Lyric, and code compiled in Lyric will not load into earlier releases. The filename extension for Interlisp compiled files has been changed from DCOM to LCOM in order to minimize possible confusion.

The XCL Compiler writes its output on a new kind of object file, the DFASL file. These files are quite different from the DCOM/LCOM files produced by the Interlisp Compiler. DFASL files are somewhat more compact, much faster to load and can represent a wider range of data objects than was possible in LCOMs.

Interlisp source files from Koto can be compiled using the new XCL compiler. However, some files need to be remade in Lyric before compilation: files containing bitmaps, Interlisp arrays, or the **UGLYVARS** and/or **HORRIBLEVARS** File Manager commands. To compile such a file, first **LOAD** it, then call **MAKEFILE** to write it back out. This action causes the bitmaps and other unusual objects to be written back in a format acceptable to the new compiler.

The default behavior of the File Manager's **CLEANUP** and **MAKEFILE** functions is to use the new XCL Compiler to compile files, rather than the old Interlisp Compiler. To change this behavior, see Section 17.2, Storing Files.

Note that if you call the compiler explicitly, rather than via **CLEANUP** or **MAKEFILE**, you should be careful to specify the correct compiler. The new compiler is invoked by calling **CL:COMPILE-FILE**. If you inadvertently call **BCOMPL** on a file for which **CLEANUP** has routinely been using the new XCL compiler, there are two undesirable consequences: (1) Any Common Lisp

functions on the file will not be compiled (the Interlisp compiler does not recognize **CL:DEFUN**), and (2) the DFASL files produced by earlier calls on the XCL compiler will still be loaded by **FILESLOAD** in preference to the LCOM file produced by **BCOMPL**.

Lisp provides a facility, **XCL:DEFOPTIMIZER**, by which you can advise the compiler about efficient compilation of certain functions and macros. **XCL:DEFOPTIMIZER** works with both the old Interlisp Compiler and the Lyric XCL Compiler. See the *Common Lisp Implementation Notes* for a description of the compiler.

---

### Warning when Loading Compiled Files

**CAUTION:** Files compiled in Medley cannot be loaded back into Lyric. Medley-compiled .LCOM and .DFASL files will produce an error message when loaded into Lyric. (Lyric-compiled .LCOM and .DFASL files can be loaded and run in Medley.) If you need to run a Medley file in Lyric, load the source file and use the Lyric compiler.

---

### Warning with Declarations

**CAUTION:** There is a feature of the BYTECOMPILER that is not supported by either the XCL compiler or SEdit. It is possible to insert a comment at the beginning of your function that looks like

(\* DECLARATIONS: --)

The tail, or -- section, of this comment is taken as a set of local record declarations which are then used by the compiler in that function just as if they had been declared globally. The XCL compiler does not directly support this feature. If the body of the function gets DWIMIFIED for some other reason, the record declarations will happen to be noticed, otherwise they will not be seen and the compiler will signal an error if it can't find an appropriate top-level record definition.

There are two caveats that you should note:

1. The compiler will give error messages "undefined record name ..." for the records that are declared this way, but will generate correct code.
2. SEdit does not recognize such declarations. Thus, if the "Expand" command is used in SEdit, the expansion will not be done with these record declarations in effect. The code that you see in the editor will not be the same code compiled by the BYTECOMPILER.



## Section 18.3 Local Variables and Special Variables

---

### (II:18.5)

The new execs always use the Common Lisp interpreter, causing LET and PROG statements at top level, particularly in a so-called Interlisp exec, to create lexical bindings, rather than deep or "special" bindings. This can be worked around by setting **il:specvars** to T, which will cause the interpreter to create special bindings for all variables. This can also be worked around by wrapping the form to be "interlisp evaluated" in the IL:INTERLISP special form, which causes the Interlisp interpreter to be invoked.

---

## Chapter 19 Masterscope

---

Masterscope is now a Lisp Library Module, not part of the environment.

---

## Chapter 21 CLISP

---

CLISP infix forms do not work under the Common Lisp evaluator; only "clean" CLISP prefix forms are supported. You should run DWIMIFY in Koto on all other CLISP code before attempting to load it in Lyric. The remainder of this note describes the specific limitations on CLISP in Lyric.

There are two broad classes of transformations that DWIM applies to Lisp code:

1. A sort of macro expander that transforms **IF**, **FOR**, **FETCH**, etc. forms into "pure" Lisp code in well-defined ways.
2. A heuristic "corrector" that performs spelling correction and transforms CLISP infix forms such as X+Y into (PLUS X Y), sometimes having to make guesses as to whether X+Y might really have been the name of a variable.

An operational way of distinguishing the two is that DWIMIFY applied to code of type (1) makes no alterations in the code, whereas for code of type (2) it physically changes the form. Another difference is that code of type (2) must be dwimified before it can be compiled (user typically sets **DWIMIFYCOMPFLG** to **T**), whereas the compiler is able to treat code of type (1) as a special kind of macro.

Broadly speaking, code of type (2) is no longer fully supported. In particular, DWIM is invoked only when the code is encountered by the Interlisp evaluator. This means code typed to an "Old Interlisp" Executive, and code inside of an interpreted Interlisp function. Furthermore, some CLISP infix forms no longer DWIMIFY correctly. It is likely that CLISP infix will not be supported at all in future releases.

Expressions typed to the new Executives and inside of Common Lisp functions are run by the Common Lisp evaluator (**CL: EVAL**). As far as this evaluator is concerned, DWIM does not exist, and forms beginning with "CLISP" words (**IF**, **FOR**, **FETCH**, etc) are macros. These macros perform no DWIM corrections, so all of the subforms must be correct to begin with. This is a change from past releases, where the DWIM expansion of a CLISP word form also had the side effect of transforming any CLISP infix that it might have contained. For example, the macro expansion of

```
(if X then Y+1)
```

treats Y+1 as a variable, rather than as an addition. The correct form is

```
(if X then (PLUS Y 1)),
```

which is the way an explicit call to DWIMIFY would transform it.

If you have CLISP code from Koto you are advised to DWIMIFY the code before attempting to run or compile it in Lyric. Because of differences in the environments, not all CLISP constructs will DWIMIFY correctly in Lyric. In particular, the following do not work reliably, or at all:

1. The list-composing constructs using < and > do not DWIMIFY if the < is unpacked (an isolated symbol), because in Common Lisp, < is a perfectly valid CAR of form. On the other hand, the closing > *must* be unpacked if the last list element is quoted, since, for example, (<A 'B>) reads as (<A (QUOTE B>)).
2. Because of the conventional use of the characters \* and - in Common Lisp names, those characters are only recognized as CLISP operators when they appear unpacked.
3. On the other hand, the operators + and / are the names of special variables in Common Lisp (Steele, p. 325), and hence cause no error when passed unpacked to the evaluator. Thus (LIST X + Y) returns a list of three elements, with no resort to DWIM; however, the parenthesized version (LIST (X + Y)) and the packed version (LIST X+Y) both work.

If you routinely DWIMIFY code, so that no CLISP infix forms (type 2 above) remain on your source files, you may not need to make any changes. However, note that the fact that DWIMIFY of prefix forms implicitly performed infix transformations can hide code that escaped being completely dwimified before being written to a file.

There is a further caution regarding even routinely dwimified code that has not been edited since before Koto. Two uses of the assignment operator ( ) no longer work, if not explicitly dwimified, because their canonical form (the output of DWIMIFY) has changed, and the old form is no longer supported when the form is simply evaluated, macro-expanded, or compiled (with **DWIMIFYCOMPFLG = NIL**):

1. Iterative statement bindings must always be lists. For example, the old form

```
(bind X_2 for Y in --)
```

is now canonically

```
(bind (X _ 2) for Y in --).
```

2. In a WITH expression, assignments must be dwimified to remove `_`. For example, the old form

```
(with MYRECORD MYFIELD _ (FOO))
```

is now canonically

```
(with MYRECORD (SETQ MYFIELD (FOO))).
```

DWIMIFY in Koto correctly made these transformations; however, in some older releases, it did not. Such old code must be explicitly dwimified (which you can do for these cases in Lyric). The errors resulting from failure to do so can be subtle. In particular, the compiler issues no special warning when such code is compiled. For example, in case 1, the macro expansion of the old form treats the symbol `x_2` as a variable to bind, rather than as a binding of the variable `x` with initial value 2. The only hint from the compiler that anything is amiss is likely to be an indication that the variable `x` is used freely but not bound. Case 2 is even subtler: the symbols `MYFIELD` and `_` are treated as symbols to be evaluated; since their values are not used, the compiler optimizes them away, reducing the entire expression to simply `(FOO)`, and there is thus no warning of any sort from the compiler.

---

## Chapter 22 Performance Issues

---

### Section 22.3 Performance Measuring

---

#### (II:22.8)

The Interlisp-D **TIME** function has been withdrawn and replaced with the Common Lisp **TIME** macro (the symbol **TIME** is shared between IL and CL and thus need not be typed with a package prefix). The functionality of the *TIMEN* and *TIMETYP* arguments to the old **TIME** can be had by keywords to the **TIME** macro. The *Common Lisp Implementation Notes* describe the new **TIME** macro and its associated command in more detail.

[This page intentionally left blank]



## VOLUME III—INPUT/OUTPUT

---

### Chapter 24 Streams and Files

---

The Xerox Common Lisp file system supports multiple streams open simultaneously on the same file. This is an *incompatible change* to the semantics of Interlisp-D. You may have to modify old programs if they have not followed the guidelines listed in Sec 24.5 of the *Interlisp-D Reference Manual*. Some of the implications of this change for Interlisp programs are described below.

In prior releases of Interlisp-D, the system treated the *name* of an open file as a synonym for the *stream* open on the file. This meant that only one stream could be open at any time on a given file. In the Lyric release, a file name is no longer a unique name for an open stream. Thus, file names are no longer acceptable as the file/stream argument to any input/output or file system function that operates on an open stream (**READ**, **PRINT**, **CLOSEF**, **COPYBYTES**, etc). The only non-stream values acceptable as stream designators are the symbols **NIL** and **T**, designating the primary and terminal input/output streams. An attempt to use a litatom, even a "full file name," as a stream designator signals the error "LITATOM 'streams' no longer supported." Strings no longer designate an input stream whose source is the string itself—programs should call **OPENSTRINGSTREAM** instead, or use the comparable Common Lisp forms, such as **CL:WITH-INPUT-FROM-STRING**.

The functions **OPENFILE** and **OPENSTREAM** are now synonymous—both return a stream instead of a "full file name." The functions **INPUT** and **OUTPUT** also return streams. One exception to this is that **INPUT** and **OUTPUT** return **T** in the case where the primary input or output stream was previously directed to the terminal. However, this special behavior is for the Lyric release only; we recommend that you convert old code that depended on testing (**EQ (OUTPUT) T**). Note that the values of the variables **\*STANDARD-INPUT\*** and **\*STANDARD-OUTPUT\*** are always streams, even if they are directed to the terminal.

The function **FULLNAME** can be used to obtain the name of a stream. For your convenience, the print syntax of streams now includes the name of the stream (if to a file) and its access (input, output, etc.). Functions, such as **UNPACKFILENAME**, that manipulate file names generally accept a stream as well, extracting the name of the file from the stream.

**INFILEP** still returns a full file name, as it is merely recognizing a file, not opening a stream to it. Programmers should be wary of code that subsequently tries to use the value of **INFILEP** as a stream argument. And, of course, the **FILENAME** argument to **OPENSTREAM** is still a name (a symbol or string), not a stream. **OPENSTREAM** also accepts a Common Lisp pathname as its **FILENAME** argument.

The function **CLOSEALL** is no longer implemented. The function **OPENP** returns **NIL** when passed a file name (or anything else but an open stream). However, for the Lyric release, (**OPENP NIL**) still returns a list of all streams open to files.

The functions **GETFILEINFO** and **SETFILEINFO** can still be given either an open stream or a file name. However, in the latter case, the request refers to the file, not to any stream open on the file. Thus, requesting the value of the attribute **LENGTH** for a file name may return a different value than requesting the value of the attribute **LENGTH** for a stream currently open on the file. **GETFILEINFO** returns **NIL** if given a file name and an attribute that only makes sense for streams (e.g., **ACCESS**, **ENDOFSTREAMOP**).

There is no difference between Common Lisp and Interlisp streams. A stream opened by an Interlisp function can be passed as argument to a Common Lisp input/output function, and vice versa.

Even though multiple streams per file are supported, the streams must still obey consistent access rules. That is, if a stream is open for output, no other streams on that file can be opened. It is not possible to **RENAMEFILE** or **DELFILE** a file that has *any* open stream on it.

The RS-232 or TTY ports are inherently single-user devices (rather than real files) thus, multiple streams cannot be open simultaneously on RS-232 or TTY.

---

## Section 24.15 Deleting, Copying, and Renaming Files

---

### (III:24.15)

The support of multiple streams per file now makes it possible to use **COPYFILE** without worrying about there being other readers of the file, in particular even when there is already a stream open on the file for sequential-only access (a case that failed in prior releases). Of course, **COPYFILE** cannot be used if the file already has an *output* stream open.

---

## Chapter 25 Input/Output Functions

---

---

### Variables Affecting Input/Output

---

There are several implicit parameters that affect the behavior of the input/output functions: the numeric print base, the primary output file, etc. In Common Lisp, these parameters are controlled by binding special variables. In Interlisp they are controlled by a functional interface; e.g., an output expression is wrapped in (RESETFORM (RADIX 8) --) to cause numbers to be printed in octal.

Where the input/output parameters in Common Lisp and Interlisp have essentially the same semantics, they have been integrated in

Lisp. That is, binding the Common Lisp special variable and calling the Interlisp function are equivalent operations, and they affect both Interlisp and Common Lisp input/output. However, it is considerably more efficient to bind a special variable than to call a function in a **RESETFORM** expression. In addition, binding a variable has only a local effect, whereas calling a function to side-effect the input/output parameters can also affect other processes. Thus, you are encouraged to use special variable binding to change parameters formerly changed via functional interface.

All of these variables are accessible in both the Common Lisp and Interlisp packages, so no package qualifier is required when typing them.

These parameters are as follows:

**\*PRINT-BASE\* vs RADIX**

Binding **\*PRINT-BASE\*** to an integer  $n$  from 2 to 36 tells the printing functions to print numbers in base  $n$ . This is equivalent to (**RADIX**  $n$ ). Note: this variable should not be confused with **\*PRINT-RADIX\***, another Common Lisp variable that controls whether Common Lisp functions include radix specifiers when printing numbers.

**\*STANDARD-INPUT\* vs INPUT**

Binding **\*STANDARD-INPUT\*** to an input stream specifies the stream from which to read when an input function's stream argument is **NIL** or omitted. Evaluating **\*STANDARD-INPUT\*** is the same as evaluating (**INPUT**), except that (**INPUT**) returns **T** if the primary input for the process is the same as the terminal input stream (this compatibility feature is for the Lyric release only).

**\*STANDARD-OUTPUT\* vs OUTPUT**

Binding **\*STANDARD-OUTPUT\*** to an output stream specifies the stream to which to print when an output function's stream argument is **NIL** or omitted. Evaluating **\*STANDARD-OUTPUT\*** is the same as evaluating (**OUTPUT**) except that (**OUTPUT**) returns **T** if the primary output for the process is the same as the terminal output stream (this compatibility feature is for the Lyric release only).

**\*PRINT-LEVEL\* & \*PRINT-LENGTH\*  
vs PRINTLEVEL**

Binding **\*PRINT-LEVEL\*** to a positive integer  $a$  and **\*PRINT-LENGTH\*** to a positive integer  $d$  is equivalent to calling (**PRINTLEVEL**  $a$   $d$ ). Binding either variable to **NIL** is equivalent to specifying a value of -1 for the corresponding argument to **PRINTLEVEL**, i.e., it specifies infinite depth or length. Note that in Interlisp, print level is "triangular"—the print length decreases as the depth increases. In Common Lisp, the two are independent. Thus, although print level for both Interlisp and Common Lisp is controlled by a common pair of variables, the Interlisp and Common Lisp print functions interpret them (specifically **\*PRINT-LENGTH\***) slightly differently. In addition, Interlisp observes print level only when printing to the terminal, whereas Common Lisp observes it on all output.

**\*READTABLE\* vs SETREADTABLE**

Binding **\*READTABLE\*** to a readtable specifies the readtable to use in any input/output function with a readtable argument omitted or specified as **NIL**. Evaluating **\*READTABLE\*** is the same as evaluating (**GETREADTABLE**). There is no longer a "NIL" or "T" readtable in Interlisp. See the discussion of readtables for more details.



Although the binding style is to be preferred to the **RESETFORM** expression, one difference should be noted with respect to error checking. In a form such as

```
(RESETFORM (RADIX n)
 some-printing-code)
```

the value of *n* is checked immediately for validity, and an error is signalled if *n* is not an integer between 2 and 36. However, in

```
(LET ((*PRINT-BASE* n))
 some-printing-code)
```

there is no error checking at the time of the binding; rather, an error will not be signalled until the code attempts to print a number.

Similarly, the values of **\*STANDARD-INPUT\*** and **\*STANDARD-OUTPUT\*** must be actual streams, not the values that print functions coerce to streams, such as **NIL**, **T** or window objects.

---

## Integration of Common Lisp and Interlisp Input/output Functions

---

Common Lisp and Interlisp have slightly different rules for reading and printing, regarding such things as escape characters, case sensitivity and number format. Each has two kinds of printing function, an escaped version (intended for reading back in) and an unescaped version. In order that Common Lisp and Interlisp programs can more freely intermix, Xerox Lisp isolates most of the reading/printing differences in the readtables used by both languages, rather than in the functions themselves. The exact rules have been chosen as a reasonable compromise between backward compatibility with Interlisp and integration with Common Lisp. This section outlines the details of this integration.

In what follows, the phrase "the readtable" generally refers to the readtable in force for the read or print operation being discussed. Specifically, this means an explicit readtable (other than **NIL** or **T**) passed as readtable argument to an Interlisp function, or else the current binding of **\*READTABLE\***. See the section on readtables for more details.

---

## Section 25.2 Input Functions

---

The functions **IL:READ** and **CL:READ**, given the same readtable, interpret an input in exactly the same way. That is, the functions obey Common Lisp syntax rules when given a Common Lisp readtable, and Interlisp syntax when given an Interlisp readtable. Thus, the principal difference between the two is in the functionality provided by **CL:READ**'s extra arguments: end of file handling and the ability to specify that the read is recursive, which is mostly important when reading input containing circular structure references (the **##** and **#=** macros). See *Common Lisp, the Language* for details of **CL:READ**'s optional arguments.

There is one further difference between **IL:READ** and **CL:READ**, in the handling of the terminating character. If the read terminates on a white space character, **CL:READ** consumes the character, while **IL:READ** leaves the character in the buffer, to be read by the next input operation. Thus, **IL:READ** is equivalent to **CL:READ-PRESERVING-WHITESPACE**. This difference is so that Interlisp

code that calls (**READC**) following a (**READ**) of a symbol will behave consistently between Koto and Lyric.

The Interlisp function **SKREAD** now defaults its readtable argument to the current readtable, viz., the value of **\*READTABLE\***, rather than **FILERDTBL**. This makes it consistent with all the other input functions, and is usually the correct thing, especially with the new reader environments used by the File Manager, but it is an incompatible change from Koto. **SKREAD** is also now implemented using Common Lisp's **\*READ-SUPPRESS\*** mechanism, which means that, unlike in Koto, it does something reasonable when it encounters read macros.

In the Medley release, reading in bitmaps from files is significantly faster.

---

## Section 25.3 Output Functions

---

The discussion here is limited to the four basic printing functions: the unescaped and escaped Interlisp printing functions (**IL:PRIN1**, **IL:PRIN2**) and the corresponding Common Lisp functions (**CL:PRINC**, **CL:PRIN1**). All other print functions ultimately reduce to these. For example, **IL:PRINT** calls **IL:PRIN2**; **CL:FORMAT** with the **~S** directive performs a **CL:PRIN1**.

**IL:PRIN1** is essentially unchanged from previous releases. It uses no readtable at all, so is unaffected by the value of **\*READTABLE\***. It can be thought of as implicitly using the INTERLISP readtable.

Roughly speaking, **IL:PRIN2** and **CL:PRIN1** behave the same when given the same readtable. In particular, they both produce output acceptable to either **READ** function given the same readtable. Their minor differences are listed below.

**CL:PRINC** behaves like **CL:PRIN1**, except that it never prints escape characters or package prefixes. Thus, unlike **IL:PRIN1**, it is affected by the value of **\*READTABLE\***.

For the benefit of user-defined print functions, **IL:PRIN2** and **CL:PRIN1** bind **\*PRINT-ESCAPE\*** to **T**, while **IL:PRIN1** and **CL:PRINC** bind it to **NIL**. Thus, the print function can always examine **\*PRINT-ESCAPE\*** to decide whether it needs to print objects in a way that will read back correctly (Common Lisp user print functions may want to use **CL:WRITE** to pass **\*PRINT-ESCAPE\*** through transparently; Interlisp functions should choose **IL:PRIN2** or **IL:PRIN1** appropriately).

---

## Printing Differences Between **IL:PRIN2** and **CL:PRIN1**

---

There are two respects in which the Interlisp print functions (both **IL:PRIN1** and **IL:PRIN2**) differ from the Common Lisp ones, independent of readtable:

**Line Length.** The Interlisp functions respect the output stream's line length, while the Common Lisp functions all ignore it (they never insert newline characters when output approaches the right margin).

**Print Level.** The Interlisp functions respect the print level variables only when printing to the terminal (unless **PLVLFILEFLG** is true, see the *Interlisp-D Reference Manual*) or when printing with a Common Lisp readtable, whereas the Common Lisp functions respect them on *all* output.

## Internal Printing Functions

---

Interlisp has several functions (e.g., **NCHARS**, **STRINGWIDTH**, **CHCON**, **MKSTRING**) that operate on the "prin1 pname" of an object, or optionally on its "prin2 pname" when given an extra flag and readtable as arguments. These functions are essentially unchanged in Lyric.

In terms of the discussion above, the "prin1 pname" of an object continues to be the characters that would be produced by a call to **IL:PRIN1** at infinite print level and line length, and with **\*PRINT-BASE\*** bound to 10 (unless **PRXFLG** is true, see *Interlisp-d Reference Manual*). The "prin2 pname" of an object is the list of characters that would be produced by a call to **IL:PRIN2** (or **CL:PRIN1**) using the specified readtable (or **\*READTABLE\*** if none is given), again at infinite print level and line length.

Exception: the function **STRINGWIDTH** computes the width of the expression as if it were printed at the current **\*PRINT-LEVEL\*** and **\*PRINT-LENGTH\***.

## Printing Differences between Koto and Lyric

---

The Common Lisp and Interlisp printing functions use the same strategy for escaping characters in symbol names. Because of this, symbols may print differently in Lyric than they did in Koto, for two reasons: the use of the Common Lisp multiple escape character, and the escaping of numeric print names. Although the appearance is different, the functionality is the same—symbols are still printed in a way that allows them to be correctly read.

Roughly speaking, the multiple escape character is used to escape symbol names that would require two or more single escape characters. Thus, for example, a symbol that printed as `%(OH% NO%)` in Koto will print in Lyric as `| (OH NO) |`. However, in the old readtables that lack a multiple escape character (e.g., OLD-INTERLISP-T), the single escapes are still used. Multiple escapes are also used to print a symbol containing lower-case letters when the readtable is case-insensitive, e.g., `|Small|` in a Common Lisp readtable. Keep in mind also that some additional characters are now "special", e.g., colon in all new readtables, semi-colon in Common Lisp. Thus, the typical NS File "full name" will be printed with the multiple escape character.

Since it is now possible to create symbols that have "numeric" print names, such symbols must be printed with suitable escape characters, so that on input they are not misinterpreted as numbers. For example, the symbol whose print name is "1.2E3" is printed as `|1.2E3|`. In read tables lacking a multiple escape character, the single escape character is used instead, e.g., `%(1.2E3%)` in the old Interlisp T readtable. A print name is considered

numeric according to the definition of "potential number" in Common Lisp (p. 341). Even if such a symbol is not readable in the current system as a number, it still needs to be escaped in case it is read into another system that treats it as numeric (either another Common Lisp implementation, or a future implementation of Xerox Lisp). Thus, some old Interlisp symbols now print escaped where they didn't in Koto; e.g., the **PRINTOUT** directive | .P2 | is a potential number.

## Bitmap Syntax

---

Bitmaps are printed in a new syntax in Lyric. When **\*PRINT-ARRAY\*** is **NIL** (the default at top level), a bitmap prints in roughly the same compact form as previously:

```
#<BITMAP @ nn,nnnnnn>
```

If **\*PRINT-ARRAY\*** is **T**, a bitmap prints in a manner that allows it to be read back:

```
#* (Width Height [BitsPerPixel]) XXXXXXXXX...
```

*Width* and *Height* are measured in pixels; *BitsPerPixel* is supplied for bitmaps of other than the default of 1 bit per pixel. Each *X* represents four bits of a row of the bitmap; the characters @ and A through o are used in this encoding. Thus, there are  $4 \lceil \text{Width} * \text{BitsPerPixel} / 16 \rceil$  *X*'s for each row.

**MAKEFILE** binds **\*PRINT-ARRAY\*** to **T** so that bitmaps print readably in files. E.g., if the value of **FOO** is a bitmap, the command (**VAR** **FOO**) dumps something like

```
(RPAQQ FOO #*(10 10)ADSDKJFDKJH...)
```

Note that with this new format, bitmaps are readable even inside a complex list structure. This means you need no longer use the **UGLYVARS** command when dumping a list containing bitmaps if the bitmaps were previously the only "unprintable" part of the list.

## Section 25.8 Readtables

---

### (III:25.34)

The input/output syntaxes of Common Lisp and Interlisp differ in a few significant ways. For example, Common Lisp uses "\" as the escape character, whereas Interlisp uses "%". Common Lisp input is case-insensitive (lower-case letters are read as upper-case), whereas Interlisp is case-sensitive. In Xerox Lisp, these differences are accommodated by having different readtables for the two dialects. Which syntax is used for input or output depends on which readtable is being used (either as an explicit argument to the read/print function or by being the "current" readtable).

Interlisp readtables have been extended to include features of Common Lisp syntax. There is a registry of named readtables to make it easier to choose a readtable. The default Interlisp readtable has been modified to make it look a little closer to Common Lisp.

Also, Lisp has a new mechanism for maintaining read/print consistency. This means that even though Koto files may contain

characters that are now "special", such as colon, you need make no changes to them—the File Manager knows how to load them correctly. See *IRM*, Chapter 17, Reader Environments and File Manager for details of this mechanism.

---

### Differences Between Interlisp and Common Lisp Read Tables

---

When reading or printing, the readtable dictates the syntax rules being followed. As in past releases, the readtable indicates which characters must be escaped when printing a symbol (and **\*PRINT-ESCAPE\*** is true). In addition, in Lyric the readtable specifies such things as which escape character to use (`\` or `%`) and the package delimiter to print on package-qualified symbols. The less obvious rules are detailed below.

**Printing numbers.** Numbers are always printed in the current print base (the value of the variable **\*PRINT-BASE\***, or equivalently the value of **(RADIX)**). Whether to print a radix specifier is determined by the readtable. In Common Lisp, a radix specifier is printed exactly when the value of **\*PRINT-RADIX\*** is true. The radix specifier is a suffix decimal point in base 10, or a prefix using `#` for other bases. In Interlisp, a radix specifier is printed if the base is not 10, **\*PRINT-ESCAPE\*** is true, and the number is not less than the print base. The radix specifier is a suffix `Q` for octal, or a prefix using `#` (or `|` in old Interlisp readtables) for other bases. There is no decimal radix specifier.

**Reading numbers.** In Common Lisp, numbers are read in the current value of **\*READ-BASE\***, and a trailing decimal point is interpreted as a decimal radix specifier. In Interlisp, numbers are always read in base 10, and trailing decimal point denotes a floating-point number.

**Case conversion.** In a case-insensitive readtable (as Common Lisp is), the value of **\*PRINT-CASE\*** controls how upper-case symbols are printed, and lower-case letters in symbols are escaped. In a case-sensitive readtable (as Interlisp is), **\*PRINT-CASE\*** is ignored, so all letters in symbols are printed verbatim. **\*PRINT-CASE\*** is also ignored by **PRIN1**, which implicitly uses an Interlisp readtable.

**Ratios.** The character slash (`/`) is interpreted as the ratio marker in all readtables except old Interlisp readtables (specifically, those whose **COMMONNUMSYNTAX** property is **NIL**). This is so that old files containing symbols with slashes are not misinterpreted as ratios. Thus, the characters "1/2" are read in new readtables as the ratio 1/2, but in old Interlisp readtables as the 3-character symbol `|1/2|` (`|` is the multiple-escape character, see below). Ratios are printed in old Interlisp readtables in the form `|. (/ numerator denominator)`.

**Packages.** Symbols are interned with respect to the current package (the binding of **\*PACKAGE\***) except in old Interlisp readtables (specifically, those whose **USESILPACKAGE** property is **T**), where symbols are read with respect to the INTERLISP package, independent of the binding of **\*PACKAGE\***. Again, this is a backward-compatibility feature: Interlisp had no package system, so programmers were not confronted with the need to read and print in a consistent package environment.

**Print Level elision.** When **\*PRINT-LEVEL\*** or **\*PRINT-LENGTH\*** is exceeded, the printing functions denote elided elements and elided tails by printing "&" and "--" with an Interlisp readable, or "#" and "... " with a Common Lisp readable.

---

### Section 25.8.2 New Readtable Syntax Classes

---

The following new syntax classes are recognized by **GETSYNTAX** and **SETSYNTAX**:

**MULTIPLE-ESCAPE** This character inhibits any special interpretation of all characters (except the single escape character) up until the next occurrence of the multiple escape character. In Common Lisp and in the new Interlisp readtables this character is the vertical bar ("|"). For example, |(a)| is read as the 3-character symbol "(a)"; |x|y|z| is read as the 5 character symbol "x|y|z".

There is no multiple escape character in the old Interlisp readtables.

**PACKAGEDELIM** This character separates a package name from the symbol name in a package-qualified symbol. In Common Lisp and in the new Interlisp readtables this character is colon (":"). In the old Interlisp readtables the package delimiter is control-↑ ("↑↑"); it is not intended to be easily typed, but exists only to have a compatible way to print package-qualified symbols in obsolete readtables. See *Common Lisp, the Language* for details of package specification.

---

### Additional Readtable Properties

---

Read tables have several additional properties in Xerox Lisp. These are accessible via the function **READTABLEPROP**:

**(READTABLEPROP RDTBL PROP NEWVALUE)** [Function]

Returns the current value of the property *PROP* of the readtable *RDTBL*. In addition, if *NEWVALUE* is specified, the property's value is set to *NEWVALUE*. The following properties are recognized:

**NAME** The name of the readtable (a string, case is ignored). The name is used for identification when printing the readtable object itself, and can be given to the function **FIND-READTABLE** to retrieve a particular readtable.

**CASEINSENSITIVE** If true, then unescaped lower-case letters in symbols are read as upper-case when this readtable is in effect. This property is true by default in Common Lisp readtables and false in Interlisp readtables.

**COMMONLISP** If true, then input/output obeys certain Common Lisp rules; otherwise it obeys Interlisp rules. This is described in more detail in the section on reading and printing. Setting this property to true also sets **COMMONNUMSYNTAX** true and **USESILPACKAGE** false.

**COMMONNUMSYNTAX** If true, then the Common Lisp rules for number parsing are followed; otherwise the old Interlisp rules are used. This affects the interpretation of "/" and the floating-point exponent specifiers "d", "f", "l" and "s". It does not affect the interpretation of decimal point and **\*READ-BASE\***, which are controlled by the **COMMONLISP** property. **COMMONNUMSYNTAX** is true for Common Lisp

readtables and the new Interlisp readtables; it is false for old Interlisp readtables.

**USESILPACKAGE**

This is a backward compatibility feature. If **USESILPACKAGE** is true, then the Interlisp input/output functions read and print symbols with respect to the Interlisp package, independent of the current value of **\*PACKAGE\***. This property is true by default for old Interlisp readtables and false for others.

The following properties let the print functions know what characters are being used for certain variable syntax classes so that they can print objects in a way that will read back correctly. Note that it is possible for several characters to have the same syntax on input, but only one of the characters is used for output. Also note that only the three syntax classes **ESCAPE**, **MULTIPLE-ESCAPE** and **PACKAGEDELIM** are parameterized for output; the others (such as **LEFTPAREN** and **STRINGDELIM**) are hardwired—the same character is always used.

**ESCAPECHAR**

This is the character code for the character to use for single escape. Setting this property also gives the designated character the syntax **ESCAPE** in the readtable.

**MULTIPLE-ESCAPECHAR**

This is the character code for the character to use for multiple escape. Setting this property also gives the designated character the syntax **MULTIPLE-ESCAPE** in the readtable.

**PACKAGECHAR**

This is the character code for the package delimiter. Setting this property also gives the designated character the syntax **PACKAGEDELIM** in the readtable.

**(FIND-READTABLE NAME)**

[Function]

Returns the readtable whose name is *NAME*, which should be a symbol or string (case is ignored); returns **NIL** if no such readtable is registered. Readtables are registered by calling **(READTABLEPROP rdtbl 'NAME name)**.

**(COPYREADTABLE RDTBL)**

[Function]

**COPYREADTABLE** has been extended to accept a readtable name as its *RDTBL* argument (the old value **ORIG** could be considered a special case of this). For example, **(COPYREADTABLE "INTERLISP")** returns a copy of the INTERLISP readtable. **COPYREADTABLE** preserves all syntax settings and properties except *NAME*.

## Section 25.8 Predefined Readtables

The following readtables are registered in the Lyric release of Lisp:

**INTERLISP**

This is the "new" Interlisp readtable. It is used by default in the Interlisp Exec and by the File Manager to write new versions of pre-existing source files. It thus replaces the old T readtable, **FILERDTBL**, **CODERDTBL** and **DEDITRDTBL**. It differs from them in the following ways:

**| (vertical bar)**

has syntax **MULTIPLE-ESCAPE** rather than being used as a variant of the Common Lisp dispatching **#** macro character.

|               |                                                                                                                                                |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| #             | is used as the Common Lisp dispatching # macro character. For example, to type a number in hexadecimal, the syntax is #xnnn rather than  xnnn. |
| : (colon)     | has syntax <b>PACKAGEDELIM</b> .                                                                                                               |
| ' (quote)     | reads the next expression as (QUOTE expression).                                                                                               |
| ` (backquote) | are used to read backquoted expressions                                                                                                        |
| , (comma)     |                                                                                                                                                |

In addition, the Common Lisp syntax for numbers is supported (the readtable has property **COMMUNNUMSYNTAX**). For example, the characters "1/2" denote a ratio, not a symbol. Note, however, that trailing decimal point still means floating point, rather than forcing a decimal read base for an integer.

The syntax for quote, backquote, and comma is the same as in OLD-INTERLISP-T, so you will not see any difference when typing to an Interlisp Exec. However, the fact that files are also written in the new INTERLISP readtable means that the prettyprinter is now able to print quoted and backquoted expressions much more attractively on files (and to the display as well).

**LISP** This readtable implements Common Lisp read syntax, exactly as described in *Common Lisp, the Language*.

**XCL** This readtable is the same as LISP, except that the characters with ASCII codes 1 thru 26 have white-space (**SEPRCHAR**) syntax. This readtable is intended for use in File Manager files, so that font information can be encoded on the file.

The following readtables are provided for backward compatibility. They are the same as the corresponding readtables in the Koto release, with the addition of the **USESILPACKAGE** property.

**ORIG** This is the same as the ORIG readtable described in the *Interlisp-D Reference Manual*. When using a readtable produced by (**COPYREADTABLE** 'ORIG), expressions will read and print exactly the same in Koto and Lyric.

**OLD-INTERLISP-FILE** This is the same as the FILERDTBL described in the *Interlisp-D Reference Manual*. This readtable is used to read source files produced in the Koto release. Note that in Lyric, FILERDTBL is no longer used when reading or writing new files; see the section on reader environments.

**OLD-INTERLISP-T** This is the same as the T readtable described in the *Interlisp-D Reference Manual*.

If you wish to change the syntax used by a standard readtable, it is recommended instead that you copy the readtable, give it a distinguished name, and make the change in the new readtable. This will reduce the likelihood that you will try to read another user's files in an incompatible readtable, or that another user will fail reading yours. See chapter 17, Reader Environments and the File Manager, for more details.



## Koto Compatibility Considerations

---

In order to consistently read a data structure that you have previously printed, it is important that **READ** and **PRINT** both use the same readtable and package. Code that calls **READ** or **PRINT** without explicitly specifying a readtable (via the *RD\_TBL* argument or by doing a **SETREADTABLE**) is thus in some danger of reading and printing inconsistently.

### Specifying Readtables and Packages

---

In Koto, the "primary" (NIL) readtable was not significantly different from the other Interlisp readtables, and users tended not to make significant modifications to the primary readtable anyway. As a result, it was easy to write code that was not careful about readtables and get away with it. In Lyric, however, there are significant differences among commonly used readtables. Thus, if code using the default readtable called **PRINT** under, say, the Common Lisp Executive and tried to **READ** the expression back while running under an Interlisp Executive, it might very well get inconsistent results.

Lyric also introduces the extra complication of the default package, which is the other important parameter affecting the behavior of **READ** and **PRINT**.

Programmers are thus advised to fix any code that uses **READ** and **PRINT** as a way of storing and retrieving Lisp expressions to be sure to specify a readtable and package environment. For new code in Lyric, this can be done by binding the special variables **\*READTABLE\*** and **\*PACKAGE\***. If it is necessary to write code that works in both Koto and Lyric, the programmer should pass an explicit readtable to all **READ** and **PRINT** functions, or set the primary readtable using (**RESETFORM** (**SETREADTABLE** *rdtbl*) --). If the readtable chosen is either *FILERDTBL* or one derived as a copy of *ORIG*, then **READ** and **PRINT** will automatically use the *INTERLISP* package in Lyric, thereby avoiding any need to specify a binding for **\*PACKAGE\***.

### The T Readtable

---

An additional possible incompatibility exists with regard to the Koto T readtable: The T readtable was "the readtable used when reading from the terminal". In Lyric, the T readtable is synonymous with NIL, and all Executives bind **\*READTABLE\*** to the appropriate value for the Exec. This is unlikely to be a major source of incompatibility, as few programs depend on printing something in the T readtable in a way that needs to read back consistently.

### PQUOTE Printed Files

---

In Lyric, the prettyprinter automatically prints quoted and backquoted expressions attractively. Hence, the *PQUOTE* Lispusers module is now obsolete. However, if you have written files in the past with the *PQUOTE* module loaded into your environment, you need to do the following in Lyric in order to load those files:

```
(SETSYNTAX (CHARCODE '"') '(MACRO FIRST READQUOTE)
FILERDTBL)
```

You can then load the old files. New files produced in Lyric by **MAKEFILE** will automatically be loadable, so you need only perform the **SETSYNTAX** change as long as you still have old files written with PQUOTE. Remember, of course, that as long as the **SETSYNTAX** is in effect (as with the old PQUOTE module), if you read old files that were written without PQUOTE you may read them incorrectly.

#### Back-Quote Facility

---

The back-quote facility now fully conforms with *Common Lisp the Language*. This means some cases of nested back-quote now work correctly. Back-quote forms are also more attractively displayed by the prettyprinter. Users should beware, however, that the back-quote facility does not attempt to create fresh list structures unless it is necessary to do so. Thus for example,

'(1 2 3)

is equivalent to

'(1 2 3)

not

(LIST 1 2 3)

If you need to avoid sharing structure you should explicitly use **LIST**, or **COPY** the output of the back-quote form.

---

## Chapter 28 Windows and Menus

---

### Section 28.5.1 Menu Fields

---

(III:28.38)

With the Medley release, multi-column menus can have rollout submenus.

[This page intentionally left blank]

## 4. CHANGES TO INTERLISP-D IN LYRIC/MEDLEY

---

**NOTE:** Chapter 4 is organized to correspond to the original *Interlisp-D Reference Manual*, and explains changes that have occurred in Interlisp-D with the Lyric and Medley releases. To make it easy to use this chapter with the *IRM*, information is organized by *IRM* volume and section numbers. Section headings from the *IRM* are maintained to aid in cross-referencing.

Lyric information as well as Medley release enhancements are included. Medley additions are indicated with revision bars in the right margin.

### VOLUME I—LANGUAGE

---

## Chapter 3 Lists

---

### Section 3.2 Building Lists From Left To Right

---

#### (I:3.7)

The functions **DOCOLLECT** and **ENDCOLLECT** are no longer supported.

#### (I:3.8)

The description of the **ADDTOSCRATCHLIST** function has been revised to read:

(ADDTOSCRATCHLIST VALUE) [Function]

For use inside a **SCRATCHLIST** form. *VALUE* is added on to the end of the value being collected by **SCRATCHLIST**. When the **SCRATCHLIST** returns, its value is a list containing all of the things that **ADDTOSCRATCHLIST** has added.

### Section 3.10 Sorting Lists

---

#### (I:3.17)

(SORT DATE COMPAREFN) [Function]

There is no safe interrupt to **SORT**—if you abort a call to **SORT** by *any* means the possibility exists for losing elements from the list being sorted.

---

## Chapter 6 Hash Arrays

---

(I:6.1)

|                   |                                                 |            |
|-------------------|-------------------------------------------------|------------|
| (HASHARRAY        | MINKEYS OVERFLOW HASHBITSFN EQUIVFN RECLAIMABLE |            |
| REHASH-THRESHOLD) |                                                 | [Function] |

---

The function **HASHARRAY** has two new optional arguments, *RECLAIMABLE* and *REHASH-THRESHOLD*. If *RECLAIMABLE* is true, then entries in the hash table are considered "reclaimable" in the sense that the system is permitted to remove any key and its associated value from the hash table at any time. In practice, the contract is less severe: the system only removes keys when a hash table fills and is about to be rehashed, and then it only removes keys whose reference count is one, and to which there are thus no pointers outstanding except possibly from the stack (local variables). This is useful for hash tables that serve to cache information about Lisp objects to avoid recomputation; for example, the system hash table **CLISPARRAY** is now reclaimable. Discarding keys keeps the table from necessarily needing to grow, and potentially allows the storage consumed by both the key and value to be reclaimed.

---

### Section 6.1 Hash Overflow

---

(I:6.3)

You should note changes to the wording of two of the possibilities for the overflow method:

The first sentence for **NIL** should read: The array is automatically enlarged by *at least* a factor of 1.5 every time it overflows.

The explanation for "a positive integer N" should read: The array is enlarged to include *at least* N more slots than it currently has.

---

## Chapter 7 Integer Arithmetic

---

(I:7.5)

The variables **MIN.INTEGER** and **MAX.INTEGER** have been removed from the *Interlisp-D Reference Manual*. Therefore, calling **(MIN)** and **(MAX)** is an error.

(I:7.7)

|          |  |            |
|----------|--|------------|
| (FIXR N) |  | [Function] |
|----------|--|------------|

---

When N is exactly half way between two integers, **FIXR** rounds it to the even number. For example **(FIXR 1.5)**  $\Rightarrow$  2 and **(FIXR 2.5)**  $\Rightarrow$  2.

---

## Section 7.3 Logical Arithmetic Functions

---

The function **INTEGERLENGTH** does *not* coerce floating point numbers to integers; rather, it signals an error, "Arg not Integer". (This was true in Koto as well.)

---

## Section 7.5 Other Arithmetic Functions

---

### (I:7.13)

The algorithms for **SIN**, **COS**, and other trigometric functions have been tuned and are now accurate to at least six significant figures.

---

## Chapter 8 Record Package

---

### (I:8.11)

When using **BLOCKRECORD**, it is an error to try to declare a record with a zero-length field. Previously, the system would go into an infinite loop. In the Medley release, the system will now detect this and signal an error.

---

## Chapter 9 Conditionals and Iterative Statements

---



---

### Section 9.2 Equality Predicates

---

#### (I:9.3)

**(EQUALALL X Y)**

[Function]

Add the following NOTE to the **EQUALALL** function:

Note: In general, **EQUALALL** descends all the way into all datatypes, both those defined by the user and those built into the system. If you have a data structure with fonts and pointers to windows, **EQUALALL** will descend into those also. If the data structures are circular, as windows are, **EQUALALL** can cause a Stack Overflow error.

---

### Section 9.8.3 Condition I.s. oprs

---

**UNTIL N (N a number)**

[I.S. Operator]

**REPEATUNTIL N (N a number)**

[I.S. Operator]

These descriptions were included in the *Interlisp-D Reference Manual* in error and should be removed. **UNTIL** and **REPEATUNTIL** work *only* with predicate expressions, not numbers.

---

## Chapter 10 Function Definition, Manipulation , and Evaluation

---

### Section 10.2 Defining Functions

---

(I:10.11)

In the definition of the **MOVD** function, the sentence "**COPYDEF** is a higher-level function that only moves expr definitions, but..." should be revised to read:

**COPYDEF** is a higher-level function that not only moves expr definitions, but also works for variables, records, etc.

### Section 10.5 Functional Arguments

---

(I:10.19)

**FUNARG** functionality (non-**NIL** second argument to **FUNCTION**) has been withdrawn. Most of the uses for Interlisp **FUNARG**'s are better written using the lexical closure functionality of Common Lisp.

### Section 10.6.2 Interpreting Macros

---

The variables **SHOULD\_COMPILE\_MACRO\_ATOMS** and **UNSAFEMACRO\_ATOMS** no longer exist.

---

## Chapter 11 Variable Bindings and the Interlisp Stack

---

(II:11.2)

In Lisp there is a fixed amount of space allocated for the stack. When this space is exhausted, the **STACK OVERFLOW** error occurs. However, if the system waited until the stack were *really* exhausted, there wouldn't be room to run the debugger. Thus, a portion of the stack space is reserved; when the stack intrudes into the reserved area, it causes a stack overflow interrupt, and subsequently a call to the debugger.

In order not to get a **STACK OVERFLOW** error while inside the debugger, this intrusion into the reserved area is only noted once. If the reserved area is exhausted, then a "hard" stack overflow occurs (a 9319 MP halt), from which the only recourse is a hard reset via **STOP** (or Ctrl-D from TeleRaid). Following a hard reset, the stack is cleared, stack overflow detection is reenabled, and all processes are restarted.

The implications of this are that you should not attempt any deep computations while inside the debugger for a stack overflow error, and you should call (**HARDRESET**) as soon as possible in order that subsequent stack overflows can again be caught in the debugger before they advance to the MP halt. In order to make this more convenient, the system automatically calls (**HARDRESET**) if you exit the debugger via the **^** or **OK** commands, or abort with a Ctrl-D. The only way to exit the debugger without having a

**(HARDRESET)** occur is by using the **RETURN** command. You can disable this feature by setting **AUTOHARDRESETFLG** to **NIL**, in which case you must be sure to perform the **(HARDRESET)** yourself if you want the next stack overflow to be detected early enough to enter the debugger.

---

### Section 11.2.1 Searching the Stack

---

(STKPOS FRAMENAME N POS OLDPOS) [Function]

(STKPOS 'STKPOS) does not cause an error; it merely returns NIL. (This was true in Koto as well.) It is still not permissible to create a pointer to the active frame; however, **STKPOS** never attempts to, as it starts searching for the specified frame by looking first at its caller.

---

### Section 11.2.2 Variable Bindings in Stack Frames

---

(I:11.7)

(STKARG N POS —) [Function]

(STKNARGS POS —) [Function]

The functions **STKARG** and **STKNARGS** will now return the number of arguments supplied to a Lambda Nospread when there is a break. The **?=** command will show all the arguments.

(SETSTKARGNAME N POS NAME) [Function]

The function **SETSTKARGNAME** does not work for interpreted frames.

---

### Section 11.2.5 Releasing and Reusing Stack Pointers

---

(CLEARSTK FLG) [Function]

(CLEARSTK NIL) is a no-op—the ability to clear all stack pointers is inconsistent with the modularity implicit in a multi-processing environment.

CLEARSTKLST [Variable]

NOCLEARSTKLST [Variable]

The variables **CLEARSTKLST** and **NOCLEARSTKLST** are no longer used. (More precisely, they are used only by the Old Interlisp Executive, which means that programs can no longer depend on them.)

---

### Section 11.2.7 Other Stack Functions

---

(II:11.13)

In the **REALFRAMEP** function, the **INTERPFLG** argument description has been corrected to read:

If **INTERPFLG=T** returns **T** if **POS** is not a dummy frame. For example, if **(STKNAME POS)=COND**, **(REALFRAMEP POS)** is **NIL**, but **(REALFRAMEP POS T)** is **T**.



---

## Chapter 12 Miscellaneous

---

### Section 12.2 Idle Mode

---

The following properties in **IDLE.PROFILE** are new or have meanings different from the documentation in the *Interlisp-D Reference Manual*:

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>ALLOWED.LOGINS</b> | <p>The authentication aspects of this property have been separated into the <b>AUTHENTICATE</b> property. The value of this property now speaks specifically to who is allowed to exit idle mode: If the value is <b>NIL</b> (or any other non-list), no login at all is required to exit Idle mode. Otherwise, the value is a list composed of any of the following:</p> <ul style="list-style-type: none"> <li><b>*</b> Require login, but let anyone exit idle mode. This will overwrite the previous user's name and password each time idle mode is exited.</li> <li><b>T</b> Let the previous user (as determined by <b>USERNAME</b>) exit idle mode. If the user name has not been set, this is equivalent to <b>*</b>.</li> </ul> <p>A user name Let this specific user exit idle mode.</p> <p>A group name Allow any members of this group (an NS Clearinghouse group name) to exit idle mode.</p> <p>The initial value for <b>ALLOWED.LOGINS</b> is <b>(T *)</b>, i.e., anyone is allowed to exit idle mode.</p> |
| <b>AUTHENTICATE</b>   | <p>The value of this property determines what mechanism is used to check passwords. If <b>T</b>, use the NS authentication protocol (requires the presence of an NS Authentication server accessible via the network). If <b>NIL</b>, do not check the password at all—accept any password. This is only particularly useful if <b>ALLOWED.LOGINS</b> contains <b>*</b>.</p> <p>The initial value of <b>AUTHENTICATE</b> is <b>T</b>.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <b>FORGET</b>         | <p>If this is the symbol <b>FIRST</b>, the user's password will be erased when idle mode is entered. Otherwise, this property is relevant only when <b>ALLOWED.LOGINS</b> is <b>NIL</b> (if <b>ALLOWED.LOGINS</b> is a list, then some sort of login is required, which will have the effect of erasing any previous login): If <b>FORGET</b> is non-<b>NIL</b>, the user's password will be erased when idle mode is exited. Initial value is <b>T</b> (erase password on exit).</p> <p>Note: If the password is erased on <i>entry</i> to Idle mode (value <b>FIRST</b>), any program left running when idle mode is entered may fail if it tries doing anything requiring passwords (such as accessing file servers).</p>                                                                                                                                                                                                                                                                                               |
| <b>LOGIN.TIMEOUT</b>  | <p>Value is a number indicating how many seconds Idle's prompt for a login should remain up before timing it out and resuming Idle mode. Initial value is 30. This feature avoids the problem of having an Idle machine "freeze up" indefinitely (stop running the idle pattern) just because someone brushed against the keyboard.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>RESETVARS</b>      | <p>This property is no longer used; rather, the value of the global variable <b>IDLE.RESETVARS</b> is used instead.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

**SUSPEND.PROCESS.NAMES**

This property is no longer used; rather, the value of the global variable **IDLE.SUSPEND.PROCESS.NAMES** is used instead.

## Section 12.3 Saving Virtual Memory State

---

### **AROUNDEXITFNS**

[Variable]

This variable provides a way to "advise" the system on cleanup and restoration activities to perform around **LOGOUT**, **SYSOUT**, **MAKESYS** and **SAVEVM**; it subsumes the functionality of **BEFORESYSOUTFORMS**, **AFTERLOGOUTFORMS**, etc. Its value is a list of functions (names) to call around every "exit" of the system. Each function is called with one argument, a symbol indicating which particular event is occurring:

**BEFORELOGOUT**

The system is about to perform a **LOGOUT**. The event function might want to save state, notify a network connection that it is about to go away, etc.

**BEFORESYSOUT**  
**BEFOREMAKESYS**  
**BEFORESAVEVM**

The system is about to perform a **SYSOUT**, **MAKESYS**, or **SAVEVM**. Often these three events are treated equivalently; however, sometimes the distinction is interesting. For example, a program might want to perform a much more extensive tidying-up before **MAKESYS** than if it is merely doing a routine **SAVEVM**.

**AFTERLOGOUT**  
**AFTERSYSOUT**  
**AFTERMAKESYS**  
**AFTERSAVEVM**

The system is starting up a virtual memory image that was saved by performing a **LOGOUT**, **SYSOUT**, etc. Ordinarily, the event function should treat all of these the same—in all four cases, some arbitrary amount of time has passed, remote files may have come and gone, a different user may be logged in, or the virtual memory image might even be running on a different workstation.

**AFTERDOSYSOUT**  
**AFTERDOMAKESYS**  
**AFTERDOSAVEVM**

The system is continuing in the same virtual memory image following a **SYSOUT**, **MAKESYS**, or **SAVEVM** (as opposed to having just booted the same virtual memory image). Ordinarily, these events are uninteresting; they exist solely so that actions taken by the **BEFORExxx** events can be compensated for after the event. For example, if the before event cleared a cache, the after event might initiate refilling it. There is, of course, no event **AFTERDOLOGOUT**, as **LOGOUT** does not "continue".

## Section 12.4 System Version Information

---

**(l:12.13)**

In the description of the **MACHINETYPE** function, add another machine, the **DOVE** (for the Xerox 1186).

## VOLUME II—ENVIRONMENT

---

### Chapter 13 Interlisp Executive

---

(I:23.37)

(**READLINE** *RD\_TBL* — —)

[Function]

The *Interlisp-D Reference Manual* states:

The description on p 13.37 of **READLINE**'s behavior when one or more spaces precede the carriage return applies only when **LISPXREADFN** is **READ**. **LISPXREADFN** is initially **TTYINREAD**, which ignores spaces before the carriage return, and thus will never prompt you with "... " for an additional line. Also, the new Executive does not use **READLINE** at all, so you will never see this behavior in a new Executive, independent of the setting of **LISPXREADFN**.

---

### Chapter 14 Errors and Breaks

---

#### Section 14.5 Break Window Variables

---

(II:14.15)

Setting the variable **BREAKREGIONSPEC** to **NIL** no longer creates problems if there is a subsequent break.

#### Section 14.8 Catching Errors

---

(II:14.22)

The **Nlambda** functions **ERSETQ** and **NLSETQ** now allow evaluation of an arbitrary number of forms, rather than only one.

(II:14.26)

For Medley, the Interlisp interpreter's handler for **RESETFORM** has been fixed (in Lyric, it worked only from the Common Lisp interpreter, or compiled) .

---

## Chapter 17 File Package

---

Note: The File Package is now known as the File Manager.

### Section 17.8.1 Functions for Manipulating Typed Definitions

---

(II:17.26)

**(HASDEF NAME TYPE SOURCE SPELLFLG)**

[Function]

Clarification: **HASDEF** for type FNS (or NIL) indicates that *NAME* has an editable source definition, not that *NAME* is defined at all. Thus if *NAME* exists on a file for which you have loaded only the compiled version but not the source, **HASDEF** returns **NIL**.

### Section 17.8.2 Defining New File Package Types

---

(II:17.31)

In the **WHENCHANGED** File Package Type Property the *REASON* argument passed to **WHENCHANGED** no longer is **T** or **NIL**. The Note has been revised as follows:

Note: The *REASON* argument passed to **WHENCHANGED** functions is either **CHANGED** or **DEFINED**.

(II:17.32)

The Nospread Function **FILEPKGTYPE** returns a property list rather than an alist.

### Section 17.9.2 Variables

---

(II:17.36)

In the Lyric release, **HORRIBLEVARS** did not preserve common substructures across several variables.

In Lyric, you could not dump an **UGLYVARS** or **HORRIBLEVARS** whose printed representation required more than *ARRAY-TOTAL-SIZE-LIMIT* characters. This is no longer the case with the Medley release.

### Section 17.9.8 Defining New File Package Commands

---

(II:17.47)

The Nospread Function **FILEPKGCOM** returns a property list rather than an alist.

### Section 17.11 Symbolic File Format

---

**(PRETTYDEF PRTTYFNS PRTTYFILE PRTTYCOMS REPRINTFNS SOURCEFILE CHANGES)**

[Function]

**PrettyDEF** accepts only a symbol for its file argument.

In Lyric and Medley, **SYSPRETTYFLG** is ignored in Interlisp exec and does not pretty-print values in the executive.

**(LISPSOURCEFILEP FILE)****[Function]**

**LISPSOURCEFILEP** is more specifically defined to mean that the file is in File Manager format *and* has a file map.

---

### Section 17.11.3 File Maps

---

File maps are no longer stored on the FILEMAP property. See **GET-ENVIRONMENT-AND-FILEMAP** in the section entitled "Programmer's Interface to Reader Environments" in chapter 3.

---

## Chapter 18 Compiler

---

**CAUTION:** Files compiled in Medley cannot be loaded back into Lyric. Medley-compiled .LCOM and .DFASL files will produce an error message when loaded into Lyric. (Lyric-compiled .LCOM and .DFASL files can be loaded and run in Medley.) If you need to run a Medley file in Lyric, load the source file and use the Lyric compiler.

Note that you should not attempt to compile a file containing a function named **STOP**. The format of the .LCOM file produced by **BCOMPL** or **TCOMPL** admits an unfortunate ambiguity in the treatment of the symbol **STOP—LOAD** prefers to treat it as the token signifying the end of the file, rather than as starting the definition of the function **STOP**.

There is no such restriction for the .DFASL files produced by **CL:COMPILE-FILE**.

---

## Chapter 21 CLISP

---

---

### Section 21.8 Miscellaneous Functions and Variables

---

**(CLEARCLISPARRAY NAME — —)****[Function]**

Macro and CLISP expansions are cached in CLISPARRAY, the system's CLISP hash array. When anything changes that would invalidate an expansion, it needs to be removed from the cache. CLEARCLISPARRAY removes from the CLISP hash array any key whose CAR is NAME. The system does this automatically whenever you edit a clisp or macro form, or when you redefine a clisp word or macro definition. However, you may need to call CLEARCLISPARRAY explicitly if you change something in a more subtle way, e.g., you redefine a function used by a macro. If your change invalidates an unknown set of expansions, you might prefer to take the performance penalty of calling (CLRHASH CLISPARRAY) to invalidate the entire cache, just to make sure no incorrect expansions are kept around.

---

## Chapter 22 Performance Issues

---

### Section 22.1 Storage Allocation and Garbage Collection

---

The following should be appended to the description of garbage collection in Interlisp-D:

Another limitation of the reference-counting garbage collector is that the table in which reference counts are maintained is of fixed size. For typical Lisp objects that are pointed to from exactly one place (e.g., the individual conses in a list), no burden is placed on this table, since objects whose reference count is 1 are not explicitly represented in the table. However, large, "rich" data structures, with many interconnections, backward links, cross references, etc, can contribute many entries to the reference count table. For example, if you created a data structure that functioned as a doubly-linked list, such a structure would contribute an entry (reference count 2) for each element.

When the reference count table fills up, the garbage collector can no longer maintain consistent reference counts, so it stops doing so altogether. At this point, a window appears on the screen with the following message, and the debugger is entered:

```
Internal garbage collector tables have overflowed, due
to too many pointers with reference count greater than 1.
*** The garbage collector is now disabled. ***
Save your work and reload as soon as possible.
```

[This message is slightly misleading, in that it should say "count not equal to 1". In the current implementation, the garbage collection of a large pointer array whose elements are not otherwise pointed to can place a special burden on the table, as each element's reference count simultaneously drops to zero and is thus added to the reference count table for the short period before the element is itself reclaimed.]

If you exit the debugger window (e.g., with the RETURN command), your computation can proceed; however, the garbage collector is no longer operating. Thus, your virtual memory will become cluttered with objects no longer accessible, and if you continue for long enough in the same virtual memory image you will eventually fill up the virtual memory backing store and grind to a halt.

---

### Section 22.5 Using Data Types Instead of Records

---

#### (II:22.13)

The note in this section states that "pages for datatypes are allocated one page at a time." The note should read:

Space for datatypes is allocated two pages at a time. Thus, each datatype for which any instances at all have been allocated has at least two pages assigned to it.

## Chapter 23 Processes

---

### Section 23.1 Creating and Destroying Processes

---

(III:23.2)

**ADD.PROCESS** no longer coerces the process name to a symbol. Rather, process names are treated as case-insensitive strings. Thus, you can use strings for process names, and when typing process commands to an exec, you need not worry about getting the alphabetic case correct.

### Section 23.2 Process Control Constructs

---

The Medley release fixes the **PROCESS.EVAL** and **PROCESS.APPLY** functions. In **PROCESS.EVAL** and **PROCESS.APPLY**, with argument **WAITFORRESULT = T**, if the computation in the other process aborts (or the process is killed), then **PROCESS.EVAL** and **PROCESS.APPLY** return **:ABORTED** instead of hanging.

### Section 23.6 Typein and the TTY Process

---

#### **BACKGROUNDFN**

[Variable]

A list of functions to call "in the background". The system runs a process (called "BACKGROUND") whose sole task is to call each of the functions on the list **BACKGROUNDFN** repeatedly. Each element is the name of a function of no arguments. This is a good place to put cheap background tasks that only do something once in a while and hence do not want to spend their own separate process on it. However, note that it is considered good citizenship for a background function with a time-consuming task to spawn a separate process to do it, so that the other background functions are not delayed.

#### **TTYBACKGROUNDFN**

[Variable]

This list is like **BACKGROUNDFN**, but the functions are only called while in a tty input wait. That is, they always run in the tty process, and only when the user is not actively typing. For example, the flashing caret is implemented by a function on this list. Again, functions on this list should spend very little time (much less than a second), or else spawn a separate process.

### Section 23.8 Process Status Window

---

The Medley release modifies the way in which the Process Status Window can be reshaped and refreshed.

The Process Status Window is now created in such a way that reshaping the window reshapes **ONLY** the backtrace window, not the main window.

The process status window now refreshes itself automatically following a KILL command.



## VOLUME III—INPUT/OUTPUT

### Chapter 24 Streams and Files

#### Section 24.7 File Attributes

(GETFILEINFO FILE ATTRIB)

[Function]

NS file servers implement the following additional attributes for **GETFILEINFO** (neither of these attributes is currently settable with **SETFILEINFO**):

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>READER</b>     | The name of the user who last read the file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>PROTECTION</b> | A list specifying the access rights to the file. Each element of the list is of the form ( <i>name nametype . rights</i> ), where <i>name</i> is the name of a user or group or a name pattern, and <i>rights</i> is one or more of the symbols ALL READ WRITE DELETE CREATE MODIFY. For servers running Services release 10.0 or later, <i>nametype</i> is the symbol "--"; in earlier releases it is either INDIVIDUAL or GROUP, to distinguish the type of name. For example, the value ((Jane Jones: -- ALL) (*: -- READ)) means that user Jane Jones has full access to the file, while all members of the default domain only have read access to the file. |

#### Section 24.9 Local Hard Disk Device

**(III:24.22)**

In the Medley release, the {DSK} device now accepts a wider range of characters in file names. Almost any character in char set 0 is acceptable. Previously, if you tried to create a file whose name included, for example, an underscore, you would see a "FILE NOT FOUND" error.

#### Section 24.10 Floppy Disk Device

**(III:24.26)**

As of the Lyric release, CPM-format floppy disks are no longer supported.

#### Section 24.12 Temporary Files and CORE Device

**(III:24.30)**

In Medley, (**GETFILEINFO** xx 'LENGTH) works for both opened and closed **NODIRCORE** streams.

A closed **NODIRCORE** stream can be reopened.

## Section 24.18.1 Pup File Server Protocols

### UNIXFTPFLG

[Variable]

When the Leaf protocol was first implemented for the Vax Unix operating system, its use was inconsistent with the operation of the Pup FTP server on the same host: the Leaf server supported versions, but the Ftp server knew only about the native, versionless file system. Thus, Lisp could not use the two protocols interchangeably. For example, if it used Ftp to write a file FOO, the Ftp server would, in versionless style, overwrite the versionless file FOO, rather than create a new version FOO;6 to supersede the highest version FOO;5 created by the Leaf server.

Lisp thus makes the conservative assumption that the Ftp server is unusable for anything other than directory enumeration on a host of type UNIX. This is unfortunate, since it is often the case that Ftp is more efficiently implemented than Leaf, since one need only tune the performance of sequential access.

More recent versions of the Unix Pup software have a Leaf and Ftp server more in agreement with each other. Setting **UNIXFTPFLG** to true (it is initially NIL) informs Lisp that all the Unix servers accessible on your internetwork that possess Ftp servers are safe to use in parallel with their Leaf servers.

## Section 24.18.1 and 24.18.2 Use of BREAKCONNECTION with File Servers

### (III:24.37)

In Medley, the function **BREAKCONNECTION** can be used equally well with NS servers and Leaf servers. Formerly, it only worked on Leaf servers, and there was a separate function (BREAK.NSFILING.CONNECTION *HOST*) to handle NS servers.

### (BREAKCONNECTION *HOST FAST*)

[Function]

Breaks the file server connection to *HOST*. If *HOST* = T, breaks connections to all file servers that understand the **BREAKCONNECTION** method (currently Leaf and NS). **BREAKCONNECTION** returns the server name, or if *HOST* = T, returns a list of all hosts that responded to the **BREAKCONNECTION** request.

This function may be useful if Lisp and the server disagree about what files are open, or if the Lisp system is caching something that you do not want it to; e.g., if you get a file busy error from another workstation for a file that you may have touched on this workstation.

The behavior of **BREAKCONNECTION** is server-specific. On an NS server, **BREAKCONNECTION** releases any locks that Lisp may have on recently-accessed files, including those for open files, but does not close any files from Lisp's point of view--any subsequent access to an open file will quietly reestablish the connection. Most NS servers have a short timeout on the order of 10 minutes after which an implicit **BREAKCONNECTION** occurs if you have no files open.

On a Leaf server, **BREAKCONNECTION** first closes any files. If the argument *FAST* is true, it marks the files closed without attempting to close them cleanly. Leaf connections ordinarily do not timeout if any files at all are open.

### Section 24.18.2 NS File Server Protocols

---

(III:24.37)

Medley incorporates the random access capability on NS servers provided by the NSRANDOM LispUsers module in Lyric.

The Medley release also supports NS file names containing characters other than character set 0 (e.g., Greek characters).

### Section 24.18.3 Operating System Designations

---

DEFAULT.OSTYPE

[Variable]

If a host's name is not found in **NETWORKOSTYPES**, its operating system type is assumed to be the value of **DEFAULT.OSTYPE**. This variable may be of use to sites with many servers all of the same type. Its default value (IFS) is, unfortunately, inappropriate for most sites. It is recommended you set **DEFAULT.OSTYPE** in the initialization file that lives on the local disk (*not* in an init file on a file server, since Lisp needs to know the operating system type before talking to the server).

---

## Chapter 25 Input/Output Functions

---

### Section 25.2 Input Functions

---

(LASTC FILE)

[Function]

The function **LASTC** can return an incorrect result when called immediately following a **PEEKC** on a file that contains run-coded NS characters.

### Section 25.3.2 Printing Numbers

---

(III:25.15)

In the **PRINTNUM** function, the **FLOAT** format option (**FLOAT 7 2 NIL T**) is illegal; change the option to (**FLOAT 7 2 NIL 0**).

### Section 25.3.4 Printing Unusual Data Structures

---

(HPRINT EXPR FILE UNCIRCULAR DATATYPESEEN)

[Function]

Using **HPRINT** to save structures that include pointers to raw storage will cause stack overflows. This includes dumping things using the **VARS**, **UGLYVARS**, or **HORRIBLEVARS** filemanager commands.

For example, a font descriptor points to raw storage, and cannot be dumped; for that reason, other system data types (e.g. windows) that point to fonts also cannot be dumped.

---

## Section 25.4 Random Access File Operations

---

### (III:25.20)

The first argument in the **FILEPOS** function should be called *STR* not *PATTERN*.

### (III:25.20)

In the Medley release, the function **COPYBYTES** now accepts *START* and *END* arguments even when the input stream is not random access. This caused an error in earlier releases.

---

## Section 25.6 PRINTOUT

---

### (III:25.27)

The PRINTOUT command **.FONT** changes the **DSPFONT** font permanently, that is, even after printout finishes.

---

## Section 25.8.3 READ Macros

---

### (III:25.42-43)

These **READMACROS** appear only in the OLD-INTERLISP-T readtable. (See Section 2 for a description of Lyric readtables.)

---

# Chapter 26 User Input/Output Packages

---

---

## Section 26.3 ASKUSER

---

(ASKUSER WAIT DEFAULT MESS KEYLST TYPEAHEAD LISPXPRNTFLG  
OPTIONSLST FILE)

[Function]

**ASKUSER** does not accept a string to mean a stream open on the string; you must call **OPENSTRINGSTREAM** if that's what you mean.

---

## Section 26.4 TTYIN Display Typein Editor

---

### (III:26.22)

The following fixes have been made to TTYIN in the Medley release:

- TTYIN now respects the **DSPLEFTMARGIN** of the *tydisplaystream*, rather than assuming it is zero.
- You can now assign the keyaction 194 (octal 302--acute accent in the NS character set) to a key and TTYIN will not treat it like the UNDO key (except on the 1132, where this functionality is still on blank-middle).

- TTYIN correctly handles prompts that are wider than the window.
- TTYIN now handles NS characters correctly when you are using a fixed-width font into which you have coerced, say, Classic characters for the non-zero character sets.
- TTYIN now handles Escape completion much more efficiently. If the completion is ambiguous, it completes the unambiguous prefix (as it did in Koto but not Lyric); it also correctly interprets escape characters. For example, in an exec with Common Lisp readtable, it correctly completes symbols that start with \, or a mixed-case symbol written with vertical bars. Also, Escape completion computes character widths correctly when it lowercases an upper case string, rather than leave some garbage bits on the display.
- The off-by-the-descent bug wherein TTYIN sometimes left stray bits at the bottom of the window has been fixed.

### Section 26.4.3 Display Editing Commands

---

#### (III:26.25)

?= and Meta-P no longer hang if you had an unbalanced string quote in the input.

?, Meta-P, and the **FIX** command now work correctly when there are NS characters in the input.

The printout for ?= is now improved; it respects \*print-case\*, matches up keywords better, and prints abstract syntax descriptions (such as for cl:do) a bit more clearly.

SMARTARGLIST fetches the argument lists of cl:compiled functions, so ?= now works in more cases.

The Ctrl-X command, when the caret is already positioned at the end of the input and everything but parentheses are balanced (i.e., no unbalanced string quotes or vertical bars), types as many closing parentheses as necessary to complete the input and then returns, much as if you had typed right bracket ("]") in Interlisp. Thus, if the cursor is somewhere in the middle of the input, typing two Ctrl-X's is sufficient to complete (assuming all you needed to type were some more parens).

TTYIN can now be used as a substitute for PROMPTFORWARD. The new function TTYINPROMPTFORWARD takes the same set of arguments as PROMPTFORWARD. In the most common cases it then calls TTYIN in "promptforward" mode, so that you can use the mouse and other TTYIN commands on the input. For cases it can't handle, it calls the old PROMPTFORWARD. These cases are: DONTechotypein.FLG or KEYBD.CHANNEL is non-NIL; ECHO.CHANNEL is not a displaystream; or TERMINCHARS.LST contains a character other than cr, space or ^X and you have set the variable TTYIN.USE.EXACT.CHARS (initially NIL) to T. TTYIN saves the old definition of PROMPTFORWARD, so you can either have your program explicitly call TTYINPROMPTFORWARD instead of PROMPTFORWARD, or you can have all calls to

PROMPTFORWORD changed by doing a (MOVD 'TTYINPROMPTFORWORD 'PROMPTFORWORD).

## Section 26.4.5 Useful Macros

(III:26.29)

**CTRLUFLG** is no longer supported by default. To use this feature, turn it on explicitly: (**INTERRUPTCHAR (CHARCODE ^U) 'CTRLUFLG**).

## Chapter 27 Graphic Output Operations

### Section 27.1.3 Bitmaps

Note: The printed representation of bitmaps has changed. Please see release notes Chapter 3, Integration of Interlisp-D/ Common Lisp, "Bitmap Syntax."

(III:27.4)

The following function has been added to Bitmap Operations between the functions **EXPANDBITMAP** and **SHRINKBITMAP**:

**(ROTATE-BITMAP BITMAP)** [Function]

Given an m-high by n-wide bitmap, this function returns an n-high by m-wide bitmap. The returned bitmap is the image of the original bitmap, rotated 90 degrees clockwise.

(III:27.4)

In the Medley release, the **EDITBM** function is substantially faster with the inclusion of **FASTEDITBM** (a former LispUsers module) in the sysout.

### Section 27.3 Accessing Image Stream Fields

The following functions were not documented in the Koto release of the *Interlisp-D Reference Manual*:

**(DSPCLEOL XPOS YPOS HEIGHT)** [Function]

"Clear to end of line". Clears a region from (XPOS,YPOS) to the right margin of the display, with a height of HEIGHT. If XPOS and YPOS are **NIL**, clears the remainder of the current display line, using the height of the current font.

**(DSPRUBOUTCHAR DS CHAR X Y TTBL)** [Function]

Backs up over character code CHAR in the display stream DS, erasing it. If X, Y are supplied, the rubbing out starts from the position specified. **DSPRUBOUTCHAR** assumes CHAR was printed with the terminal table TTBL, so it knows to handle control characters, etc. TTBL defaults to the primary terminal table.

## Section 27.6 Drawing Lines

---

### (III:27.17)

The non-NIL value of the *DASHING* argument of **DRAWLINE** uses *LINEWITHBRUSH*. *LINEWITHBRUSH* is a width-by-width brush which draws then lifts.

In the Medley release, when using the color argument, Interpress **DRAWLINE** treats 16x16 bitmaps or negative numbers as shades/textures. Positive numbers continue to refer to color maps, and so cannot be used as textures. To convert an integer shade into a negative number use *NEGSHADE* (e.g. (*NEGSHADE* 42495) is -23041).

### (III:27.18)

The **RELDRAWTO** function has been corrected so that it no longer draws a spot if the *DX* and *DY* arguments are 0.

## Section 27.7 Drawing Curves

---

### (III:27.18)

For the brush width value of **NIL**, the previous default value (**ROUND 1**) has been changed. The default value for the brush width value **NIL** is the **DSPSCALE** of the stream (that is, 1 printer's point wide).

### (III:27.19)

A new image stream function, **DRAWARC**, follows **DRAWCIRCLE** in the *InterLisp-D Reference Manual*.

|                                                                                                                                      |            |
|--------------------------------------------------------------------------------------------------------------------------------------|------------|
| <b>(DRAWARC</b> <i>CENTERX</i> <i>CENTERY</i> <i>RADIUS</i> <i>STARTANGLE</i> <i>NDEGREES</i> <i>BRUSH</i><br><i>DASHINGSTREAM</i> ) | [Function] |
|--------------------------------------------------------------------------------------------------------------------------------------|------------|

Draws an arc of the circle whose center point is (*CENTERX* *CENTERY*) and whose radius is *RADIUS* from the position at *STARTANGLE* degrees for *NDEGREES* number of degrees. If *STARTANGLE* is 0, the starting point will be (*CENTERX* (*CENTERY* + *RADIUS*)). If *NDEGREES* is positive, the arc will be counterclockwise. If *NDEGREES* is negative, the arc will be clockwise. The other arguments are interpreted as described in **DRAWCIRCLE**.

## Section 27.8 Miscellaneous Drawing and Printing Operations

---

### (III:27.20)

To have a filled polygon print correctly, set the global variable **PRINTSERVICE** to floating point value 9.0 for printers running Services 9.0 or later.

When using **FILLPOLYGON** to be sent to Xerox 8044 Interpress printers, the global variable **PRINTSERVICE** must be set to the same value as the Print Service installed on your printer, currently either 8.0, 9.0 or 10.0. Thus, if your printer is running Print Service 9.0, you must set the global variable **PRINTSERVICE** to the floating

point value 9.0. This works around an incompatible change in the Xerox 8044 Interpress implementation.

In Medley, Interpress curves are now rendered at a lower accuracy, allowing faster hardcopy. The spline is now rendered at 1/150 inch; in Lyric it was 1/300 inch.

The following function was omitted from previous version of the *Interlisp-D Reference Manual*:

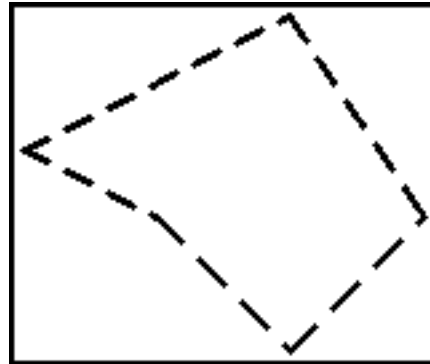
**(DRAWPOLYGON POINTS CLOSED BRUSH DASHING STREAM)** [Function]

Draws a polygon on the image stream *STREAM*. *POINTS* is a list of positions to which the figure will be fitted (the vertices of the polygon). If *CLOSED* is non-NIL, then the starting position is specified only once in *POINTS*. If *CLOSED* is NIL, then the starting vertex must be specified twice in *POINTS*. *BRUSH* and *DASHING* are interpreted as described in Chapter 27 of the *Interlisp-D Reference Manual*.

For example,

```
(DRAWPOLYGON ' ((100 . 100) (50 . 125)
 (150 . 175) (200 . 100) (150 . 50))
 T ' (ROUND 3) ' (4 2) XX)
```

would draw a polygon like the following on the display stream XX.



(III:27.20)

The function **FILLPOLYGON** contains two new arguments, *OPERATION* and *WINDNUMBER*. The new form for the function, and definitions for added arguments, follow.

**(FILLPOLYGON POINTS TEXTURE OPERATION WINDNUMBER STREAM)** [Function]

*OPERATION* is the **BITBLT** operation (see page 27.15 in the *Interlisp-D Reference Manual*) used to fill the polygon. If the *OPERATION* is **NIL**, the *OPERATION* defaults to the *STREAM* default *OPERATION*.

*WINDNUMBER* is the number for the winding rule convention. This number is either 0 or 1; 0 indicates the "zero" winding rule, 1 indicates the "odd" winding rule.

When filling a polygon, there is more than one way of dealing with the situation where two polygon sides intersect, or one polygon is fully inside the other. Currently, **FILLPOLYGON** to a display stream uses the "odd" winding rule, which means that intersecting polygon sides define areas that are filled or not filled somewhat like a checkerboard. For example,



```
(FILLPOLYGON ' ((125 . 125) (150 . 200) (175 . 125)
 (125 . 175) (175 . 175))
 GRAYSHADE WINDOW)
```

would produce a display something like this:



This fill convention also takes into account all polygons in *POINTS*, if it specifies multiple polygons.

---

### Section 27.12 Fonts

---

A revised set of font printing metrics is a part of the Lyric release of Lisp. Note that Koto font files are still available to users who request them.

With the revised font set the interline spacing (line leading) is now consistent across all fonts within a point size. Previously, text with multiple fonts (but with the same point size, i.e., if a word were made bold or italic, or if the family were changed) would have different leading on different lines. The new .WD files clean up document appearance.

Note that these printer metric changes affect only hardcopy, not the display. The contents of the display fonts are essentially unchanged in Lyric.

Generally, line leading in the Lyric font files is tighter than in previous releases of the fonts. The default line leading is now the same as the font's nominal point size. As a consequence of the above, any text file (one not already formatted for Interpress) which is printed after installation of the new fonts will be formatted to a different length. This means that decisions regarding TEdit line leading, widows and orphans, left/right pages, references to page numbers, etc. will need to change. Koto documentation produced by users may need to be reformatted with different line leading, using the new fonts.

All of the font files now have a new naming scheme, which allows **FONTSAVAILABLE** to be able to do more accurate pattern matching. For example, the display font file for modern 8 bold italics used to be named:

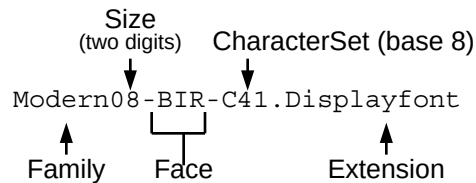
```
Modern8-B-I-C41.Displayfont
```

The file is now named:

```
Modern08-BIR-C41.Displayfont
```

In general font files use the following format:

The family name (e.g., Modern); a two digit size (e.g., 08); a three letter Face (e.g., BIR, for Bold Italic Regular); the letter C followed by the font's character set in base 8 (e.g., C41); and finally an extension (e.g., Displayfont).



The old file naming convention is still supported, however, with the exception of the old Strike file naming convention. In Lyric, **FONTCREATE** will first search for fonts using the new font naming convention, and if the desired font is not found it will search using the Koto convention.

**Compatibility considerations** You can continue using the old printer metrics (.WD files) in Lyric, thus preserving document looks between Koto and Lyric. If you choose to do so, it is recommended that you rename your old .WD files to the new naming scheme (see above), so that you benefit from the changes to the font searching mechanisms. However, we strongly urge you to use the new .WD files. Otherwise, if you exchange TEdit documents with a site that is using the new files, the documents will print differently at the two sites. The creation date, rather than the naming convention, determines whether a .WD file represents the old or new format.

If, after installing the new .WD files, you wish to print a document using the old Koto formatting, make the font variable **INTERPRESSFONTDIRECTORIES** point to a directory containing the Koto font files. Also any Lyric printer font file information must be uncached from the sysout. To uncache the fonts, perform

```
(for INFO in (FONTSAVAILABLE '* '* '* '* '*
 'INTERPRESS)
 do (APPLY 'SETFONTDESCRIPTOR INFO))
```

(III:27.30)

**(STRINGWIDTH STR FONT FLG RDTBL)** [Function]

In Lyric **STRINGWIDTH** observes **\*PRINT-LEVEL\*** and **\*PRINT-LENGTH\***.

In Medley, **STRINGWIDTH** with a NIL argument no longer returns the string width of the string with **\*STANDARD-OUTPUT\*** font. It now uses **DEFAULTFONT**.

Some new font manipulation functions have been added to Lisp. They are:

**(WRITESTRIKEFONTFILE FONT CHARSET FILENAME)** [Function]

Takes a display font font descriptor and a character set number, and writes that character set into a file suitable for reading in again. Note that the font descriptor's current state is used (which was perhaps modified by **INSPECT**ing the datum), so this provides a mechanism for creating/modifying new fonts.

For example:

```
(WRITESTRIKEFONTFILE (FONTCREATE 'GACHA 10) 0
 '{DSK}Magic10-MRR-C0.DISPLAYFONT)
```

writes a font file which is identical in appearance to the current state of Gacha 10 charset 0.

If your DISPLAYFONTDIRECTORIES includes {DSK}, then a subsequent (FONTCREATE 'MAGIC 10) will create a new font descriptor whose appearance is the same as the old Gacha font descriptor.

However, the new font is identical to the old one in appearance only. The individual datatype fields and bitmap may not be the same as those in the old font descriptor, due to peculiarities of different font file formats.

---

### Section 27.13 Font Files and Font Directories

---

(III:27.31)

Press fonts are not part of the sysout since PRESS is now a Library module.

---

### Section 27.14 Font Classes

---

(III:27.32-27.48)

This section has been expunged from the *InterLisp-D Reference Manual*. Renumber the sections which followed the old Section 27.14 as

**SECTION 27.15 ⇒ SECTION 27.14 Font Profiles**

**SECTION 27.16 ⇒ SECTION 27.15 Image Objects**

**SECTION 27.17 ⇒ SECTION 27.16 Implementation of Image Streams**

---

### Section 27.14 Font Profiles

---

(III:27.34)

The variable **FONTCHANGEFLG** has an additional value, **ALL**. **FONTCHANGEFLG=ALL** indicates that all calls to **CHANGEFONT** are executed.

(III:27.33-34)

The function **FONTNAME** no longer exists. This function was previously used in Interlisp-D to collect the names and values of variables on **FONTDEFSVARS**. The variable **FONTDEFSVARS** is no longer used; it was appropriate when most output devices were fixed-pitch, "line-printer" style devices, but is not suitable for use when most output devices are laser printers.

---

## Chapter 28 Windows and Menus

---

---

### Section 28.4 Windows

---

(III:28.13, 28.38)

The **ADDMENU** function will change a window's **RESHAPEFN** and also will change the window's **REPAINTFN**.

---

### Section 28.4.5 Reshaping Windows

---

#### (III:28.17)

The Lisp window system allows the following minimum window sizes:

When creating a new window, the width and height specified must be at least 9, or else you will get an error "region too small to use as a window"

When reshaping a window, the smallest shape you can get is width = 26 and height = height of the font to be used in the window. If you specify a smaller region, **SHAPEW** will simply adjust it to fit these limits.

---

### Section 28.4.8 Shrinking Windows Into Icons

---

#### (III:28.22)

##### **SHRINKFN**

[Window property]

In previous releases, there was a bug in the attached window system such that if an attached window had a **SHRINKFN** of the single symbol DON'T, attempting to shrink the window resulted in a break with the message "UNDEFINED FUNCTION DON'T." For this case in Lyric, all windows that can be shrunk will be, while those windows with a **SHRINKFN** of the symbol DON'T will be left open.

To facilitate the management of window regions, the window property **EXPANDREGIONFN** has been added to Lisp. This feature allows applications to arrange for reshaping a window when it is expanded.

##### **EXPANDREGIONFN**

[Window property]

**EXPANDREGIONFN**, if non-NIL, should be the function to be called (with the window as its argument) before the window is actually expanded.

The **EXPANDREGIONFN** must return **NIL** or a valid region, and must not do any window operations (e.g., redisplaying). If **NIL** is returned, the window is expanded normally, as if the **EXPANDREGIONFN** had not existed. The region returned specifies the new region for the main window only, not for the group including any of its attached windows. The window will be opened in its new shape, and any attached windows will be repositioned or rejustified appropriately. The main window must have a **REPAINTFN** which can repaint the entire window under these conditions.

As with expanding windows normally, the **OPENFN** for the main window is not called.

Also, the window is reshaped without checking for a special shape function (e.g., a **DOSHAPEFN**).

##### (III:28.23)

Add the variable **DEFAULTICONFN** to the Icon section of the *InterLisp-D Reference Manual*:

##### **DEFAULTICONFN**

[Variable]

Changes how an icon is created when a window having no **ICONFN** is shrunk or when **SHRINKW**, with a *TOWHAT* argument of a string, is called. The value of **DEFAULTICONFN** is a function of two arguments (window text); text is either **NIL** or a string. **DEFAULTICONFN** returns an icon window.

The initial value of **DEFAULTICONFN** is **MAKETITLEBARICON**. It creates a window that is a title bar only; the title is either the text argument, the window's title, or "Icon made <date>" for titleless windows. **MAKETITLEBARICON** places the title bar at some corner of the main window.

An alternative behavior is available by setting **DEFAULTICONFN** to be **TEXTICON**. **TEXTICON** creates a titled icon window from the text or window's title. It is described further in Appendix B (ICONW).

##### (III:28.23)

You can now copy-select titled icons such as those used by FileBrowser, SEdit, TEdit, Sketch. The default behavior is that the icon's title is unread (via **BKSYSBUF**), but if the icon window has a **COPYFN** property, that gets called instead, with the icon window as its argument. For example, if the name displayed in an icon is really a symbol, and you want copy selection to cause the name to be unread correctly with respect to the package and read table of the exec you are copying into, you could put the following **COPYFN** property on the icon window:

```
(lambda (window)
 (il:bksysbuf <fetch symbolic name from window> t))
```

---

#### Section 28.4.11 Terminal I/O and Page Holding

##### (III:28.29)

**TTYDISPLAYSTREAM** has been fixed so that it can be successfully used with non-windows.

---

#### Section 28.5 Menus

Two features have been added to this section, **ICONW** for creating icons, and **FREE MENU**, for creating and using free menus. Both features were formerly part of the Lisp Library.

The description for **ICONW** is in Appendix C. The **FREE MENU** description is in Appendix D.

The Lyric version of Free Menu differs in some respects from the Koto version of Free Menu. Following is a description of the incompatible feature changes from the old version to the new version of Free Menu. Some of the terminology used in these

notes is introduced in the Free Menu documentation found in Appendix B. Please reference Appendix B before reading the following notes.

- The function **FREEMENU** is used to create a Free Menu, replacing and combining the functions **FM.MAKEMENU** and **FM.FORMATMENU**.

The description of Free Menu has these changes:

1. There is no longer a **WINDOWPROPS** list in the Free Menu Description. Instead, the window properties **TITLE** and **BORDER** that were previously set in the **WINDOWPROPS** list can now be passed to the function **FREEMENU**. Other window properties (like **FM.PROMPTWINDOW**) can be set directly after Free Menu returns the window using the system function **WINDOWPROP**. See Appendix B, Section 28.7.14, Free Menu Window Properties.
2. Setting the initial state of an item is now done with the item property **INITSTATE** in the item description, rather than the **STATE** property.

Free Menu Items has been modified as follows:

1. **3STATE** items now have states **OFF**, **NIL**, and **T** (instead of a **NEUTRAL** state). They appear by default in the **NIL** state.
2. **STATE** items are general purpose items which maintain state, and replace the functionality of **NCHOOSE** items. To get the functionality of **NCHOOSE** items, specify the property **MENUITEMS** (a list of items to go in a popup menu), which instructs the **STATE** item to popup the menu when it is selected. **STATE** items do not display their current state by default, like **NCHOOSE** items used to. Instead, if you want the state displayed in the Free Menu, you have to link the **STATE** item to a **DISPLAY** item using a Free Menu Item Link named "DISPLAY". The current state of the **STATE** item will then automatically be displayed in the specified **DISPLAY** item. The item properties **MENUFONT** and **MENUTITLE** also apply to the popup menu.
3. **NWAY** items are declared slightly differently. There is now the notion of an NWay Collection, which is a collection of items acting as a single nway item. The Collection is declared by specifying any number of NWay items, each with the same **COLLECTION** property. NWay Collections have properties themselves, accessible by the macro **FM.NWAYPROPS**. These properties can be specified in property list format as the value of the **NWAYPROPS** Item Property of the first NWay item declared for each Collection. NWay Collections by default cannot be deselected (a state in which no item selected). Setting the Collection property **DESELECT** to any non-nil value changes this behavior. The state of the NWay Collection is maintained in its **STATE** property.
4. **EDIT** items no longer will stop at the edge of the window. Editing is either restricted by the **MAXWIDTH** property, or else it is not restricted at all. The **EDITSTOP** property is obsolete.

When you start editing with the right mouse button the item is first cleared.

5. **EDITSTART** items now specify their associated edit item (there can only be one, now) by a Free Menu Item Link named "EDIT" from the **EDITSTART** item to the **EDIT** item.
6. **TITLE** items are replaced by **DISPLAY** items, which work the same way.

With Free Menu, the item interface functions can take the actual item datatype, the item's *ID* or *LABEL*, or a list of the form (**GROUPID** *ITEMID*) specifying a particular item in a group, as the *ITEM* argument.

The description for **ICONW** is in Appendix B. The **FREE MENU** description is in Appendix C.

These changes have occurred in the Free Menu Interface functions:

(**FREEMENU** *DESCRIPTION TITLE BACKGROUND BORDER*) [Function]

Replaces **FM.MAKEMENU** and **FM.FORMATMENU**. The desired format is not specified as the value of the **FORMAT** property in the group's PROPS list.

(**FM.GETITEM** *ID GROUP WINDOW*) [Function]

Replaces **FM.ITEMFROMID**.

Searches within *GROUP* for an item whose ID property is *ID*.

*ID* is matched against the item ID and then the item **LABEL**. If *GROUP* is **NIL**, the entire menu is searched.

(**FM.GETSTATE** *WINDOW*) [Function]

Replaces **FM.READSTATE**.

Returns a property list of the selected item in the menu. This list now also includes the NWay Collections and their selected item.

(**FM.CHANGELABEL** *ITEM NEWLABEL WINDOW UPDATEFLG*) [Function]

Has a new argument order. Now works by rebuilding the item label from scratch, taking the original specification of **MAXWIDTH** and **MAXHEIGHT** into account. *NEWLABEL* can be an atom, string, or bitmap. If *UPDATEFLG* is set, then the Free Menu Group's regions are recalculated, so that boxed groups will be redisplayed properly.

(**FM.CHANGESTATE** *X NEWSTATE WINDOW*) [Function]

Has a new argument order.

*X* is either an item or an NWay Collection ID. *NEWSTATE* is an appropriate state to the type of item. If an NWay collection, *NEWSTATE* is the actual item to be selected, or **NIL** to deselect. Toggle items take either **T** or **NIL** as *NEWSTATE*, and **3STATE** items take **OFF**, **NIL**, or **T**, and **STATE** items take any atom, string, or bitmap as their new state. For **EDIT** items, *NEWSTATE* is the new label, and **FM.CHANGELABEL** is called to change the label of the **EDIT** item.

(**FM.RESETSHAPE** *WINDOW ALWAYSFLG*) [Function]

Replaces **FM.FIXSHAPE**

(FM.HIGHLIGHTITEM *ITEM WINDOW*)

[Function]

Replaces **FM.SHADEITEM** and **FM.SHADEITEMBM**.

**FM.HIGHLIGHTITEM** will programmatically highlight an item, as specified by its **HIGHLIGHT** property. The highlighting is temporary, and will be undone by a redisplay or scroll. To programmatically shade an item an arbitrary shade, use the new function **FM.SHADE**.

---

### Section 28.6.2 Attached Prompt Windows

(GETPROMPTWINDOW *MAINWINDOW #LINES FONT DONTCREATE*)

[Function]

In the Lyric release, the prompt window created by **GETPROMPTWINDOW** is *not* independently closeable, as it was in Koto. That is, selecting **Close** from the right-button window menu in the prompt window is the same as selecting it from the menu of any other window in the group—the entire window group is closed.

---

### Section 28.6.3 Window Operations and Attached Windows

#### (III:28.51)

Communication of Window Menu Commands between Attached Windows is dependent on the name of function used to implement the window command, e.g., **CLOSEW** implements **CLOSE** (refer to **PASSTOMAINCOMS** documentation under Attached Windows). Consequently, if an application intercepts a window command by changing **WHENSELECTEDFN** for an item in the WindowMenu (for example, to advise the application that a window is being closed), windows may not behave correctly when attached to other windows.

To get around this problem, the Medley release provides the variable *\*attached-window-command-synonyms\**. This variable is an ALIST, where each element is of the form (new-command-function-name . old-command-function-name).

For example, if an application redefines the WindowMenu to call *my-close-window* when **CLOSE** is selected, that application should:

```
(cl:push '(my-close-window . il:closew)
il:*attached-window-command-synonyms*)
```

in order to tell the attached window system that *my-close-window* is a synonym function for **CLOSEW**.

---

## Chapter 29 Hardcopy Facilities

#### (III:29.3)

The **HARDCOPYW** function now has an additional argument, **HARDCOPYTITLE**, which allows you to change or eliminate the



"Window Image" message on IP screen images. Moreover, **HARDCOPYW** function now allows you to print large images occupying more than one page.

(**HARDCOPYW** WINDOW/BITMAP/REGION FILE HOST SCALEFACTOR ROTATION  
PRINTERTYPE **HARDCOPYTITLE**) [Function]

---

**HARDCOPYTITLE** is a string specifying a title to print on the page containing the screen image. If **NIL**, the string "Window Image" is used. To omit a title, specify the null string.

---

## Chapter 30 Terminal Input/Output

---

### Section 30.1 Interrupt Characters

---

#### (III:30.2)

- |           |                                                                                                                                                                                      |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Control-P | The Control-P ( <b>PRINTLEVEL</b> ) interrupt is no longer supported. The interrupt of that name still exists and is defaultly assigned to Control-P, but has no effect on printing. |
| Control-T | The Control-T interrupt flashes the window belonging to the tty process and prints its status information in the prompt window. This avoids disrupting the user typescript.          |

#### (III:30.3)

(**INTERRUPTCHAR** CHAR TYP/FORM **HARDFLG** —) [Function]

---

If the argument *TYP/FORM* is a symbol designating a predefined system interrupt (**RESET**, **ERROR**, **BREAK**, etc), and **HARDFLG** is omitted or **NIL**, then the hardness defaults to the standard hardness of the system interrupt (e.g., **MOUSE** for the **ERROR** interrupt).

### Section 30.2.3 Line Buffering

---

#### (III:30.11-12)

The **BKSYSBUF** function has been changed, for compatibility reasons. The description now reads as follows:

(**BKSYSBUF** *X FLG RDTBL*) [Function]

---

**BKSYSBUF** appends the **PRIN1**-name of *X* to the system input buffer. The effect is the same as though the user had typed *X*. Returns *X*.

If *FLG* is *T*, then the **PRIN2**-name of *X* is used, computed with respect to the readtable *RDTBL*. If *RDTBL* is **NIL** or omitted, the current readtable of the TTY process (which is to receive the characters) is used. Use this for copy selection functions that want their output to be a readable expression in an Exec.

Note that if you are typing at the same time as the **BKSYSBUF** is being performed, the relative order of the typein and the characters of *X* is unpredictable.

#### (III:30.12)

Add the function **BKSYSCHARCODE** used in line buffering:

(BKSYSCHARCODE CODE) [Function]

This function appends the character code *CODE* to the system input buffer. The function **BKSYSBUF** is implemented by repeated calls to **BKSYSCHARCODE**.

### Section 30.4.1 Changing the Cursor Image

(III:30.14)

The **CURSOR** record has been changed to a DATATYPE, and its field names have changed in the following way:

| <u>Old Field Name</u> | <u>New Field Name</u> |
|-----------------------|-----------------------|
| <b>CURSORBITMAP</b>   | <b>CUIMAGE</b>        |
| <b>CURSORHOTSPOTX</b> | <b>CUHOTSPOTX</b>     |
| <b>CURSORHOTSPOTY</b> | <b>CUHOTSPOTY</b>     |

The **CURSORHOTSPOT** field no longer exists; its value can be fetched by composing **CUHOTSPOTX** and **CUHOTSPOTY** into a **POSITION**, or stored by destructuring a **POSITION** into those fields.

In Lyric, the **CURSORCREATE** function accepted as its argument bitmaps of any size, but caused an obscure error. In Medley, a bitmap that is bigger than 16 high or 16 wide will cause an ILLEGAL ARGUMENT error.

### Section 30.5 Keyboard Interpretation

(III:30. 19–20)

(KEYDOWNP KEYNAME) [Function]

(KEYACTION KEYNAME ACTIONS —) [Function]

*KEYNAME* is interpreted differently in Lyric: If *KEYNAME* is a small integer, it is taken to be the *internal* key number. Otherwise, it is taken to be the name of the key. This means, for example, that the name of the "6" key is not the number 6. Instead, spelled-out names for all the digit keys have been assigned. The "6" key is named SIX. It happens that the key number of the "6" key is 2. Therefore, the following two forms are equivalent:

(KEYDOWNP 'SIX)

(KEYDOWNP 2)

Note: The key labeled HELP on the 1186 is named DBK-HELP for use in KEYACTION.

### Section 30.6 Display Screen

(III:30.22-23)

(CHANGEBACKGROUND SHADE —) [Function]

The function **CHANGEBACKGROUND** treats the *SHADE* argument as a 4 X 4 texture. The **CHANGEBACKGROUND** function, on the other hand, treats the *SHADE* argument as a 2 X 8 texture.

Therefore, note that the same *SHADE* argument, when used by the two functions, will not necessarily produce the same background and border shades on the display screen.

---

#### (III:30.23)

The **VIDEORATE** function works only on the 1108. Append the following note to the **VIDEORATE** function description:

(VIDEORATE TYPE)

[Function]

Note: **VIDEORATE** does not work on the 1186.

### Section 30.7 Miscellaneous Terminal I/O

---

#### (III:30.24)

(BEEPON FREQ)

[Function]

The argument *FREQ* is measured in hertz, not in **TICKS**.

---

## Chapter 31 Ethernet

---

### Section 31.3.1 Name and Address Conventions

---

#### (III:31.8-9)

Amend the first paragraph, describing **NSADDRESS**, to list, in order, the components of **NSADDRESS**:

Addresses of hosts in the NS world consist of three parts, a network number, a machine number, and a socket number. These three parts are embodied in the Interlisp-D data type **NSADDRESS**. The components of **NSADDRESS** are 32-bit network, 48-bit host, 16-bit socket.

Move the following sentence from page 31.9 of the *IRM* to the last paragraph of Name and Address Conventions on page 31.8:

If you wish to manipulate **NSADDRESS** and **NSNAME** objects directly you should load the Lisp Library Module **ETHERRECORDS**.

#### NS Address Format

---

In Medley, you can now specify NS addresses in decimal notation, the form presented by the Chat interface of Network Services software. In this notation, a decimal number is broken up by hyphens every 3 digits, much like commas in standard American numerical notation. You can also specify a full 48-bit host number in octal without breaking it into 16-bit segments.

An NS address is specified in the form:

*net#host#socket*

If the address contains a hyphen in any field, the entire address is interpreted in decimal; otherwise in octal. The field *socket* is optional, and is defaulted appropriately for the application; if specified, it is a single integer in the same radix as the rest of the address. The field *net* and its terminating # are optional, defaulting usually to the directly-connected network. The fields *net* and *host* are non-negative integers written in one of three forms:

- A sequence of 16-bit octal numbers, separated by periods.
- A single integer in octal radix.
- A sequence of 3-digit decimal numbers, separated by hyphens.

The special variable **\*NSADDRESS-FORMAT\*** specifies the form used whenever the system prints an NS address object. Its possible values are:

**NIL** Octal radix, with the host number in three 16-bit parts, the same as in Lyric.

**:OCTAL** Octal radix without separators.

**:DECIMAL** Decimal radix with hyphens.

For example, the following all represent the same address, in the three formats listed above:

1750#0.125000.76771#

1750#25200076771#

1-000#2-852-158-969#

The following functions exist for manipulating NS addresses:

**(PARSE-NSADDRESS STR DEFAULTSOCKET)** [Function]

Parses the string *STR* into an NS address by the rules listed above, or returns NIL if *STR* is not a well-formed address. If *DEFAULTSOCKET* is non-NIL and the string does not include a socket field, the socket of the resulting NS address is set to *DEFAULTSOCKET*.

**(COERCE-TO-NSADDRESS HOST DEFAULTSOCKET)** [Function]

Returns an NSADDRESS object corresponding to *HOST*, or NIL if it can't. This function should be called by any software wanting to convert a user-supplied NS host specification into a network address. *HOST* can be any one of the following:

- The name of a host, whose address is found by consulting the Clearinghouse data base.
- A symbol or string in the syntax of an NS address, as described above.
- An NSADDRESS object.
- A list of the form (NSHOSTNUMBER *a b c*), specifying the host number as three 16-bit values. In this case, the network number is omitted (zero).

If *DEFAULTSOCKET* is non-NIL and the socket is unspecified, the socket of the result is set to *DEFAULTSOCKET* (if *HOST* is an NSADDRESS object, it is copied in this case).

#### Network Routing Maintenance

---

The representation of Pup and NS routing tables has changed, and the background gateway listener processes have been tuned to significantly reduce their overhead.

The INFO command for the Pup and NS gateway listener processes in the process status window now display their routing tables. Clicking with the left button displays the table in random order, middle button displays the table sorted by network number (this takes a little longer).

#### Section 31.3.2 Clearinghouse Functions

---

##### (III:31.9)

The variable **AUTHENTICATION.NET.HINT** has been added to Clearinghouse Functions. It follows the **CH.NET.HINT** variable in the *Interlisp-D Reference Manual*.

##### AUTHENTICATION.NET.HINT

[Variable]

**AUTHENTICATION.NET.HINT** can be set to **CH.NET.HINT** to speed up the initial authentication connection. Its value is interpreted in the same manner as **CH.NET.HINT**.

#### Section 31.3.3 NS Printing

---

##### (III:31.12)

With the Medley release there is now a single Printer Watcher process for all NS printers. This mean you won't get a stack overflow if you hardcopy many files in quick succession.

#### Section 31.3.5.3 Performing Courier Transactions

---

##### (III:31.20-21)

The **COURIER.OPEN** function requires that a courier server be running on the host machine.

#### Section 31.3.5.3.3 Using Bulk Data Transfer

---

##### (III:31.24-25)

The following is a correction and clarification to the description in the *Interlisp-D Reference Manual* of receiving values from a bulk data transfer:

It is possible for a Courier procedure to return both bulk data, in the form of a bulk data sink, and a single value (or list of values) as the normal result of the call. However, the Lisp function **COURIER.CALL** only returns one value, either the bulk data stream (when the bulk data sink argument is NIL) or the regular value. There are two principal ways in which a caller can obtain both values.

The usual way to get both values is to pass a function as the bulk data argument, have it retrieve the bulk data and process it as a side-effect (e.g., store it into a variable bound around the **COURIER.CALL**), then return NIL so that the procedure's returned value is returned from **COURIER.CALL**.

The other way, which is documented incorrectly in the *IRM*, is to pass NIL as the bulkdata argument, thus getting the bulk data stream back from **COURIER.CALL**, process the stream, and then get the procedure's returned value by closing the stream. Contrary to the *IRM*, however, you have to close the bulk data stream using its internal close function, **SPP.CLOSE**, rather than the user-level function **CLOSEF**, which consumes the value internally and returns only the stream.

### Section 31.5 Pup Level One Functions

|                        |            |
|------------------------|------------|
| <b>\10MBTYPE.PUP</b>   | [Variable] |
| <b>\10MBTYPE.3TO10</b> | [Variable] |

The values of these variables are the 10MB Ethernet encapsulation types for PUP packets and Pup-to-10MB address translation packets, respectively. The initial values of these variables are 512 and 513, respectively. However, these values are illegal for an Ethernet conforming to IEEE 802.3 specifications.

New encapsulation types have been defined for IEEE 802.3 networks. To use them, set the variable **\10MBTYPE.PUP** to 2560 (decimal) and **\10MBTYPE.3TO10** to 2561. Then call either **(RESTART.ETHER)** or **(LOGOUT)**, so that the Ethernet code can reinitialize itself. It may be convenient for a site to smash these values directly into the standard sysout everyone fetches by using the function **READSYS** and its ^v command from the TeleRaid Library module (the sysout must be on disk or a random-access file server). Note that *all* pup hosts on a network (servers as well as workstations) must simultaneously choose to use the new values; those using different values will be unable to communicate with each other. The System Tool must also be upgraded at the same time.

### Section 31.6.1 Creating and Managing XIPs

The function NSNET.DISTANCE was previously undocumented.

|                              |            |
|------------------------------|------------|
| <b>(NSNET.DISTANCE NET#)</b> | [Function] |
|------------------------------|------------|

Returns the "hop count" to network *NET#*, i.e., the number of gateways through which an XIP must pass to reach *NET#*, according to the best routing information known at this point. The local (directly-connected) network is considered to be zero hops away. Current convention is that an inaccessible network is 16 hops away. **NSNET.DISTANCE** may need to wait to obtain routing information from an Internetwork Router if *NET#* is not currently in its routing cache.

[This page intentionally blank]

This section contains release notes indicating changes that have occurred in the library modules since the Lyric release. Medley changes are indicated with revision bars in the right margin.

Refer to the *Lisp Library Modules* manual, Medley release, for complete documentation of the library modules.

---

## Modules that are New, Moved, or Replaced

---

### Modules Moved from the Library to LispUsers

---

Big  
 BitMapFns  
 BusExtender  
 BusMaster  
 CirclPrint  
 CheckSet  
 CompileBang  
 Color  
 C150Stream  
 DECL  
 DInfo  
 FileCache  
 HelpSys  
 Iris  
 LambdaTran  
 PCallStats  
 ReadAIS

### Modules Moved from LispUsers to the Library

---

Cash-File  
 Hash-File  
 SysEdit  
 TableBrowser

### Modules Moved to Their Own Manuals

---

TEdit  
 Sketch  
 CML, CMLArray, CMLArrayInspector (part of Xerox Common Lisp)

### Modules Moved From the Sysout Into the Library

---

DEdit  
 Masterscope  
 Match



Press

### Modules Moved From the Library Into the Sysout

---

IconW  
FreeMenu

### Modules Replaced

---

Old: FX-80stream, FastFX-80stream, FXprinter  
New: FX-80Printer

Old: WhereIs  
New: Where-Is

### New Modules

---

SysEdit  
TableBrowser  
TextModules

---

## Details of Changes

---

### 4045XLPSream

---

Enabled its graphics capabilities; added 1108 cable/connector pin-outs.

A new function has been added to allow owners of the international 4045 (non-USA model) to use the 4045XLPSream software.

(4045XLP.CHANGE.MODE *MODE*)

[Function]

This function changes the internal parameters of the software to allow printing on A4 paper with the international fonts. *MODE* is a string, either "USA" or "INTERNATIONAL", with the default being "USA". Do not use this function unless you have the international font set and A4 paper tray on a non-USA 4045. A4 page size is 2475 pixels wide by 3525 pixels high in portrait, and 3525 x 2475 in landscape mode.

### Cash-File

---

The new library module Cash-File was formerly in LispUsers. Cash-File is a front end to Hash-File which uses a hash table to cache accesses to hash files. This can provide a significant performance improvement in applications which access a small number of keys repeatedly. For example, the Where-Is library

module uses this module to achieve acceptable interactive performance.

---

## Centronics

---

Added cable/connector pin-out.

---

## Chat

---

Added information about EMACS.

---

## CopyFiles

---

When told to copy to a non-existent NS subdirectory, it now asks if it should create it.

---

## DataBaseFns

---

Clarifications in the documentation of LOADDBFLG and SAVEDBFLG are included in Medley.

---

## EditBitMap

---

Added a description of its user interface.

---

## FileBrowser

---

Added enhanced features to Load, Compile, Edit; it now preserves path name of source files when copying to another machine or user; sorts files by attributes; and prints hard copies of directory listings.

The FB command now ignores the package of the attributes you optionally specify, so you can easily use it from a non-Interlisp exec.

The enclosing \*'s are now included with the names of the variables \*EDITMODE\* and \*DEFAULT-CLEANUP-COMPILER\* .

---

In addition to having outstanding problems fixed, FileBrowser has several new features and NS enhancements.

New features:

- There is an Abort button available during any operation of indefinite duration.
- You can scroll or reshape a FileBrowser that is "busy", e.g, while doing a Recompute.
- The browser title includes a timestamp of when the browser contents were last Recomputed.
- There is a new subcommand of See, "FileBrowse", which opens a FileBrowser on the selected subdirectory. This replaces the odd functionality of the old See/Edit commands that assumed

that any file with null name and extension must be a directory; those commands now always treat the selection as a file. FileBrowse is mainly useful in the following situations:

- When browsing NS directories with depth set finite, or when browsing the top level of a server, which is automatically depth 1.
- When browsing on Unix, a device that gives Lisp no indication of whether a filename is a directory or not.
- There is a subcommand of Recompute, "Set Depth" that can be used to set the enumeration depth for future recomputes and recursive FileBrowses. You can also set the depth in an FB command by appending :DEPTH n to the command line, e.g., FB "{Pogo:}<Carstairs>" :DEPTH 1.

The depth counts levels of directories below the last directory in the pattern not containing a wildcard; depth 1 means just the immediate descendants of that directory. Depth is ignored for nontrivial patterns, i.e., anything but "\*.\*".

- Another new subcommand of Recompute, "Shape to Fit", widens or narrows the browser so that all fields, and no more, are visible but not wider than the screen.
- Directory items are now displayed like files, e.g., you'll see a single line

Lisp>

rather than the double line

<Carstairs>Lisp>

NIL

In addition, the "page" count for a directory item is now the total page size of the directory subtree rooted there.

- FileBrowser consumes somewhat less storage now, and there have been some performance improvements, especially for very large browsers.

### FTPServer

---

FTPServer now supports DELFILE.

The Medley release fixes several bugs in Lisp's handling of PUP FTP connections relating to password handling and filename recognition.

### FX-80Driver

---

New software, new text, and 1108/1186 cable/connector pin-outs have been added.

Comments are now printed in a compressed font .

---

**GCHax**

---

Documentation contains a new description of the STORAGE function.

---

**Grapher**

---

Grapher can now print graphs larger than one page. The variable GRAPH/HARDCOPY/FORMAT is used to control the format of the graph when printing to paper. See the function HARDCOPYGRAPH and the variable GRAPH/HARDCOPY/FORMAT in the documentation for Grapher for more information.

A new GRAPH.PROPS field has been added to Graph record, which produces a list in property-list format, and is accessed by the function GRAPHERPROP.

---

**Hash**

---

Hash is provided for backwards compatability. New applications should use the Medley library module Hash-File instead of this module.

---

**Hash-File**

---

Hash-File is a new library module, upgraded from LispUsers. Hash-File is similar to but not compatible with the Lyric library module, Hash. Hash-File is modeled after the Common Lisp hash table facility, and Hash was modeled after the Interlisp hash array facility.

---

**Kermit**

---

Reference to an excellent text/reference book has been added.

---

**MasterScope**

---

Break when graying a browser has been fixed.

In Medley, MasterScope .LCOM files have been changed to .dfasl file extensions. MasterScope now recognizes Common Lisp structures.

---

**NSMaintain**

---

The module NSMaintain has been completely revised and has all new documentation. Most commands auto-complete on one or two keystrokes. The Change Password command works again, and there are several new commands for listing objects in the Clearinghouse data base and for manipulating the access lists of groups. There is a more rational set of default inputs offered for most commands, and better feedback is given as to whether a command succeeded or failed.

---

**RS232**

---

The RS232.TRACE function is now documented in the Medley release.

### Spy

---

This version of SPY library module works better with Common Lisp and incorporates several new features:

- Enters the pending mode when you bring up the SPY menu by pressing the left or middle button while the control key is down. Any action invoked from the menu is deferred until you next press the left or middle mouse button. For example, you can delete several nodes and then do one update.
- Keeps track of non-symbol frame names.
- Shows the package prefix of symbols in the display.
- Invokes "Merge" menu item from a node menu allowing for a node to merge with its caller.
- Updates SPY.NOMERGEFNS to correspond more closely to "system" functions in Medley.
- Knows about the Medley interpreter.

### TableBrowser

---

- New functions TB.UNSELECT.ITEM and TB.UNSELECT.ALL.ITEMS fill an inadvertant void in the Lyric version.
- Several off-by-ones in the display algorithms have been fixed.
- Performance on large browsers is improved.
- Clarification of TBAFTERCLOSEFN documentation is included in the Medley release.
- New options to TB.MAKE.BROWSER:
  - The option LINESPERITEM, previously documented but not implemented, is now supported. Alternatively, you can specify explicitly the height of items by giving the options ITEMHEIGHT (total height of each item) and/or BASELINE (the height of the "baseline" relative to the bottom of the item; zero if you don't set it). The BASELINE is used for two things: (1) the ypos of the window is set there when the browser's print function is called, and (2) selection and deletion marks are centered between the baseline and the top of the item. Specifying LINESPERITEM is a shorthand method for setting ITEMHEIGHT to fontheight\*#lines and BASELINE to fontheight\*(#lines-1)+fontdescent (i.e., font's baseline for the first line of the item), so that the selection marker, deletion lines, and positioning for printing all point at the first line of a multi-line item. One further difference: if you change the font of the browser, TableBrowser will recompute the height and baseline parameters if you specified LINESPERITEM, but not if you specified ITEMHEIGHT.

- You can specify an auxiliary window that is to be horizontally scrolled in parallel with the main window by giving the window as the `HEADINGWINDOW` option. The `WIDTH` of the window's `EXTENT` property is maintained in synch with main window. You still need to create the auxiliary window, attach it where you want it and supply it with a `REPAINTFN`. This is how `FileBrowser` implements its header line consisting of "Name" and the attribute names.
- The option `LINETHICKNESS` specifies how thick to draw deletion lines. It defaults to `TB.DELETEDLINEHEIGHT`, initially 1. Making it the height of an item gives an alternative "total blackout" method of deletion.

## TCP-IP

Added revised/expanded installation procedure.

DIR to VMS via TCP now works.

TCP Chat hosts can now be lowercase.

(`TCPFTP.SERVER`) now spawns process and runs in it .

TCP-IP to a Sun returns the top-level directory.

`TCPFTP.DEFAULT.FILETYPES` now contains correct entries for `LCOM`, `lcom`, `DFASL`, and `dfasl`.

Files loaded by the high-level modules `TCPFTP`, `TCPFTPSRV`, `TCPCHAT`, and `TCPTFTP` automatically load their dependencies. If you load files by hand, you must also load their dependencies first. See the section "File Dependencies," or the TCP-IP documentation for more information.

There is a new flag:

`TCP.ALWAYS.READ.HOSTS.FILE`

[Variable]

This flag is initially T. Setting it to NIL will cause the system to parse the `hosts.txt` file only when the filename (stored in the configuration file) is different from the previously read filename, or the write date of the file has changed. The `hosts.txt` file will always be read at least once when loading the software into a clean sysout.

If you change your `IP.INIT` file while TCP-IP is running, you will be prompted to confirm **Restarting TCP**. In most cases, you should confirm the restart.

### TExec

---

A TEXEC executive window no longer has GET in the menu of possible actions, since GETting text into an executive window makes no sense.

### TextModules

---

TextModules is a new library module with the Medley release. It can be used to import and export textfiles and File Manager files. It can bring portable Common Lisp sources into the File Manager without losing any of their contents, and create new textfiles based on the File Manager's description of the textfile contents.

### Virtual Keyboards

---

The Standard-Russian virtual keyboard now has uppercase Be ( . . . ) and Ve ( . . . ) in the right places.

---

Loading VirtualKeyboards now adds the item KEYBOARD to the default window menu as well as the background menu. Selecting this item from the default window menu allows you to specify a keyboard for an individual window.

### Where-Is

---

Where-Is is a new library module, upgraded from LispUsers. This module replaces the Lyric library module WhereIs. This is a new implementation of a facility similar to but not compatible with the Lyric library module WhereIs. Where-Is indexes all definers, but WhereIs only indexed Interlisp FNS definitions.

---

## Additional Notes

---

DEdit is not error-protected. Doing a ^in a DEdit break window closes the DEdit window, too.

In addition, the modules work under all Lisp environments (Interlisp-D, Xerox Common Lisp, Common Lisp). However, many of the functions and variables used within the modules are those of Interlisp-D, and therefore you'll have to make sure that, when you are not in Interlisp, you use the IL: prefix.

---

## Koto CML Library Module

---

If you have files that used the Koto CML library module, with its package-style symbol naming conventions, you will need to convert them to use the correct symbols in Lyric /Medley. The procedure is

briefly as follows: see the *Common Lisp Implementation Notes*, chapter 11, "Reader compatibility feature" for complete details on this mechanism:

First, set the global variable LITATOM-PACKAGE-CONVERSION-ENABLED to T. Then for each of your files, do

```
(LOAD file 'PROP)
```

```
(MAKEFILE file 'NEW)
```

Afterwards be sure to set the global variable LITATOM-PACKAGE-CONVERSION-ENABLED back to NIL.



[This page intentionally left blank]

## A User's Guide to TEdit—Release Notes

For the Medley Release, TEdit has increased the number of expanded characters, added options to the **Put** and **Get** submenus, clarified several options in the Paragraph Looks and Page Layout menus, and added several minor items to the programmer's interface.

### Expanded Characters

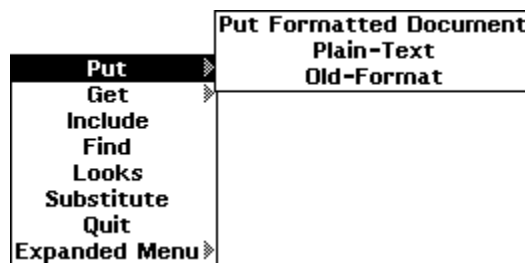
TEdit added the following abbreviations and expansions to the characters shown in Table 1.

Table 1. TEdit's Abbreviations and their Expanded Characters

| Abbreviation   | Expanded character name                   | Expansion Character |
|----------------|-------------------------------------------|---------------------|
| p              | Pilcrow<br>(proofreader's paragraph mark) | ¶                   |
| t              | Trademark                                 | ™                   |
| tm             | Trademark                                 | ™                   |
| r              | Registered trademark                      | ®                   |
| 1/3            | Built-up fraction                         | ⅓                   |
| x              | Times sign                                | ×                   |
| /              | Division sign                             | ÷                   |
| o (oh)         | Degrees sign                              | °                   |
| L              | Pound sterling sign                       | £                   |
| Y              | Yen sign                                  | ¥                   |
| +              | Plus-or-minus sign                        | ±                   |
| ^(shift-6)     | Up arrow (NS character)                   | ↑                   |
| ua             | Up arrow (NS character)                   | ↑                   |
|                | Down arrow (NS character)                 | ↓                   |
| da             | Down arrow (NS character)                 | ↓                   |
| <-             | Left arrow (NS character)                 | ←                   |
| la             | Left arrow (NS character)                 | ←                   |
| _ (underscore) | Left arrow (NS character)                 | ←                   |
| ->             | Right arrow (NS character)                | →                   |
| ra             | Right arrow (NS character)                | →                   |
| =              | Two-way arrows (NS characters)            | ↔                   |

### Put Submenu

The drag-through menu for the **Put** command now has the following entries:



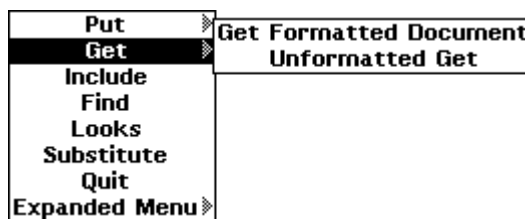
The **Put** command has a submenu that offers you several options for saving your file:

- Keep the formatting in the file. Use this **Put Formatted Document** option, which is the default, unless you have special requirements.
- Save the file as plain text, regardless of formatting. Using this **Plain-Text** option removes all of TEdit's formatting from the file, leaving plain text.
- Save TEdit files in an "old" format. This **Old-Format** option allows you save files in the format of a previous release of TEdit. This format option is provided for backward compatibility.

## Get Submenu

---

The drag-through menu for the **Get** command now has the following entries:



**Get** has a submenu that offers you the option of retrieving a formatted file (**Get Formatted Document**), or retrieving a file as though it were plain text, with most formatting information appearing as black rectangles ( • ).

---

## Clarified Paragraph Looks Menu Options

---

Both the menu options **New Page: Before After** and **Displaymode: Hardcopy** have expanded explanations.

### New Page: Before After

---

Sometimes a page break occurs so that the first paragraph on a page is marked with the **Before** command. In these cases, the text flows continuously from the previous page to this page; a blank page does not appear between them.

---

## Displaymode: Hardcopy

---

The Hardcopy displaymode command now works only when the text is printed in Interpress fonts.

---

## Clarified Page Layout Menu Options

---

When specifying text to appear before or after page numbers, you can only enter text in the brackets; image objects are not allowed.

---

You may now specify a landscape page layout.

---

In the page layout menu, Modern 10 MRR is now the default page number font instead of Gacha 10. Also, there is a global variable, TEDIT.DEFAULT.FOLIO.LOOKS, that you can set to be any character-looks specification acceptable to TEDIT.LOOKS. The default (i.e., if you don't specify one in the page layout menu) is taken from there.

If you have set page formatting in the past, the page-numbering font has been set as well (even if you specified nothing). This behavior continues, but the default is more sensible, and can be changed.

---

You may now number the first page of a TEdit file 0 (zero).

---

TEdit now preserves text before and after page numbers after a file is saved.

---

Using numbers with decimal points in the "Text before page number" field in the page-layout menu now works properly.

---

## Added Items to Programmer's Interface

---

The TEXTOBJ data structure has a correction, the TEDIT.INCLUDE, TEDIT.PARALOOKS and TEXTPROP functions are expanded, and the global variable TEDIT.KNOWN.FONTS is now documented.

---

## Corrected the AFTERQUITFN Property

---

(AFTERQUITFN *WINDOW TEXTSTREAM*) is an optional user-supplied Lisp function that is called after ending an editing session to perform any required cleanup. The *WINDOW* argument was omitted in the manual.

### **Corrected the TITLEMENUFN Property**

---

TEDIT.TITLEMENUFN is a window property, not a TEdit property as documented in the manual.

### **Corrected the TEXTOBJ Data Structure**

---

The data structure called TEXTOBJ has as its first field of interest WINDOW, not WINDOW as documented in the manual.

### **Expanded the TEDIT.INCLUDE Function**

---

TEDIT.INCLUDE now accepts optional START and END arguments that instruct it to restrict its attention to a portion of the TEdit file, treating this part as a separate file. This feature is useful when you require that several distinct TEdit documents reside within a single TEdit file, for example, for database applications. Each document can be formatted and extracted separately.

---

#### **CAUTION**

If you use START and END arguments with INCLUDE, and then format the entire TEdit file, you will lose the formatting.

---

#### **CAUTION**

TEDIT.INCLUDE and OPENTEXTSTREAM take optional arguments that let you take a document out of the middle of a file. This option requires that Lisp be able to determine the length of the file before it is read. Some file protocols (TCP FTP in particular) don't let Lisp do this. If you try to use this option with a file that resides at the other end of a TCP connection (or, more generally, on any device where you cannot tell the length of the file until you have read the whole file), it won't work. The result will be that your document will contain no characters.

---

### **Expanded the TEDIT.PARALOOKS Function**

---

TEDIT.PARALOOKS can be used to set NEWPAGEBEFORE, NEWPAGEAFTER, HARDCOPY, TYPE, SUBTYPE, REVISED, KEEP, STYLE, CHARSTYLES, and USERINFO parameters.

REVISED, if non-NIL, causes a revision bar to be printed one pica to the right of the right margin of the paragraph. It is a vertical bar 1 point wide from the top of the top line's ascent to the bottom of the bottom line's descent.

USERINFO can be used as a property list for saving information of interest to the user. It is generally used in a number of undocumented features (e.g. footnote support).

KEEP, STYLE, and CHARSTYLES are reserved for a future release.

## Expanded the TEXTPROP Function

You can now also use your own properties, but these properties are not saved with the document if you **Put** it.

## Added Documentation for a Global Variable

The following documentation should be added to TEdit's Global Variables.

TEDIT.KNOWN.FONTS

[Variable]

A list of available fonts that appear in the Character Looks menu. The list is in the form ((name-in-the-menu-1 'Real-font-name-1) (name-in-the-menu-2 'Real-font-name-2) ...), for example, ((Classic 'CLASSIC) (Times 'TIMESROMAN)).

## Changes to Programmer's Interface to TEdit

### STREAM AND TEXTOBJ

All public TEdit functions (non- \) that take a *TEXTOBJ* argument accept either a *TEXTOBJ* or a text *STREAM* as that argument's value.

## Changes, Additions and Corrections to TEdit functions

The function TEDIT.SINGLE.PAGEFORMAT is incorrectly documented in the Lisp Library. The following corrections should be noted: The arguments *PG#X*, *PG#Y*, and *PG#FONT* should be *PX*, *PY*, and *PFONT*, respectively.

The argument *PG#ALIGNMENT* should be *PQUAD*.

The order for the arguments, *TOP BOTTOM LEFT RIGHT* should be *LEFT RIGHT TOP BOTTOM*.

The argument *#COLS* should be *COLS*.

*INTERCOLSPACE* should be *INTERCOL*. And between the *INTERCOL* and *UNITS* arguments there is a *HEADINGS* argument.

The functions and its arguments look like:

```
(TEDIT.SINGLE.PAGEFORMAT PAGE#S? PX PY PFONT PQUAD LEFT RIGHT
 TOP BOTTOM COLS COLWIDTH INTERCOL
 HEADINGS UNITS PAGEPROPS PAPERSIZE) [Function]
```

*PAGE#S?* T if you want page numbers on this kind of page, else NIL.

*PX* The horizontal location of the page number, measured from the left edge of the paper. Negative values are measured from the paper's right edge.

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>PY</i>        | The vertical location of the baseline for the page numbers, measured from the bottom of the paper. Negative values are measured from the top of the paper.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <i>PFONT</i>     | The font to be used to display the page numbers. This can be any specification that is acceptable to TEDIT.LOOKS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <i>PQUAD</i>     | An atom that tells how the page number is to be aligned on the location specified by <i>PX</i> and <i>PY</i> . LEFT means the location is the lower-left corner of the page number. RIGHT means the location is the lower-right corner. CENTERED means the page number will be centered around the <i>PX</i> you specified.                                                                                                                                                                                                                                                                                                                                                                                            |
| <i>LEFT</i>      | The left margin—the distance from the left edge of the paper to the left edge of the first text column.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <i>RIGHT</i>     | The right margin—the distance from the right edge of the rightmost text column to the right edge of the paper.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>TOP</i>       | The top margin of the page—the distance from the top of the paper to the top of the first line of body text.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <i>BOTTOM</i>    | The bottom margin—the distance from the bottom of the last line of body text to the bottom of the paper.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| <i>COLS</i>      | Number of columns (default is one).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <i>COLWIDTH</i>  | The column width (default is to evenly divide the available space among the # <i>COLS</i> columns).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <i>INTERCOL</i>  | The space between the right edge of one column and the left edge of the next column. Defaults to evenly divide the space left after the columns are set up. If there is more than one column, one or the other of <i>COLWIDTH</i> and <i>INTERCOLSPACE</i> <i>must</i> be specified.                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <i>HEADINGS</i>  | A list of lists in the form of ( (HEADINGNAME XLOCATION YLOCATION ) (HEADINGNAME XLOCATION YLOCATION )<br>. . . (HEADINGNAME XLOCATION YLOCATION ) ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <i>UNITS</i>     | The units used in setting the values you specified. May be one of the atoms PICAS, IN, INCHES, CM, POINTS. Default is POINTS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <i>PAGEPROPS</i> | <p>A property list of extra information. Properties are STARTINGPAGE#, FOLIOINFO, and LANDSCAPE?.</p> <p>STARTINGPAGE# is the first page's number; it is ignored if this isn't the first page.</p> <p>FOLIOINFO is a list of information about page numbers, (FORMAT TEXTBEFORE TEXTAFTER). FORMAT can be one of ARABIC, LOWERROMAN, UPPERROMAN, or NIL (i.e., ARABIC). TEXTBEFORE is the text preceding the number, and TEXTAFTER is the text following the number.</p> <p>LANDSCAPE? determines if the document is printed in the usual vertical format or printed in landscape format (horizontally). If NIL the document is printed vertically, if non-NIL the document is printed landscape. Defaults to NIL.</p> |
| <i>PAPERSIZE</i> | Is one of LETTER, LEGAL, the metric paper sizes (A0, A1, A2 A3, A4, A5, B0, B2, B3, B4), or NIL (which defaults to letter size).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

---

TEDIT.GET accepts an open stream as the file to GET from. You may still pass it a TEXTOBJ, however.

(TEDIT.GET *STREAM FILE UNFORMATTED?*) [Function]

Performs the TEdit Get command, loading the text from *FILE* onto the editing stream *STREAM*—replacing the text that is being edited currently. If *FILE* is not supplied, the user will be asked for a file name. If *UNFORMATTED?* is non-NIL, *FILE* is treated as a plain-text document, and all of its contents are included—even TEdit formatting information.

---

You can now use TEDIT.PUT to store a TEdit document in the middle of a larger file (e.g., for saving TEdit documents as part of a database). The complete documentation is now as follows:

(TEDIT.PUT *STREAM FILE FORCENEW UNFORMATTED? OLDFORMAT?*) [Function]

Performs the TEdit Put command, saving the text from the text stream *STREAM* onto the file named *FILE*. If *FILE* is NIL, the user will be prompted for a file name. In this case, if *FORCENEW* is NIL, the user is offered the old file name as a default; if non-NIL, no default is given, forcing the user to specify a file name. If *UNFORMATTED?* is non-NIL, only characters are put in the file—no formatting. If *OLDFORMAT?* is non-NIL, the file will be written in the format used by the previous version of TEdit, for backward compatibility.

In order to store a TEdit document as part of another file, call TEDIT.PUT, passing an open stream on the file as the *FILE* argument. The stream should be open for output and positioned at the place you want TEdit to store the document (call this file pointer *START*). When TEDIT.PUT returns, the stream's end-of-file pointer will be just after the last byte in the newly-inserted document. Call this file pointer *END*. To subsequently retrieve the document from the middle of this other file, call OPENTEXTSTREAM on the file, passing the *START* and *END* pointers as the *START* and *END* arguments.

Note: When TEDIT.PUT returns, the stream will be open for INPUT.

---

The functions TEDIT.MOVE and TEDIT.COPY were not documented in Koto. They are:

(TEDIT.MOVE *FROM TO*) [Function]

*FROM* and *TO* are SELECTIONS. Moves the text described by *FROM* to the place described by *TO*, within the same text stream or between different text streams. The text described by *FROM* is deleted from its original location.

(TEDIT.COPY *FROM TO*) [Function]

*FROM* and *TO* are SELECTIONS. Copies the text described by *FROM* to the place described by *TO*, within the same text stream or between different text streams. The text described by *FROM* is not deleted in the *FROM* location.



## Changes in the Documentation of TEdit Functions

The following functions have had the documentation of their arguments changed to reflect what will appear if you do a ?= or evaluate ARGLIST on one of these functions. Arguments that were corrected are indicated by bold italics (***arg***). Please note that what changed was the documentation, not the way the functions operate or the values of the arguments themselves.

|                                                                                            |            |
|--------------------------------------------------------------------------------------------|------------|
| (TEDIT.SETSEL <i>LEAVECARETLOOKS OPERATION</i> <i>STREAM CH# LEN POINT PENDINGDELFLG</i> ) | [Function] |
| (COERCETEXTOBJ <i>STREAM TYPE OUTPUTSTREAM</i> )                                           | [Function] |
| (TEDIT.DELETE <i>STREAM SEL LEN</i> )                                                      | [Function] |
| (TEDIT.INCLUDE <i>STREAM FILE START END</i> )                                              | [Function] |
| (TEDIT.FIND <i>STREAM TARGETSTRING START# END# WILDCARDS?</i> )                            | [Function] |
| (TEDIT.GET.LOOKS <i>STREAM CH#ORCHARLOOKS</i> )                                            | [Function] |
| (TEDIT.PARALOOKS <i>STREAM NEWLOOKS SEL LEN</i> )                                          | [Function] |
| (TEDIT.COMPOUND.PAGEFORMAT <i>FIRST VERSO RECTO</i> )                                      | [Function] |
| (TEXTOBJ <i>STREAM</i> )                                                                   | [Function] |
| (TEXTSTREAM <i>STREAM</i> )                                                                | [Function] |
| (TEDIT.CARETLOOKS <i>STREAM LOOKS</i> )                                                    | [Function] |
| (TEDIT.NORMALIZECARET <i>STREAM SEL</i> )                                                  | [Function] |
| (COPYTEXTSTREAM <i>ORIGINAL CROSSCOPY</i> )                                                | [Function] |
| (TEDIT.PROMPTPRINT <i>TEXTSTREAM MSG CLEAR?</i> )                                          | [Function] |
| (TEDIT.SETSYNTAX <i>CHAR CLASS TABLE</i> )                                                 | [Function] |
| (TEDIT.GETSYNTAX <i>CH TABLE</i> )                                                         | [Function] |
| (TEDIT.SETFUNCTION <i>CHARCODE FN RTBL</i> )                                               | [Function] |
| (TEDIT.WORDGET <i>CH TABLE</i> )                                                           | [Function] |
| (TEDIT.WORDSET <i>CHARCODE CLASS TABLE</i> )                                               | [Function] |
| (TEDIT.INSERT.OBJECT <i>OBJECT STREAM CH#</i> )                                            | [Function] |

The following functions were previously documented as accepting a TEXTOBJ. They all still take a TEXTOBJ but they will now also accept a STREAM as the first argument.

|                                                                 |            |
|-----------------------------------------------------------------|------------|
| (TEDIT.FIND <i>STREAM TARGETSTRING START# END# WILDCARDS?</i> ) | [Function] |
| (TEDIT.GET.LOOKS <i>STREAM CH#ORCHARLOOKS</i> )                 | [Function] |
| (TEDIT.PARALOOKS <i>STREAM NEWLOOKS SEL LEN</i> )               | [Function] |
| (TEXTSTREAM <i>STREAM</i> )                                     | [Function] |
| (TEDIT.NORMALIZECARET <i>STREAM SEL</i> )                       | [Function] |
| (TEDIT.PROMPTPRINT <i>TEXTSTREAM MSG CLEAR?</i> )               | [Function] |

## New Features

For the benefit of NS file server users, TEdit now writes files of type TEDIT, instead of BINARY. As a result, LISTFILES and the FileBrowser are able to determine that the file is a TEdit file and call

TEdit to create the hardcopy. Previously, it was necessary that the TEdit file explicitly have the extension ".TEdit".

```
(OPENSTREAM file 'OUTPUT 'NEW ' ((TYPE TEDIT))).
```

This change is for formatted files only. Plain text files are still written as type TEXT. Also, on devices that don't support arbitrary file types (such as conventional mainframe file servers), the type TEDIT coerces to BINARY. Unfortunately, if you subsequently copy the file to an NS file server from such a device, the knowledge of its "true" file type is lost.

---

## A User's Guide to Sketch—Release Notes

---

The Medley release of Sketch includes several new features, many added in response to user's requests. A programmer's interface allows sketches to be created by programs. This interface is described in a separate document (*The Programmer's Interface to Sketch*.)

---

### Manipulating Sketch Elements

---

---

#### Adding and Deleting Control Points

---

Individual control points can now be added to and deleted from wires and curves.

---

#### Deleting Control Points

---

You now have the option to delete elements or delete a control point. Just select the **Delete** command, move the mouse cursor out through the grey arrow, then select the point to be deleted.

---

#### Defaults Command

---

---

##### Better Feedback for Creating Wires, Circles and Ellipses

---

Sketch now provides better feedback when you are creating circles, ellipses and wires. You are now prompted with an image of what the figure will look like if you release the left button. You can get the old feedback behavior (for example, if this is too slow) by selecting the **Feedback** subcommand from the **Defaults** submenu, then selecting the **Points only** subcommand from its submenu.

---

#### Arrowheads

---

A curved arrowhead shape was added and is now the default. Also, a command was added to the menu of arrowhead change operations that implements "look same" for arrowheads. To make the arrowheads on a collection of elements look the same: select **Change**; then, when prompted to select the elements to change, first select the element that has the desired arrowhead, then, in the same selection, add the elements that you want to look like the first one; then select the item **Arrowheads**, then the item **Both**, then the item **Same as First**.

---

#### Deleting Characters During Type-in

---

You can now delete characters by using the UNDO key, just as you would in TEdit. Type in a word or a phrase, then press the UNDO key, and the text will be deleted.

## Using Bit Maps in a Sketch

### Zooming Bitmaps

The bit image element provides a bitmap that zooms. Selecting the **Bit image** command from the command menu will prompt you for a region of the screen that will be inserted as a bit image into the sketch.

### Changing Bitmaps

When you apply a **Change** command to a bit image that it is being viewed at actual size, you will be prompted with the same menu as a bitmap image object. If the image is being displayed at other than original scale, you will be given the menu shown below.

Scaled bitmap operations

Perform edit operations on the source bitmap of this image.  
 Make the image shown be the source  
 Make the source be at this scale  
 Make the image shown be the source at the source scale  
 Save this image to be used as a source at this scale

*Menu of commands offered when **Change** command is applied to a bit image that is not at the original scale.*

## Freezing Sketch Elements

It is now possible to freeze elements, that is to make them unaffected by edit changes. Frozen elements will not have their control points highlighted (and hence cannot be selected) after an edit command has been selected. This provides a way to keep part of the figure fixed while editing on an overlapping part. It also reduces the number of control points. The **Freeze** command is a subcommand to the **Group** command. It will prompt you for a collection of elements that will then be frozen. Elements can be unfrozen by the **UnFreeze** command that is a subcommand to the **UnGroup** command.

## Aligning Sketch Elements

Sketch contains a set of commands to align elements. The main menu command **Align** prompts for a collection of control points and moves them so that they all line up with the leftmost one.

## Placing Multiple Copies of Elements

There is a new feature in Sketch that makes it much easier to place multiple copies of a collection of elements. While positioning the image of the elements during the **Copy** command, hold down the COPY key. A new copy of the elements will be positioned everytime a mouse button (left or right) is pressed and released, until either the image is placed completely outside the viewer or the COPY key is released before the mouse button is released.

## Making the Window Fit the Sketch

The **Fit to window** subcommand under the **Move View** command will zoom the sketch so that it just fits within the current window. It

has a sub-subcommand **Fit window to sketch** that will reshape the window so that the entire sketch (at the size shown) just fits within it. This is useful if you change a sketch that was edited from a document.

### Overlaying Figure Elements

---

Elements that have a filling property (boxes, text boxes, circles, polygons and closed curves) now have a mode property that determines how the filling should effect elements it covers. The option **Filling mode** now appears in the **Which aspect?** submenu.

### Changing How Elements Overlap

---

Elements have an order in which they are displayed. An element that is displayed early can be covered by elements layed down later. Thus, changing the order in which overlapping elements are displayed can effect the resulting image. The **Bury** command provides three subcommands to change the order in which elements are displayed.

The **Bury** command will prompt you to select an element or elements and will change their order so that they are displayed first. That is, they will appear underneath any other elements. If you select more than one element, they will all be displayed before any non-selected elements and their relative order maintained. The **Send to bottom** subcommand does the same thing as **Bury**.

The **Bring to top** command is a subitem to the **Bury** command. It will prompt you to select an element or elements and will change their order so that they are displayed last. That is, they will appear on top of any other elements. If you select more than one element, they will all be displayed after any non-selected elements and their relative order maintained.

The **Reverse order** command is a subitem to the **Bury** command. It will prompt you to select a collection of elements and will reverse their display orders. A special case is when two elements are selected. In this case the element positions are switched.

### Loading the Sketch Library Module in the 1186 Environment

---

The SKETCH executable files are too large to be contained on one floppy. The files are now distributed on two floppies: *Medley Library Floppy #3* and *Medley Library Floppy #4*. To load SKETCH, type the Interlisp exec command:

(FILESLOAD LOAD-SKETCH)

The LOAD-SKETCH function will copy all SKETCH files from #3; then prompt you to insert #4, and the remainder of the files will be copied.

---

## The Programmer's Interface

---

The programmer's interface allows Sketch to be used as a tool by other programs. It is documented in the *Programmer's Interface to Sketch* .

---

### New Behavior for the Get Command

---

The action of the **Get** command was changed to be consistent with the TEdit **Get** command. It now deletes any sketch elements that are in the sketch prior to the **Get** command. The affect of the old **Get** command is available as the **Include** command on a submenu to the **Get** command.

---

### Establishing Initial Defaults for Sketch

---

The variable SK.DEFAULT.FONT, if non-NIL, is used as the default font. If SK.DEFAULT.FONT is NIL, the default font (DEFAULTFONT) is used.

The following variables are used to establish the default setting for a new sketch. Descriptions of legal values can be found in the *Programmer's Interface to Sketch*. SK.DEFAULT.BRUSH is the default brush. SK.DEFAULT.ARROW.LENGTH is the default arrowhead size. SK.DEFAULT.ARROW.TYPE is the default type (one of LINE, CURVE, CLOSEDLINE or SOLID). SK.DEFAULT.ARROW.ANGLE is the default angle for arrowheads. SK.DEFAULT.TEXT.ALIGNMENT is the default text alignment. SK.DEFAULT.TEXTBOX.ALIGNMENT is the default textbox alignment. SK.DEFAULT.DASHING is the default dashing. SK.DEFAULT.TEXTURE, SK.DEFAULT.BACKCOLOR and SK.DEFAULT.OPERATION are combined to create the default filling.

## 1108 User's Guide Release Notes

---

### What to Look For

---

The *1108 User's Guide* was extensively reorganized and rewritten for the Lyric Release. This made it nearly identical to the *1186 User's Guide*. This section contains a summary of changes affecting 1108 environments with the Medley release. Details are described in update pages available for the *1108 User's Guide*.

In every 1108 chapter that requires use of Lisp expressions of any kind, there is a notice regarding the use of **IL:** and a suggestion that expressions, functions, and variables be typed into an Interlisp Exec.

### 4. File System

---

Medley will accept floppy names up to 40 characters in length. Some of the Lyric font floppies have names in excess of 40 characters. Medley truncates the floppy name to 40 characters if asked to read a Lyric floppy with a longer name. The function FLOPPY.NAME is used to name a floppy. When it is not given any arguments, it returns the name stored on the floppy disk. When it is given a *NAME* argument, the floppy name is set to *NAME*. The 40 character limitation holds for both 1108 and 1186 floppies.

---

The {DSK} device on the 1108 and 1186 now accepts a wider range of characters in file names. Almost any character in character set 0 is acceptable. Previously, if you tried to create a file whose name included, for example, an underscore, you would see a "FILE NOT FOUND" error.

---

The 1108 and 1186 file systems had a problem with large partitions which would manifest itself as "HARD DISK error - can't find file page" when accessing newly created files. This would only appear on logical volumes larger than 64K pages. This problem has been fixed.

---

The function FILENAMEFROMID is now implemented.

### 6. System Tools

---

In System Tools, it is no longer necessary to execute a **Floppy Info!** command before attempting a **List!**.

---

The Medley System Tool now displays an error message when an NS Domain or Organization name is more than the allowed 20 characters long.

---

The Medley System Tool now supports sysout and microcode installation using the TCP FTP protocol. This feature may be used by selecting the "TCP/FTP" device type in the main System Tool window. Update pages for the *1108 User's Guide* describing this feature, are included with the Medley release.

## **7. Input/Output**

---

Every time you allocate space on a floppy disk that has fewer than 200 free pages, a message is printed in the prompt window. That message gives an approximate number of free pages remaining after the allocation; it's intended to give you warning when your floppy is nearing full. The page count is correct only within +/- 2 pages because the message is printed in the course of the allocation, and the floppy's directory may grow when the new file is added to it .

## **8. Machine Diagnostics**

---

Medley Boot Diagnostics for the 1108 include changed floppy disk names and slight changes in the prompts for running diagnostics from floppies.



---

## 1186 User's Guide Release Notes

---

### What to Look For

---

The *1186 User's Guide* was extensively reorganized and rewritten for the Lyric Release. This section contains a summary of changes affecting 1186 environments with the Medley release. Details are described in update pages available for the *1186 User's Guide*.

In every 1186 chapter that requires use of Lisp expressions of any kind, there is a notice regarding the use of **IL:** and a suggestion that expressions, functions, and variables be typed into an Interlisp Exec.

### 1. Introduction

---

For Medley, the Xerox Lisp logo window has been changed to reflect the new name, Envos.

### 4. File System

---

Medley will accept floppy names up to 40 characters in length. Some of the Lyric font floppies have names in excess of 40 characters. Medley truncates the floppy name to 40 characters if asked to read a Lyric floppy with a longer name. The function FLOPPY.NAME is used to name a floppy. When it is not given any arguments, it returns the name stored on the floppy disk. When it is given a *NAME* argument, the floppy name is set to *NAME*. The 40 character limitation holds for both 1108 and 1186 floppies.

---

The {DSK} device on the 1108 and 1186 now accepts a wider range of characters in file names. Almost any character in character set 0 is acceptable. Previously, if you tried to create a file whose name included, for example, an underscore, you would see a "FILE NOT FOUND" error.

---

The 1108 and 1186 file systems had a problem with large partitions which would manifest itself as "HARD DISK error - can't find file page" when accessing newly created files. This would only appear on logical volumes larger than 64K pages. This problem has been fixed.

---

The function FILENAMEFROMID is now implemented.

### 5. Software Installation

---

The SKETCH executable files are too large to be contained on one floppy. The files are now distributed on two floppies: *Medley Library Floppy #3* and *Medley Library Floppy #4*. To load SKETCH, type the Interlisp exec command:

(FILESLOAD LOAD-SKETCH)

The LOAD-SKETCH function will copy all SKETCH files from #3; then prompt you to insert #4, and the remainder of the files will be copied.

## 6. System Tools

In System Tools, it is no longer necessary to execute a **Floppy Info!** command before attempting a **List!**.

The Medley System Tool now displays an error message when an NS Domain or Organization name is more than the allowed 20 characters long.

The Medley System Tool now supports sysout and microcode installation using the TCP FTP protocol. This feature may be used by selecting the "TCP/FTP" device type in the main System Tool window. Update pages for the *1186 User's Guide* describing this feature, are included with the Medley release.

## 7. Input/Output

Every time you allocate space on a floppy disk that has fewer than 200 free pages, a message is printed in the prompt window. That message gives an approximate number of free pages remaining after the allocation; it's intended to give you warning when your floppy is nearing full. The page count is correct only within +/- 2 pages because the message is printed in the course of the allocation, and the floppy's directory may grow when the new file is added to it.

The following function applies only to 1186 users:

( **dove.xor.cursor** *&optional xor-p*)

[Function]

If no argument is given, this function returns the current state of the 1186 cursor (nil implies an or'ing cursor, t an xor'ing cursor). If an argument is given, changes the state of the 1186 cursor appropriately.

## 8. Diagnostics

Medley Boot Diagnostics for the 1186 include changed floppy disk names and slight changes in the prompts for running diagnostics from floppies.

[This page intentionally left blank]

This section describes new features and enhancements that implement Common Lisp into the Lisp operating environment within the Medley release. This information supplements the *Common Lisp Implementation Notes*, Lyric release. Medley enhancements are indicated with revision bars in the right margin.

---

## New Features Since Lyric

---

The following description summarizes the new Common Lisp implementation features that have been added or changed since the Lyric release.

**New compiler Interface** -- The Medley compiler gives better progress reports and it is now possible to invoke the compiler on any definer (not just functions, as before).

**New Implementation of Defstruct** -- A new version of defstruct compiles more compactly and gives more options so that defstruct has at least as much functionality as the Interlisp record package.

**Adoption of features and clarifications suggested by the Common Lisp Cleanup Committee** -- Among other changes, the behavior of **append** on dotted lists is now better defined, and a new function **xcl:row-major-aref** has been added.

**Common Lisp Veneer on the Interlisp record package** -- A collection of macros that make the use of existing Interlisp datatypes more appealing has been added.

**Performance enhancements** -- A closure caching scheme now insures that repeated calls to symbol-functions of the same symbol will return EQ compiled-function objects.

New opcodes have been added for several common list functions, such as **member** and **assoc**.

---

## Common Lisp Definers

---

The Medley release contains a new implementation of definers and a reworking of the top level of the XCL Compiler. These represent upward compatible changes that have the effect of allowing the Common Lisp compiler to print out progress reports indicating which definer is currently being compiled. To receive the full benefit of these changes, recompile any file containing a **defdefiner** expression.

It is now possible to compile individual definers by using any of the following forms:

**Compile-Definer**

---

(xcl:compile-definer *name type*)

Compile and install the definer of type *type* named *name* .

**EXAMPLE:**

```
(xcl:compile-definer 'foo 'structures)
```

In this example, the definer will compile and install the structures definition of foo.

**Compile-Form**

---

(xcl:compile-form *form*)

Compile and evaluate *form*.

**EXAMPLE:**

```
(xcl:compile-form '(progn (defconstant c 1) (defun foo (a b) (+ c a b))))
```

In this example, the definer will compile and evaluate the progn using compile-file semantics.

**EXAMPLE:**

```
(xcl:compile-form '(with-collection (dotimes (i 10) (collect i))))
```

In this example, the definer returns:

```
(0 1 2 3 4 5 6 7 8 9)
```

**Define-File-Environment**

---

Rather than establishing **il:makefile-environment** props and **il:filetypes** on the root name of a file, you can define a file environment using the form:

(xcl:define-file-environment *filename* &key *readtable package base compiler*)

This produces an object of file-manager type **xcl:file-environments**. The *filename* can be either a string or a symbol. The rootname of the file is constructed by interning the *filename* in the Interlisp package. Puts the *compiler* argument (if any) under the **il:filetype** prop of the file rootname. Puts the *readtable*, *package* and *base* arguments (if any) under the **il:makefile-environment** prop of the file rootname. None of the arguments are evaluated. There are no defaults.

**EXAMPLE:**

```
(xcl:define-file-environment myfile :package "XCL-USER" :readtable "XCL" :compiler :compile-file)
```

In this example, *compile-file* is put under the **il:filetype** prop of *myfile*. The *readtable*, *XCL* and *compile* arguments are put under the **il:makefile-environment** prop of *myfile*.

**NOTE:** **xcl:define-file-environment** is a definer and hence will not be installed if **il:dfnflg** is **il:prop** or if a file is prop loaded.

## Site-Name Special Uses

The following special variables are defined and may be set in your init file to inform Common Lisp of site information:

**xcl:\*short-site-name\***

This variable is used in the function **short-site-name**.

**xcl:\*long-site-name\***

This variable is used in the function **long-site-name**.

### EXAMPLES:

```
(setq xcl:*short-site-name* "AIS")
```

```
(setq xcl:*long-site-name* "Artificial Intelligence Systems")
```

In these examples, (short-site-name) returns "AIS" and (long-site-name) returns "Artificial Intelligence Systems".

## Record Access

The Medley release contains several methods for accessing existing Interlisp records using Common Lisp syntax. These features help to integrate Interlisp and Common Lisp. The following sections describe these additions.

### Define-Record

(**xcl:define-record** *name interlisp-record-name*

&key *conc-name constructor predicate fast-accessors*) [Definer]

Creates a structures object named by the symbol *name* that provides Common Lisp accessors, setters, predicates and constructors for the Interlisp record named by the symbol *interlisp-record-name*. The Interlisp record must be defined before the **xcl:define-record** expression is evaluated. The keyword arguments are treated as in defstruct. The package of constructed names is taken from the value of **\*package\*** at the time of evaluation (as in defstruct). The system contains no predeclared **define-records**.

### EXAMPLE:

The form:

```
(xcl:define-record menu il:menu)
```

allows you to write:

```
(menu-items foo) and (setf (menu-items foo) fie)
```

rather than:

```
(il:fetch (il:menu il:items) il:of foo)
```

### Record-Fetch

---

(xcl:record-fetch *record field object*) [Macro]

Evaluates *object*. Does not evaluate *record* and *field*. Both *record* and *field* must be symbols. Symbols with the same p-names are interned in the Interlisp package and are used to construct an **il:fetch** form. **xcl:record-fetch** may be used with **setf** and expands to the suitable replace form.

### Record-FFetch

---

(xcl:record-ffetch *record field object*) [Macro]

Similar to **xcl:record-fetch**, but an **il:ffetch** form is generated instead. Evaluates *object*. Does not evaluate *record* and *field*. Both *record* and *field* must be symbols. Symbols with the same p-names are interned in the Interlisp package and are used to construct an **il:ffetch** form. Ffetch may be used with **setf** and expands to the suitable freplace form.

### Record-Create

---

(xcl:record-create *record &rest keyword-pairs*) [Macro]

Evaluates the second element of each pair. Does not evaluate *record* (*record* must be a symbol). A symbol with the same p-name is interned in the Interlisp package and used to construct an **il:create** form. The rest of the arguments form keyword pairs. The first element of each pair should be a symbol such that a symbol with the same p-name exists in the Interlisp package and names either a valid slot for this record or is one of **:using**, **:copying**, **:reusing**, or **:smashing**.

### Array Reference

---

(xcl:row-major-aref *array index*) [Function]

Returns the element of *array* given by the row-major-index *index*. The array can be of any dimension. This function can be used with **setf**.

### Shadowing of Global Macros

---

The XCL Compiler now properly handles shadowing of global macros by lexical functions. In the Lyric Compiler, lexical functions defined with **flet** did not shadow global definitions of the same name. This has been fixed in Medley.

### Evaluating Load-time Expressions

---

The XCL Compiler now handles **il:loadtimeconstant** correctly. The new Compiler substitutes the entire expression for each reference to the value of a load-time constant. There are potential problems if the code depends on the expression being evaluated exactly once, e.g. if it contains (IDATE).

### Common Lisp Defstruct Options

---

The Medley release contains a new implementation of **defstruct** that offers greater compiled-code compaction, and several new extensions that increase efficiency. This implementation introduces functionality that allows **defstruct** to parallel the Interlisp

record module in flexibility. These features also help to integrate Interlisp and Common Lisp. The following sections describe these additions.

### Defstruct Options

#### :inline

Can be one or both of :accessor and :predicate or t, implying '(:accessor :predicate) or nil, implying no optimizations allowed or :only, implying all accessors and the predicate will be inline only and not funcallable (not usable with the Lisp primitive "funcall"). The default is '(:accessor :predicate).

Copiers and constructors are never inline. The option (:inline :only) implies that no funcallable accessors will be generated (similarly, the predicate, if any, will not be funcallable).

#### :fast-accessors

Can be t or nil. t implies inline accessors will not type check. The default is nil.

Note that funcallable accessors (if any), always type check, if possible.

**NOTE:** This represents a change from the Lyric implementation, which allowed specification of a list of slot names that had fast inline accessors.

#### :template

Can be t or nil, t implies that no datatype will be instantiated. (:template t) implies no :type option. The default is nil.

Templated defstructs have no predicates, copiers or constructs. It is an error to supply any such option in combination with (:template t). Templated defstructs are intended to be used as are IL:blockrecord's. It is possible for a templated defstruct to include another templated structure, but it is an error for a standard defstruct to include a templated structure.

Funcallable accessors (accessors that may be used with the Lisp primitive "funcall") share code with suitable closure templates if the defstruct is compiled with the XCL Compiler. Byte compiled defstructs still generate explicit defun's for all funcallable accessors.

### Defstruct Slot Options

#### :type

The following specialized types are recognized:

(unsigned-byte {1 - 16})

(signed-byte {16, 32})

float, etc.

(member t nil)

il:fullpointer

il:xpointer

il:fullxpointer



### Warning When Using Defstruct

---

Defstruct automatically generates a number of auxilliary functions without checking whether redefining those functions will affect the system. To avoid redefining key functions, you should be aware of the names that will be used. For example:

Do not attempt to define a Structure named TREE. This use of Defstruct implicitly redefines the built-in Common Lisp function COPY-TREE, which renders your system inoperable.

If you have already tried to define a (DEFSTRUCT TREE A B) structure by mistake, you will need to reload your system.

### Macros for Collecting Objects

---

#### xcl:with -collection

---

(xcl:with-collection &body forms) [Macro]

(xcl:collect form) [Macro]

This pair of macros is provided for efficiently collecting objects into a list. In Common Lisp, there is no direct facility provided for doing this, so one must either push objects onto a list, then reverse it, or maintain a tail pointer to the list and use **rplacd** to add new items. The latter has an efficient implementation in Xerox Common Lisp, and **xcl:with-collection** is provided to take advantage of it.

Lexically within the body of an **xcl:with-collection**, the macro **xcl:collect** is defined. It will append the value of its argument to the end of the list being collected. The value of **xcl:with-collection** is the collected list.

**xcl:collect** may be used inside of functions passed as arguments to other functions.

#### EXAMPLE:

```
(xcl:with-collection
 (maphash
 #'(lambda (key val)
 (when (interesting-p val) (xcl:collect key)))
 the-hash-table))
```

will collect a list of all the "interesting" keys in the order that they were encountered.

It is an error to use **xcl:collect** outside the scope of an **xcl:with-collection**. Proper lexical nesting is observed, so an instance of **xcl:collect** applies to the most deeply nested **xcl:with-collection** that is found in.

## Macros for Writing Macros

### **xcl:once-only**

**(xcl:once-only** (*{ variable }*\*) *&body forms*)

[Macro]

This macro is provided to aid in writing macros. **xcl:once-only** helps solve the problem of multiple evaluation of subforms of a macro.

#### EXAMPLE:

```
(defmacro test (reference form)
 `(setf ,reference (cons ,form ,form)))
```

This example has the problem that **form** will be evaluated twice. To avoid this, one might instead write:

```
(defmacro test (reference form)
 (let ((value (gensym)))
 `(let ((,value ,form))
 (setf ,reference (cons ,value ,value)))))
```

This solves the problem of multiple evaluation, but introduces some others. If **form** is in fact something simple, like a reference to a variable or a literal, there was no need to create the temporary variable, thus "wasting" a symbol. This can be extremely important in Xerox Common Lisp as symbol space is limited and symbols are never reclaimed. If there are many temporary values to be computed, the macro definition becomes cluttered with calls to **gensym** that obscure the essence of the code.

**xcl:once-only** helps solve these problems. For each of the variables listed, **xcl:once-only** determines if its value (at macroexpansion time) is simple: a symbol or a literal. If it is, appearances of that variable in the macroexpansion will remain unchanged. If it is not, the macroexpansion will contain code to store the value in a temporary **gensym**'ed variable and use that variable in the macroexpansion. Thus, the example could be written as

```
(defmacro test (reference form)
 (xcl:once-only (form)
 `(setf ,reference (cons ,form ,form))))
```

Then `(test (aref the-array x) y)` will expand to something like

```
(setf (aref the-array x) (cons y y))
```

while `(test (aref the-array x) (random-form))` will expand to something like

```
(let ((#:g377 (random-form)))
 (setf (aref the-array x) (cons #:g377 #:g377)))
```

Note that **xcl:once-only** does not attempt to preserve order of evaluation. If this is important then you will still have to create temporary variables yourself.

## Common Lisp Append Datatypes

---

A clarification adopted by X3J13 involves the behavior of the APPEND function with non-lists. The cdr of the last cons in any but the last argument given to APPEND is discarded (whether NIL or not) when preparing the list to be returned. In the case where there is no last cons (i.e., the argument is not a list) in any but the last list argument, the entire argument is effectively ignored. In this situation, if the last argument is a non-list, the result of APPEND can be a non-list. NB: APPEND and COPY-LIST now produce different results for non-lists.

### EXAMPLE:

```
(append '(a b c . d) '())
```

produces the result:

```
(a b c)
```

### EXAMPLE:

```
(append '(a b . c) '() 3)
```

produces the result:

```
(a b . 3)
```

### EXAMPLE:

```
(append 3 17)
```

produces the result:

```
17.
```

## Closure Cache

---

The Medley sysout contains a closure cache that provides increased time and space efficiency. Less new memory is allocated because repeated calls to symbol-function of the same symbol now will cons exactly one closure object. Repeated calls to symbol-function of the same symbol now return EQ-compiled function objects.

## Symbols and Packages

---

### Pkg-goto and In-package

---

PKG-GOTO is now a synonym for IN-PACKAGE. The PKG-GOTO function can be used to change packages in an exec.

PKG-GOTO takes one argument, which can be either a double-quoted string, a symbol, or a package structure. This function is used to set package in an exec.

(xcl:pkg-goto package-name &key nicknames use) [Function]

PKG-GOTO operates like IN-PACKAGE, but asks for confirmation if a new package is being created. The function is useful at the top level in the exec, to avoid creating new packages when a name is misspelled.

### Defpackage Export argument

---

Defpackage's EXPORT argument now accepts strings. Optionally, strings can be given to :EXPORT instead of symbols. This is recommended when defpackage is used in the makefile-environment property of a file. The strings are interned in the package being defined and then exported.

## Debugging Tools

### Breaking

Even with HELPDEPTH set to zero, some errors do not cause a break. In Koto and the old Interlisp execs in Lyric, the workaround is:

```
(SETTOPVAL 'HELPFLAG 'BREAK!)
```

In Medley and Lyric's new execs, HELPFLAG is bound but not continually reset. The workaround:

```
(SETQ HELPFLAG 'BREAK!)
```

affects the current exec until the next time you call RESET (or control-D). If you want the change in HELPFLAG to be seen by other processes, you still need to use SETTOPVAL, and RESET any execs in which you want to see the effect.

For related information, see the Medley error system variable XCL:\*BREAK-ON-SIGNALS\* described in Appendix E.

### Advising

In Lyric, putting a second piece of advice on a function caused the system to believe that the function was in fact not advised, so any further advice threw out the already existing advice. This has been fixed. In Medley, the correct list entries are made regardless of whether the function was previously advised.

In Lyric, loading a file with advice caused multiple instances of the advice to be instantiated. To prevent this, ADVISE is now changed in Medley in the following way: When a new piece of advice is put on a function, the system examines the already existing advice to see if the same advice is already there. If so, the old advice is removed before adding the new advice. Sameness is determined by a test similar to CL:EQUALP, except that case distinctions are significant in strings and characters. The priority and location of the advice is taken into account when determining the "sameness." This makes it possible, for instance, to have identical advice be both :FIRST and :LAST.

Advice is no longer replicated when loaded more than once.

The debugger and inspector now display interpreted lexical closures conveniently. Displayed lexical closure contents include the function contained, and any lexical bindings in the closure. Compiled closures are not conveniently inspectable. Common Lisp eval stack frames show their associated lexical environment in a similar manner.

The :when option to XCL:BREAK-FUNCTION no longer causes the broken function to return NIL when the break is not taken. The correct values are returned.

### Argument Names Displayed for Interpreted Functions

---

In the debugger, the frame inspector window will now display the argument names for interpreted Common Lisp functions. Previously, it gave them pseudonames "arg0" "arg1" etc.

### Lexical Variables Evaluated by Debugger

---

The debugger EVAL command now evaluate expressions in the lexical environment --i.e., you can evaluate an expression and use variables that are lexically bound in your code. Only the lexical environment at the point of the break can be evaluated. You can't presently back up to any given lexical environment.

#### EXAMPLE:

```
(defun fact(x) (if(= 1 x) nil (*x(fact(1-x)))))
(fact 4)
```

;; breaks. if you then type

```
EVAL x
2
```

### Pathname Component Fixed in FS-ERROR

---

In Lyric, only one of the three FS-ERROR conditions was passed a pathname component, resulting in the File Cacher not knowing which file had the error, or resulting in pathname being lost when PROTECTION VIOLATION or FILE SYSTEM RESOURCES EXCEEDED were signaled. This problem occurred most noticeably in Lyric when Interlisp errors were translated to XCL. This condition has been fixed in Medley. FS-ERROR now correctly receives all the pathname components.

## Compiler Optimizations

---

### Warning when using LABELS construct

---

In Lyric, use of the LABELS construct generated circular structure that would not get collected. Interpreted, a LABELS construct always creates this non-collectible structure. Compiled, such structure would be created if there were non-tail-recursive or mutually referencing subfunctions. The values of any closed-over variables are captured by this structure and thus also not collected, potentially causing large storage leaks. The latter situation has been relieved somewhat for Medley.

In Medley, the unavoidable circularity has been reduced to include only the mutually referencing functions, but not any of the other data that they access. Thus, the uncollectable structure is created only when a new copy of the code blocks are created, such as by compiling the function containing the LABELS rather than each time that function is called.

### COMS added to dfasl files

---

The Medley compiler has been modified to better handle the il:define-file-info, and defpackage forms. Now, loading a dfasl file

is not implicitly SYSLOAD. Since the file COMS for the file is now included in the dfasl, that file will be noticed by the file manager unless the load is explicitly SYSLOAD. (SYSLOADing of compiled lcom and dfasl files is recommended.)

In Lyric, dfasls of file manager files did not contain the COMS of the file. In Medley, COMS are present in dfasl files, just as they are in lcom files. As with lcom files, the COMS will not be loaded when the LDFLG argument to LOAD is SYSLOAD, nor will the name of the file be added to FILELST, but instead will be added to SYSFILES.

**Note:** We discourage loading either sort of compiled file (lcom or dfasl) with any value for LDFLG but SYSLOAD. Unless you intend to edit a file, you should always load it SYSLOAD. Even when you intend to edit it, it is usually preferable to SYSLOAD it and then load the source PROP. If there are too many source files for this to be practical, we recommend use of the WHERE-IS Library module.

While the location of definitions is made known to the edit interface when files are loaded, it can be very inefficient when files are not SYSLOADed. If, for example, you load ten compiled files with LDFLG=NIL and then evaluate (ED 'FOO), then the COMS of all ten files must be searched for definitions of each manager type with name FOO. With forty manager types this comes to 400 parses of COMS -- a time-consuming operation. If you instead load the compiled files SYSLOAD and the sources PROP, then no COMS need be searched, as checking for definitions of each manager type is sufficient.

### Loadflg argument

The Medley release contains a new keyword argument to **cl:load**.

(cl:load filename &key verbose print if-does-not-exist loadflg)

The *loadflg* argument follows the semantics of the loadflg argument to **il:load**, with the exception that the loadflg argument will always be interned in the Interlisp package.

#### **EXAMPLE:**

```
(cl:load "Mycompiled-file.dfasl" :loadflg :sysload)
```

In this example, "Mycompiled-file.dfasl" will load without the file manager noticing that file.

**Note:** As explained in the previous section, we discourage loading either sort of compiled file (lcom or dfasl) with any value for *ldflg* but SYSLOAD.

### Changes in CL:MAP, CL:WRITE-STRING, CL:COERCE, CL:GENSYM and IL:DEFERREDCONSTANT

In Lyric, a compiled call to CL:MAP that had been used for effect would occasionally cons up a new list anyway. It would fail in the case that the first argument was a constant that evaluated to NIL, but not NIL itself, e.g. 'NIL. This has been fixed and no longer occurs in Medley.

CL:WRITE-STRING is now twice as fast and creates no new structure.

CL:COERCE now correctly returns the original object in all cases where Common Lisp and Lisp require it.

The CL Compiler now compiles CL:GENSYM properly.

IL:DEFERREDCONSTANT is now handled correctly by the XCL compiler.

ADD.PROCESS no longer coerces the process name to a symbol. Rather, process names are treated as case-insensitive strings. Thus, you can use strings for process names, and when typing process commands to an exec, you need not worry about getting the alphabetic case correct.

---

### Compiler keeps Special &REST arguments

---

The CL Compiler now retains special &REST arguments. The Lyric compiler threw away special &REST arguments. This has been fixed in the Medley CL Compiler.

---

### Compiler ignores TEdit formatting

---

COMPILE-FILE will now ignore TEdit formatting, but only if TEdit is loaded.

---

### Compiler notices Tail-recursive Lexical Functions

---

The XCL Compiler now performs tail recursion elimination on FLETed lexical functions.

---

### Compiler Error Message "BUG: Inconsistent stack depths seen"

---

You may occasionally see this error message while compiling. Normally, error messages from the compiler beginning with "BUG" indicate an internal compiler error. In this particular case, the compiler error may reflect an error in the code you are compiling.

There is currently no compile-time argument checking. The compiler performs an optimization that turns a tail-recursive function call into a jump back to the beginning of the function. If this tail-recursive call has the wrong number of arguments, the stack modeler in the assembler will detect this as inconsistent stack depths, leading to the above error message.

#### EXAMPLE:

```
(defun bad-length (x n)
 (if (endp x) n (bad-length (cdr x))))
```

Compiling this form will result in the error "BUG: Inconsistent stack depths seen." The recursive call to bad-length has only one argument, but the function expects two.

Thus, if you see this error message, you should check for tail-recursive function calls with the wrong number of arguments.

---

### Format ~C and WRITE-CHAR

---

In accordance with a recommendation of X3J13, the ~C FORMAT operation with no modifiers now behaves exactly the same as WRITE-CHAR for characters with no bits. The Medley release of XCL conforms to this; the Lyric release did not. If you need to obtain the Lyric behavior of ~C, use ~:C.

**WITH-OUTPUT-TO-STRING and WITH-INPUT-FROM-STRING**

---

For consistency with WITH-OPEN-STREAM and WITH-OPEN-FILE, WITH-OUTPUT-TO-STRING and WITH-INPUT-FROM-STRING now close the stream on exit from the form. WITH-OUTPUT-TO-STRING is now significantly faster when writing long strings.



[This page intentionally left blank]

---

**Replace this page with  
Table of Contents  
tab**

---

---

**Replace this page with  
Preface  
tab**

---

---

**Replace this page with  
1. Introduction  
tab**

---

---

**Replace this page with  
2. Notes and Cautions  
tab**

---

---

**Replace this page with**  
**3. Integration of Languages**  
**tab**

---

---

**Replace this page with  
4. Changes to Interlisp-D  
tab**

---

---

**Replace this page with  
5. Library Modules  
tab**

---



---

**Replace this page with  
6. User's Guides  
tab**

---

---

**Replace this page with**  
**7. Common Lisp Implementation**  
**tab**

---

---

**Replace this page with  
A. The Exec  
tab**

---

---

**Replace this page with  
B. SEdit  
tab**

---

---

Replace this page with  
C. ICONW  
tab

---

---

**Replace this page with  
D. Free Menu  
tab**

---

---

**Replace this page with  
E. Error System  
tab**

---

---

**Replace this page with  
Index  
tab**

---



**POINT SIZE:****FONT:** MODERN

|                                                     |         |                 |                       |  |
|-----------------------------------------------------|---------|-----------------|-----------------------|--|
| Table of Contents<br>3. Integration of<br>Languages | Preface | 1. Introduction | 2. Notes and Cautions |  |
|-----------------------------------------------------|---------|-----------------|-----------------------|--|

|                              |                    |                  |                                  |  |
|------------------------------|--------------------|------------------|----------------------------------|--|
| 4. Changes to<br>A. The Exec | 5. Library Modules | 6. User's Guides | 7. Common Lisp<br>Implementation |  |
|------------------------------|--------------------|------------------|----------------------------------|--|

|                   |          |              |                 |  |
|-------------------|----------|--------------|-----------------|--|
| B. SEdit<br>Index | C. ICONW | D. Free Menu | E. Error System |  |
|-------------------|----------|--------------|-----------------|--|

# TABS FOR MEDLEY RELEASE NOTES MANUAL (2-1/4" BINDER)

**TYPE: MAJOR**

**TAB SIZE: 2-3/8"**

**NO. TABS PER BANK: 5**

**NO. OF BANKS: 3**

*Point size: 10*

*Font : Modern*

**COLOR OF TABS: GREEN background, BLACK lettering**

*Point size: 10*

*Font : Modern*

x Tick marks indicate hole placement:  
3-hole punch, 5/16" diameter holes

1st hole centered: 1 3/16" from top of page, 7/16" in from side  
2nd hole centered: 5 7/16" from top of page, 7/16" in from side  
3rd hole centered: 9 11/16" from top of page, 7/16" in from side

## BANK 1

|   |  |
|---|--|
| x |  |
|   |  |
| x |  |
|   |  |
| x |  |

Table of Contents

Preface

1. Introduction

2. Notes and Cautions

3. Integration of Languages

## BANK 2

|   |  |
|---|--|
| x |  |
|   |  |
| x |  |
|   |  |
| x |  |

4. Changes to Interlisp-D

5. Library Modules

6. User's Guides

7. Common Lisp Implementation

A. The Exec

## BANK 3

x

B. SEdit

C. ICONW

x

D. Free Menu

E. Error System

x

Index

**POINT SIZE:** <sup>12</sup>

**FONT:** MODERN

|                                                     |                    |                  |                                  |  |
|-----------------------------------------------------|--------------------|------------------|----------------------------------|--|
| Table of Contents<br>3. Integration of<br>Languages | Preface            | 1. Introduction  | 2. Notes and Cautions            |  |
| 4. Changes to<br>Interlisp-D<br>A. The Exec         | 5. Library Modules | 6. User's Guides | 7. Common Lisp<br>Implementation |  |
| B. SEdit<br>Index                                   | C. ICONW           | D. Free Menu     | E. Error System                  |  |

**TAB SPECIFICATIONS for**  
**LISP RELEASE NOTES, MEDLEY RELEASE**  
**and**  
**LISP LIBRARY MODULES, MEDLEY RELEASE**

**TABS**

**Tab Paper Size** 8 1/2 by 11 inches (excluding the actual tab)

**Stock Type** Ledger; matte finish

**Stock Weight** 110 pound

**Stock Color** White

**Tab Coating** Mylar

**Mylar Color**  
PMS 422-C coordinated gray

**Holes** 3-hole punch, 5/16" diameter holes

**Hole Reinforcement** clear mylar strip

**Hole Placement** **1st hole centered:** 1 3/16" from top of page, 7/16" in from side  
**2nd hole centered:** 5 7/16" from top of page, 7/16" in from side  
**3rd hole centered:** 9 11/16" from top of page, 7/16" in from side

**Tab Cut Requirements**

---

**Number of Tab Cuts** 5

**Size of Cuts (Length)** 2 3/8"

**Depth of Tab Cut** 1/2 inch

**Cut Shapes** straight edge, curved top, curved bottom

**Print Specifications**

---

**One or Both Sides** Both sides

**Font** Modern/Optima

**Point Size** 10

**Ink Color** Black

***Type Spacing***      Centered

**PACKAGING INSTRUCTIONS:**      Shrink wrap with text

**TOTAL NUMBER COPIES:**      500

# LISP RELEASE NOTES



Address comments to:  
ENVOS  
User Documentation  
1157 San Antonio Rd.  
Mountain View, CA 94043  
415-966-6200

---

## LISP RELEASE NOTES

Medley Release 1.0

400006

September 1988

Copyright © 1988 by ENVOS Corporation.

All rights reserved.

Envos is a trademark of Envos Corporation.

Medley is a trademark of Envos Corporation.

Xerox® is a registered trademark of Xerox Corporation.

Sun® is a registered trademark of Sun Microsystems Inc.

DEC®, VAX®, VMS®, and VT100® are registered trademarks of Digital Equipment Corporation.

UNIX® is a registered trademark of AT&T Bell Laboratories.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

---

The information in this document is subject to change without notice and should not be construed as a commitment by Envos Corporation. While every effort has been made to ensure the accuracy of this document, Envos Corporation assumes no responsibility for any errors that may appear.

Text was written and produced with Envos formatting tools using Xerox printers to produce text masters. The typeface is Modern.

# TABLE OF CONTENTS

---

|                                                              |       |
|--------------------------------------------------------------|-------|
| <b>Preface</b>                                               | xvii  |
| How the Release Notes are Organized                          | xvii  |
| Notational Conventions                                       | xviii |
| How to Use the Release Notes                                 | xviii |
| Related Literature                                           | xix   |
| <b>1. Introduction</b>                                       | 1-1   |
| Summary of Medley Changes                                    | 1-1   |
| <b>2. Notes and Cautions</b>                                 | 2-1   |
| Changes and Cautions in the Medley Release                   | 2-1   |
| Changes and Cautions in the Lyric Release                    | 2-1   |
| <b>3. Common Lisp/Interlisp-D Integration</b>                | 3-1   |
| Chapter 2 Litatoms                                           | 3-1   |
| Section 2.1 Using Litatoms as Variables                      | 3-2   |
| Section 2.3 Property Lists                                   | 3-2   |
| Section 2.4 Print Names                                      | 3-2   |
| Section 2.5 Characters                                       | 3-3   |
| Chapter 4 Strings                                            | 3-3   |
| Chapter 5 Arrays                                             | 3-3   |
| Chapter 6 Hash Arrays                                        | 3-4   |
| Chapter 7 Numbers and Arithmetic Functions                   | 3-4   |
| Section 7.2 Integer Arithmetic                               | 3-4   |
| Chapter 10 Function Definition, Manipulation, and Evaluation | 3-5   |
| Section 10.1 Function Types                                  | 3-5   |
| Section 10.6 Macros                                          | 3-5   |
| Section 10.6.1 DEFMACRO                                      | 3-5   |
| Chapter 11 Stack Functions                                   | 3-5   |
| Section 11.1 The Spaghetti Stack                             | 3-5   |
| Chapter 12 Miscellaneous                                     | 3-6   |
| Section 12.4 System Version Information                      | 3-6   |
| Section 12.8 Pattern Matching                                | 3-6   |
| Chapter 13 Interlisp Executive                               | 3-7   |
| Chapter 14 Errors and Breaks                                 | 3-9   |
| Section 14.3 Break Commands                                  | 3-9   |
| Section 14.6 Creating Breaks with BREAK1                     | 3-9   |
| Section 14.7 Signalling Errors                               | 3-9   |



## TABLE OF CONTENTS

---

|                                                                 |      |
|-----------------------------------------------------------------|------|
| Section 14.8 Catching Errors                                    | 3-10 |
| Section 14.9 Changing and Restoring System State                | 3-11 |
| Section 14.10 Error List                                        | 3-11 |
| Chapter 15 Breaking Functions and Debugging                     | 3-13 |
| Section 15.1 Breaking Functions and Debugging                   | 3-13 |
| Section 15.2 Advising                                           | 3-14 |
| Chapter 16 List Structure Editor                                | 3-15 |
| Switching Between Editors                                       | 3-16 |
| Packages                                                        | 3-16 |
| Starting a Lisp Editor                                          | 3-16 |
| Mapping the Old Edit Interface to ED                            | 3-18 |
| Editing Values Directly                                         | 3-18 |
| Section 16.18 Editor Functions                                  | 3-19 |
| Chapter 17 File Package                                         | 3-19 |
| Reader Environments and the File Manager                        | 3-20 |
| Modifying Standard Readtables                                   | 3-22 |
| Programmer's Interface to Reader Environments                   | 3-23 |
| Section 17.1 Loading Files                                      | 3-24 |
| Integration of Interlisp and Common Lisp LOAD Functions         | 3-24 |
| Section 17.2 Storing Files                                      | 3-25 |
| Section 17.8.2 Defining New File Manager Types                  | 3-26 |
| Definers: A New Facility for Extending the File Manager         | 3-26 |
| Chapter 18 Compiler                                             | 3-31 |
| Warning when Loading Compiled Files                             | 3-32 |
| Warning with Declarations                                       | 3-32 |
| Section 18.3 Local Variables and Special Variables              | 3-33 |
| Chapter 19 Masterscope                                          | 3-33 |
| Chapter 21 CLISP                                                | 3-33 |
| Chapter 22 Performance Issues                                   | 3-36 |
| Section 22.3 Performance Measuring                              | 3-36 |
| Chapter 24 Streams and Files                                    | 3-37 |
| Section 24.15 Deleting, Copying, and Renaming Files             | 3-38 |
| Chapter 25 Input/Output Functions                               | 3-38 |
| Variables Affecting Input/Output                                | 3-38 |
| Integration of Common Lisp and Interlisp Input/Output Functions | 3-40 |
| Section 25.2 Input Functions                                    | 3-40 |
| Section 25.3 Output Functions                                   | 3-41 |
| Printing Differences Between IL:PRIN2 and CL:PRIN1              | 3-42 |
| Internal Printing Functions                                     | 3-42 |
| Printing Differences Between Koto and Lyric                     | 3-42 |

|                                                                     |      |
|---------------------------------------------------------------------|------|
| <u>Bitmap Syntax</u>                                                | 3-43 |
| <u>Section 25.8 Readtables</u>                                      | 3-43 |
| <u>Differences Between Interlisp and Common Lisp Readtables</u>     | 3-44 |
| <u>Section 25.8.2 New Readtable Syntax Classes</u>                  | 3-45 |
| <u>Additional Readtable Properties</u>                              | 3-45 |
| <u>Section 25.8 Predefined Readtables</u>                           | 3-47 |
| <u>Koto Compatibility Considerations</u>                            | 3-48 |
| <u>Specifying Readtables and Packages</u>                           | 3-48 |
| <u>The T Readtable</u>                                              | 3-48 |
| <u>PQUOTE Printed Files</u>                                         | 3-49 |
| <u>Back-Quote Facility</u>                                          | 3-49 |
| <u>Chapter 28 Windows and Menus</u>                                 | 3-49 |
| <u>Section 28.5.1 Menu Fields</u>                                   | 3-49 |
| <b>4. Changes to Interlisp-D in Lyric/Medley</b>                    | 4-1  |
| <u>Chapter 3 Lists</u>                                              | 4-1  |
| <u>Section 3.2 Building Lists From Left To Right</u>                | 4-1  |
| <u>Section 3.10 Sorting Lists</u>                                   | 4-1  |
| <u>Chapter 6 Hash Arrays</u>                                        | 4-1  |
| <u>Section 6.1 Hash Overflow</u>                                    | 4-2  |
| <u>Chapter 7 Integer Arithmetic</u>                                 | 4-2  |
| <u>Section 7.3 Logical Arithmetic Functions</u>                     | 4-3  |
| <u>Section 7.5 Other Arithmetic Functions</u>                       | 4-3  |
| <u>Chapter 8 Record Package</u>                                     | 4-3  |
| <u>Chapter 9 Conditionals and Iterative Statements</u>              | 4-3  |
| <u>Section 9.2 Equality Predicates</u>                              | 4-3  |
| <u>Section 9.8.3 Condition I.s. oprs</u>                            | 4-3  |
| <u>Chapter 10 Function Definition, Manipulation, and Evaluation</u> | 4-4  |
| <u>Section 10.2 Defining Functions</u>                              | 4-4  |
| <u>Section 10.5 Functional Arguments</u>                            | 4-4  |
| <u>Section 10.6.2 Interpreting Macros</u>                           | 4-4  |
| <u>Chapter 11 Variable Bindings and the Interlisp Stack</u>         | 4-4  |
| <u>Section 11.2.1 Searching the Stack</u>                           | 4-5  |
| <u>Section 11.2.2 Variable Bindings in Stack Frames</u>             | 4-5  |
| <u>Section 11.2.5 Releasing and Reusing Stack Pointers</u>          | 4-5  |
| <u>Section 11.2.7 Other Stack Functions</u>                         | 4-5  |
| <u>Chapter 12 Miscellaneous</u>                                     | 4-6  |
| <u>Section 12.2 Idle Mode</u>                                       | 4-6  |
| <u>Section 12.3 Saving Virtual Memory State</u>                     | 4-7  |
| <u>Section 12.4 System Version Information</u>                      | 4-7  |
| <u>Chapter 13 Interlisp Executive</u>                               | 4-8  |

## TABLE OF CONTENTS

---

|                                                             |      |
|-------------------------------------------------------------|------|
| Chapter 14 Errors and Breaks                                | 4-8  |
| Section 14.5 Break Window Variables                         | 4-8  |
| Section 14.8 Catching Errors                                | 4-8  |
| Chapter 17 File Package                                     | 4-9  |
| Section 17.8.1 Functions for Manipulating Typed Definitions | 4-9  |
| Section 17.8.2 Defining New File Package Types              | 4-9  |
| Section 17.9.2 Variables                                    | 4-9  |
| Section 17.9.8 Defining New File Package Commands           | 4-9  |
| Section 17.11 Symbolic File Format                          | 4-9  |
| Section 17.11.3 File Maps                                   | 4-10 |
| Chapter 18 Compiler                                         | 4-10 |
| Chapter 21 CLISP                                            | 4-10 |
| Section 21.8 Miscellaneous Functions and Variables          | 4-10 |
| Chapter 22 Performance Issues                               | 4-11 |
| Section 22.1 Storage Allocation and Garbage Collection      | 4-11 |
| Section 22.5 Using Data Types Instead of Records            | 4-11 |
| Chapter 23 Processes                                        | 4-12 |
| Section 23.6 Typein and the TTY Process                     | 4-12 |
| Section 23.8 Process Status Window                          | 4-12 |
| Chapter 24 Streams and Files                                | 4-13 |
| Section 24.7 File Attributes                                | 4-13 |
| Section 24.9 Local Hard Disk Device                         | 4-13 |
| Section 24.10 Floppy Disk Device                            | 4-13 |
| Section 24.12 Temporary Files and CORE Device               | 4-13 |
| Section 24.18.1 Pup File Server Protocols                   | 4-14 |
| Section 24.18.1-2 Use of BREAKCONNECTION with File Servers  | 4-14 |
| Section 24.18.2 NS File Server Protocols                    | 4-15 |
| Section 24.18.3 Operating System Designations               | 4-15 |
| Chapter 25 Input/Output Functions                           | 4-15 |
| Section 25.2 Input Functions                                | 4-15 |
| Section 25.3.2 Printing Numbers                             | 4-15 |
| Section 25.3.4 Printing Unusual Data Structures             | 4-15 |
| Section 25.4 Random Access File Operations                  | 4-16 |
| Section 25.6 PRINTOUT                                       | 4-16 |
| Section 25.8.3 READ Macros                                  | 4-16 |
| Chapter 26 User Input/Output Packages                       | 4-16 |
| Section 26.3 ASKUSER                                        | 4-16 |
| Section 26.4 TTYIN Display Typein Editor                    | 4-16 |
| Section 26.4.3 Display Editing Commands                     | 4-17 |
| Section 26.4.5 Useful Macros                                | 4-18 |

|                                                            |      |
|------------------------------------------------------------|------|
| Chapter 27 Graphic Output Operations                       | 4-18 |
| Section 27.1.3 Bitmaps                                     | 4-18 |
| Section 27.3 Accessing Image Stream Fields                 | 4-18 |
| Section 27.6 Drawing Lines                                 | 4-19 |
| Section 27.7 Drawing Curves                                | 4-19 |
| Section 27.8 Miscellaneous Drawing and Printing Operations | 4-19 |
| Section 27.12 Fonts                                        | 4-21 |
| Section 27.13 Font Files and Font Directories              | 4-23 |
| Section 27.14 Font Classes                                 | 4-23 |
| Section 27.14 Font Profiles                                | 4-23 |
| Chapter 28 Windows and Menus                               | 4-24 |
| Section 28.4 Windows                                       | 4-24 |
| Section 28.4.5 Reshaping Windows                           | 4-24 |
| Section 28.4.8 Shrinking Windows Into Icons                | 4-24 |
| Section 28.4.11 Terminal I/O and Page Holding              | 4-25 |
| Section 28.5 Menus                                         | 4-26 |
| Section 28.6.2 Attached Prompt Windows                     | 4-28 |
| Section 28.6.3 Window Operations and Attached Windows      | 4-28 |
| Chapter 29 Hardcopy Facilities                             | 4-29 |
| Chapter 30 Terminal Input/Output                           | 4-29 |
| Section 30.1 Interrupt Characters                          | 4-29 |
| Section 30.2.3 Line Buffering                              | 4-30 |
| Section 30.4.1 Changing the Cursor Image                   | 4-30 |
| Section 30.5 Keyboard Interpretation                       | 4-31 |
| Section 30.6 Display Screen                                | 4-31 |
| Section 30.7 Miscellaneous Terminal I/O                    | 4-31 |
| Chapter 31 Ethernet                                        | 4-32 |
| Section 31.3.1 Name and Address Conventions                | 4-32 |
| Section 31.3.2 Clearinghouse Functions                     | 4-33 |
| Section 31.3.3 NS Printing                                 | 4-34 |
| Section 31.3.5.3 Performing Courier Transactions           | 4-34 |
| Section 31.3.5.3.3 Using Bulk Data Transfer                | 4-34 |
| Section 31.5 Pup Level One Functions                       | 4-34 |
| Section 31.6.1 Creating and Managing XIPs                  | 4-35 |
| <b>5. Library Modules</b>                                  | 5-1  |
| Modules That are New, Moved, or Replaced                   | 5-1  |
| Modules Moved From the Library to LispUsers                | 5-1  |
| Modules Moved From LispUsers to the Library                | 5-1  |
| Modules Moved to Their Own Manuals                         | 5-1  |

## TABLE OF CONTENTS

---

|                                                |     |
|------------------------------------------------|-----|
| Modules Moved From the Sysout into the Library | 5-1 |
| Modules Moved From the Library into the Sysout | 5-2 |
| Modules Replaced                               | 5-2 |
| New Modules                                    | 5-2 |
| Details of Change                              | 5-2 |
| 4045XLPStream                                  | 5-2 |
| Cash-File                                      | 5-2 |
| Centronics                                     | 5-3 |
| Chat                                           | 5-3 |
| CopyFiles                                      | 5-3 |
| DataBaseFns                                    | 5-3 |
| EditBitMap                                     | 5-3 |
| FileBrowser                                    | 5-3 |
| FTPServer                                      | 5-4 |
| FX-80Driver                                    | 5-4 |
| GCHax                                          | 5-5 |
| Grapher                                        | 5-5 |
| Hash                                           | 5-5 |
| Hash-File                                      | 5-5 |
| Kermit                                         | 5-5 |
| MasterScope                                    | 5-5 |
| NSMaintain                                     | 5-5 |
| RS232                                          | 5-6 |
| Spy                                            | 5-6 |
| TableBrowser                                   | 5-6 |
| TCP- IP                                        | 5-7 |
| TExec                                          | 5-8 |
| TextModules                                    | 5-8 |
| Virtual Keyboards                              | 5-8 |
| Where-Is                                       | 5-8 |
| Additional Notes                               | 5-8 |
| Koto CML Library Module                        | 5-8 |

## 6. User's Guides

---

6-1

|                                        |     |
|----------------------------------------|-----|
| A User's Guide to TEdit—Release Notes  | 6-1 |
| Expanded Characters                    | 6-1 |
| Put Submenu                            | 6-1 |
| Get Submenu                            | 6-2 |
| Clarified Paragraph Looks Menu Options | 6-2 |
| New Page: Before After                 | 6-3 |

|                                                                 |      |
|-----------------------------------------------------------------|------|
| <u>Displaymode: Hardcopy</u>                                    | 6-3  |
| <u>Clarified Page Layout Menu Options</u>                       | 6-3  |
| <u>Added Items to Programmer's Interface</u>                    | 6-3  |
| <u>Corrected the AFTERQUITFN Property</u>                       | 6-3  |
| <u>Corrected the TEXTOBJ Data Structure</u>                     | 6-4  |
| <u>Corrected the TITLEMENUFN Property</u>                       | 6-4  |
| <u>Expanded the TEDIT.INCLUDE Function</u>                      | 6-4  |
| <u>Expanded the TEDIT.PARALOOKS Function</u>                    | 6-4  |
| <u>Expanded the TEXTPROP Function</u>                           | 6-5  |
| <u>Added Documentation for Global Variables</u>                 | 6-5  |
| <u>Changes to Programmer's Interface to TEdit</u>               | 6-5  |
| <u>STREAM and TEXTOBJ</u>                                       | 6-5  |
| <u>Changes, Additions and Corrections to TEdit Functions</u>    | 6-5  |
| <u>Changes in Documentation of TEdit Functions</u>              | 6-7  |
| <u>New Features</u>                                             | 6-8  |
| <u>A User's Guide to Sketch—Release Notes</u>                   | 6-10 |
| <u>Manipulating Sketch Elements</u>                             | 6-10 |
| <u>Adding and Deleting Control Points</u>                       | 6-10 |
| <u>Deleting Control Points</u>                                  | 6-10 |
| <u>Defaults Command</u>                                         | 6-10 |
| <u>Better Feedback for Creating Wires, Circles and Ellipses</u> | 6-10 |
| <u>Arrowheads</u>                                               | 6-10 |
| <u>Deleting Characters During Type-in</u>                       | 6-10 |
| <u>Using Bit Maps in a Sketch</u>                               | 6-11 |
| <u>Zooming Bitmaps</u>                                          | 6-11 |
| <u>Changing Bitmaps</u>                                         | 6-11 |
| <u>Freezing Sketch Elements</u>                                 | 6-11 |
| <u>Aligning Sketch Elements</u>                                 | 6-11 |
| <u>Placing Multiple Copies of Elements</u>                      | 6-11 |
| <u>Making the Window Fit the Sketch</u>                         | 6-12 |
| <u>Overlaying Figure Elements</u>                               | 6-12 |
| <u>Changing How Elements Overlap</u>                            | 6-12 |
| <u>Loading the Sketch Library Module</u>                        | 6-12 |
| <u>The Programmer's Interface</u>                               | 6-13 |
| <u>New Behavior for the Get Command</u>                         | 6-13 |
| <u>Establishing Initial Defaults for Sketch</u>                 | 6-13 |
| <u>1108 User's Guide Release Notes</u>                          | 6-14 |
| <u>What to Look For</u>                                         | 6-14 |
| <u>File System</u>                                              | 6-14 |
| <u>System Tools</u>                                             | 6-14 |

## TABLE OF CONTENTS

---

|                                 |      |
|---------------------------------|------|
| Input/Output                    | 6-15 |
| Machine Diagnostics             | 6-15 |
| 1186 User's Guide Release Notes | 6-16 |
| What to Look For                | 6-16 |
| File System                     | 6-16 |
| Software Installation           | 6-16 |
| System Tools                    | 6-17 |
| Input/Output                    | 6-17 |
| Diagnostics                     | 6-17 |

## **7. Common Lisp Implementation** 7-1

|                                  |     |
|----------------------------------|-----|
| New Features Since Lyric         | 7-1 |
| Common Lisp Definers             | 7-1 |
| Compile-Definer                  | 7-2 |
| Compile-Form                     | 7-2 |
| Define-File-Environment          | 7-2 |
| Site-Name Special Uses           | 7-3 |
| Record Access                    | 7-3 |
| Define-Record                    | 7-3 |
| Record-Fetch                     | 7-4 |
| Record-FFetch                    | 7-4 |
| Record-Create                    | 7-4 |
| Array Reference                  | 7-4 |
| Shadowing of Global Macros       | 7-4 |
| Evaluating Load-time Expressions | 7-4 |
| Common Lisp Defstruct Options    | 7-4 |
| Defstruct Options                | 7-5 |
| Defstruct Slot Options           | 7-5 |
| Warning When Using Defstruct     | 7-6 |
| Macros for Collecting Objects    | 7-6 |
| xcl:with-collection              | 7-6 |
| Macros for Writing Macros        | 7-7 |
| xcl:once-only                    | 7-7 |
| Common Lisp Append Datatypes     | 7-8 |
| Closure Cache                    | 7-8 |
| Symbols and Packages             | 7-8 |
| Pkg -goto and In -package        | 7-8 |
| Defpackage Export Argument       | 7-9 |
| Debugging Tools                  | 7-9 |
| Breaking                         | 7-9 |

|                                                                |      |
|----------------------------------------------------------------|------|
| Advising                                                       | 7-9  |
| Argument Names Displayed for Interpreted Functions             | 7-10 |
| Lexical Variables Evaluated by Debugger                        | 7-10 |
| Pathname Component Fixed in FS-ERROR                           | 7-10 |
| Compiler Optimizations                                         | 7-10 |
| Warning when using LABELS Construct                            | 7-10 |
| COMS added to dfasl files                                      | 7-11 |
| Loadflg argument                                               | 7-11 |
| Changes in MAP, WRITE-STRING, COERCE, GENSYM, DEFERREDCONSTANT | 7-11 |
| Compiler keeps Special &REST arguments                         | 7-12 |
| Compiler ignores TEdit formatting                              | 7-12 |
| Compiler notices Tail-recursive Lexical Functions              | 7-12 |
| Compiler Error Message                                         | 7-12 |
| Format ~C and WRITE-CHAR                                       | 7-13 |
| WITH-OUTPUT-TO-STRING / WITH-INPUT-FROM-STRING                 | 7-13 |

|                                       |            |
|---------------------------------------|------------|
| <b>A. The Exec</b>                    | <b>A-1</b> |
| Input Formats                         | A-2        |
| Multiple Execs and the Exec's Type    | A-4        |
| Event Specification                   | A-4        |
| Exec Commands                         | A-5        |
| Variables                             | A-9        |
| Fonts in the Exec                     | A-10       |
| Changing the Exec                     | A-11       |
| Defining New Commands                 | A-11       |
| Undoing                               | A-12       |
| Undoing in the Exec                   | A-13       |
| Undoing in Programs                   | A-13       |
| Undoable Versions of Common Functions | A-14       |
| Modifying the UNDO Facility           | A-14       |
| Undoing Out of Order                  | A-16       |
| Format and Use of the History List    | A-16       |
| Making or Changing an Exec            | A-18       |
| Editing Exec Input                    | A-20       |
| Editing Your Input                    | A-21       |
| Using the Mouse                       | A-21       |
| Editing Commands                      | A-22       |



## TABLE OF CONTENTS

---

|                                      |            |
|--------------------------------------|------------|
| Cursor Movement Commands             | A-22       |
| Buffer Modification Commands         | A-23       |
| Miscellaneous Commands               | A-23       |
| Useful Macros                        | A-24       |
| ?= Handler                           | A-24       |
| Assorted Flags                       | A-24       |
| <b>B. SEdit—The Lisp Editor</b>      | <b>B-1</b> |
| 16.1 SEDIT—The Structure Editor      | B-1        |
| 16.1.1 An Edit Session               | B-1        |
| 16.1.2 SEdit Carets                  | B-2        |
| 16.1.3 The Mouse                     | B-3        |
| 16.1.4 Gaps                          | B-4        |
| 16.1.5 Broken Atoms                  | B-4        |
| 16.1.6 Special Characters            | B-5        |
| 16.1.7 Commands                      | B-6        |
| 16.1.8 Editing Commands              | B-7        |
| 16.1.9 Completion Commands           | B-7        |
| 16.1.10 Undo Commands                | B-7        |
| 16.1.11 Find Commands                | B-8        |
| 16.1.12 General Commands             | B-9        |
| 16.1.13 Miscellaneous                | B-11       |
| 16.1.14 Help Menu                    | B-11       |
| 16.1.15 Command Menu                 | B-12       |
| 16.1.16 SEdit Programmer's Interface | B-12       |
| 16.1.17 SEdit Window Region Manager  | B-12       |
| 16.1.18 Options                      | B-13       |
| 16.1.19 Control Functions            | B-14       |
| Warning with Declarations            | B-18       |
| <b>C. ICONW</b>                      | <b>C-1</b> |
| 28.4.16 Creating Icons with ICONW    | C-1        |
| 28.4.16.1 Creating Icons             | C-1        |
| 28.4.16.2 Modifying Icons            | C-2        |
| 28.4.16.3 Default Icons              | C-3        |
| 28.4.16.4 Sample Icons               | C-3        |
| <b>D. Free Menu</b>                  | <b>D-1</b> |
| 28.7 Free Menus                      | D-1        |

|                                           |                |
|-------------------------------------------|----------------|
| 28.7.1 Making a Free Menu                 | D-1            |
| 28.7.2 Free Menu Formatting               | D-1            |
| 28.7.3 Free Menu Descriptions             | D-2            |
| 28.7.4 Free Menu Group Properties         | D-7            |
| 28.7.5 Other Group Properties             | D-8            |
| 28.7.6 Free Menu Items                    | D-8            |
| 28.7.7 Free Menu Item Description         | D-8            |
| 28.7.8 Free Menu Item Properties          | D-9            |
| 28.7.9 Mouse Properties                   | D-10           |
| 28.7.10 System Properties                 | D-10           |
| 28.7.11 Predefined Item Types             | D-11           |
| 28.7.12 Free Menu Item Highlighting       | D-14           |
| 28.7.13 Free Menu Item Links              | D-14           |
| 28.7.14 Free Menu Window Properties       | D-15           |
| 28.7.15 Free Menu Interface Functions     | D-15           |
| 28.7.16 Accessing Functions               | D-15           |
| 28.7.17 Changing Free Menus               | D-16           |
| 28.7.18 Editor Functions                  | D-17           |
| 28.7.19 Miscellaneous Functions           | D-18           |
| 28.7.20 Free Menu Macros                  | D-18           |
| <b>E. Error System</b>                    | <b>E-1</b>     |
| Summary of Error System Changes           | E-1            |
| Introduction to Error System Terminology  | E-3            |
| Program Interface to the Condition System | E-5            |
| Defining and Creating Conditions          | E-5            |
| Signalling Conditions                     | E-8            |
| Handling Conditions                       | E-11           |
| Restarts                                  | E-13           |
| <b>INDEX</b>                              | <b>INDEX-1</b> |

[This page intentionally left blank]



*Venue*

*Medley for the Sun Workstation<sup>®</sup> User's Guide*

*Release 2.0  
March, 1991*



*Venue*

*Medley for the Sun Workstation® User's Guide*

Address comments to:  
Venue  
User Documentation  
1549 Industrial Road  
San Carlos, CA 94070  
415-508-9672

---

## MEDLEY FOR THE SUN WORKSTATION® USER'S GUIDE

Release 2.0

June, 1991

Copyright © 1990, 1991 by Venue.

All rights reserved.

Envos is a trademark of Envos Corporation.

Medley is a trademark of Venue.

Xerox® is a registered trademark and InterPress is a trademark of Xerox Corporation.

UNIX® is a registered trademark of UNIX System Laboratories.

PostScript is a registered trademark of Adobe Systems Inc.

The following are trademarks of Sun Microsystems, Inc.:

Sun, Sun-2, Sun-3, Sun-4, SunOS, and SPARCstation are trademarks of Sun Microsystems, Inc.

Sun® and Sun Workstation® are registered trademarks of Sun Microsystems, Inc.

The X Window System is a trademark of the Massachusetts Institute of Technology.

IBM is a trademark of International Business Machines, Inc.

MIPS is a registered trademark of MIPS Computer Systems, Inc.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

---

The information in this document is subject to change without notice and should not be construed as a commitment by Venue. While every effort has been made to ensure the accuracy of this document, Venue assumes no responsibility for any errors that may appear.

Text was written and produced with Venue text formatting tools; Xerox printers were used to produce text masters. The typeface is Classic.

# 1. INTRODUCTION

---

Medley is an integrated programming environment, with support for the Interlisp and Common Lisp languages, an integrated windowing system, and a large collection of utilities and programs. It offers a mature and rich programming and development environment, as well as access to a large number of applications written for Interlisp, Interlisp-D, Common Lisp, and LOOPS.

Medley for the Sun Workstation has two versions, a Sun-3 version and a Sun-4 version, available on separate tapes. Medley 2.0 runs on the Sun-3 and Sun-4 workstations and the SPARCstation.

## What Medley Requires

---

### Hardware

Medley runs on Sun-3 and Sun-4 Workstations and the SPARCstation. It runs on both standalone workstations and diskless workstations linked to servers.

Medley on the Sun-3 Workstation requires the MC68881 floating-point coprocessor chip. On the Sun-4 Workstation, the Weitek 1164/1165 coprocessor is optional, but recommended.

For adequate performance, we recommend at least a 20 MHz 68020 (Sun 3/60 or 3/260), a 14 MHz SPARC (Sun 4/110 or 4/260), or a SPARCstation.

Except under X Windows, reasonable interactive performance can be expected with 8 megabytes (MB) or more of RAM. Smaller configurations of diskless workstations have been tested, but performance suffers. When using X Windows software, allow an additional 4 MB.

Naturally, larger applications will benefit from more memory. Medley's maximum working set is approximately 40 MB.

### Input/Output Devices

Medley provides access to the Sun's input/output devices, such as display, keyboard, mouse, and file systems. It also provides access to PUP and XNS Ethernet services directly.

### Bitmap Display

Medley supports all standard Sun displays and frame buffers.

### Printers

You can print on Xerox Interpress printers using the XNS networking protocols. The FX80 printer also works via the RS232 port.

If you have a PostScript printer, you can use the LispUser modules PostScriptStream and UNIXPrint to direct output to your printer.

### Software Requirements

Medley on the Sun-3 Workstation requires SunOS versions 3.2, 3.4, 3.5, 4.0, or 4.0.3. On the Sun-4 Workstation, Medley requires SunOS version 4.0, 4.0.3, or 4.1.

If you plan to run Medley under X Windows, you will need X11, version 4, or Motif.

NOTE: Medley's XNS Ethernet code will not work if you are running SunOS 3.5 configured for Kernel XNS Ethernet Support or Alpine.

## Medley and Other Applications

---

### Display Usage

When Medley is running alone, it takes over the entire display screen. When running under X, Medley uses one window as its screen; Medley maintains its own windows within that single window. Medley cannot run at the same time as Suntools or Open Windows.

### Processor Usage

Medley runs its own process scheduler; as far as the UNIX scheduler is concerned, Medley is always running. For this reason, other heavy computational jobs on the same Sun Workstation will not get as good performance as they would competing with conventional UNIX interactive applications.

Similarly, Medley may not have adequate interactive performance if it is competing with other compute-bound tasks on the same machine.

For these reasons, we recommend that Medley be used on machines that are set up primarily for a single user.

## System Components

---

Functionally, Medley consists of the following components:

|          |                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| emulator | A SunOS-executable program, which performs several functions. It executes the Interlisp-D virtual machine instruction set compatibly with the microcode of the Xerox 1100 series workstations. (This instruction set allows memory-efficient representation of Interlisp and Common Lisp programs.) It also provides access to the host machine's I/O (display, keyboard, file system), and executes some system functions directly. |
| sysout   | A virtual memory image (the <i>sysout</i> ) containing both byte-code-compiled Lisp functions and data structures. The sysout provided can be used both on the Sun Workstation and on the Xerox 1100 series machines.                                                                                                                                                                                                                |
| library  | Files of compiled Lisp code and data structures.                                                                                                                                                                                                                                                                                                                                                                                     |
| fonts    | Data describing the "looks" of printed characters used by Medley's graphics, windowing, and hardcopying subsystems. Font directories are in three groups: display fonts, InterPress printer fonts, and Press printer fonts.                                                                                                                                                                                                          |



**checksum** A script that reports inconsistent files, the correct checksum values for the files, and an error message. The checksum of individual files can be generated with the UNIX command `sum filename`. Use this when Medley installs correctly but does not run.

---

## Medley Device-Naming Conventions

Medley for the Sun Workstation lets you interact with SunOS file systems (including file systems mounted from other machines) by using host device names. The two device names are as follows:

- `{DSK}` A host name which gives you access to the SunOS file system using Xerox workstation local disk conventions.
- `{UNIX}` A host name which gives you access to the file system using normal SunOS conventions.

The `{DSK}` device name provides an interface to the Sun Workstation for users who want to maintain compatibility with existing development tools and applications originally developed on a Xerox workstation. The `{UNIX}` device name provides a way for new applications to interact naturally with UNIX. Chapter 5 explains, in greater detail, some important exceptions and restrictions to the `{DSK}` and `{UNIX}` device name.

---

## Notation Conventions

Text marked by a revision bar in the right margin contains information that was added or modified since the last release. Fonts, packages, and prompts have the following types of notation.

### Fonts

**Bold text in TITAN font** indicates text you should type in exactly as printed.

Regular TITAN font text indicates what the system prints on your workstation screen. Lisp functions and variables and UNIX files and programs are also shown in TITAN FONT.

*Text in Classic italics* indicates variables or parameters that you should replace with the appropriate word or string.

### Packages

Most Lisp symbols have a Lisp package qualifier; the INTERLISP package (`IL:`) is the default when no package qualifier is shown.

### Prompts

All examples which include SunOS dialogues use the following conventions for the SunOS prompt.

A number sign (`#`), part of the system prompt, indicates that you are logged on as `root` or is running `su`; for example,

prompt#

A percent sign (%), part of the system prompt, indicates that a user other than root is logged on; for example,

prompt%

## Compatibility

---

The Medley release on the Sun Workstation is designed for maximum compatibility with the Xerox workstation implementations. However, when moving applications to the Sun Workstation note the differences in end-of-line conventions and techniques for moving files.

### Sysout Compatibility

Sysouts of the same version are compatible with all machine types. But a sysout generated on a Sun Workstation cannot be used on a Xerox workstation.

NOTE: You cannot mix different versions of sysouts and emulators.

### Compiled-File Compatibility

Code compiled in a Medley 1.0, 1.1, 1.15 or 1.2 sysout cannot be loaded into Medley 2.0 sysouts, nor can code compiled in Medley 2.0 be loaded onto earlier sysouts. Code compiled for Medley 2.0 on a Xerox workstation cannot be loaded into Medley running on a Sun. The opposite is not possible either.

### End-of-Line Convention

Some care must be taken in moving files to and from Xerox workstations, since the default end-of-line convention in UNIX is to terminate lines with the line feed (LF) character, while, traditionally, Medley has terminated lines with the carriage return (CR) character. In particular, if you use some other file transfer mechanism, such as FTP or Kermit, be careful to transfer .TEDIT, .DFASL, and .LCOM files in binary mode.

In Medley on the Sun Workstation, the default end-of-line convention for all text files is line feed (LF). The default end-of-line convention for all binary files is carriage return (CR); this is because CR (ASCII 13) is used internally in the system.

## Release Contents

---

The release distribution contains the following documentation and software.

### Documentation

The Medley documentation kit for users moving from a Xerox workstation to a Sun Workstation contains:

- *Lisp Library Modules*, Medley Release
- *Lisp Release Notes*, Medley Release
- *Medley For the Sun Workstation® User's Guide*

- Sun Type 3 and Type 4 keyboard templates.

New customers also receive the following:

- *Interlisp-D Reference Manual, Volumes 1-3*, Koto Release
- *Xerox Common Lisp Implementation Notes*, Lyric Release
- *Lisp Documentation Tools*, Lyric Release
- Guy Steele, *Common Lisp, the Language, First Edition*

All users can also purchase this document:

- *LispUsers' Modules*, Medley Release

## Software

The software release is available on either a ¼-inch tape cartridge or a ½-inch 9-track tape. The software release is specific to the Sun architecture (Sun 3 or 4) for which you purchased Medley, but contains multiple SunOS versions. This tar tape contains the directories listed below. (See Appendix C for details of the directory contents.)

```
./install-medley
./medley
./install.sunos3/
./install.sunos4/
./install.sunos4.1/
./lisplibrary
./checksumdir
./lispsysouts
./fonts/display
./fonts/interpress
```

## LispUsers Modules

The Medley version of LispUsers Modules is a software supplement to Medley for the Sun Workstation. This is software written by our users which you may purchase separately. The support for these modules comes from each module's author; Venue has no commitment to support LispUsers' modules.

Two LispUsers Modules are particularly useful when you are running Medley on a Sun Workstation. For those users with Postscriptstream printers for output, the PostScript module is particularly useful. The LispUsers module RPC implements Sun remote procedure calls.

[This page intentionally left blank]

# TABLE of CONTENTS

---

|                                                 |           |
|-------------------------------------------------|-----------|
| <b>Preface .....</b>                            | <b>ix</b> |
| <b>1. Introduction.....</b>                     | <b>1</b>  |
| What Medley Requires .....                      | 1         |
| Hardware .....                                  | 1         |
| Input/Output Devices .....                      | 1         |
| Bitmap Display .....                            | 1         |
| Printers.....                                   | 1         |
| Software Requirements .....                     | 2         |
| Medley and Other Applications .....             | 2         |
| Display Usage .....                             | 2         |
| Processor Usage .....                           | 2         |
| System Components .....                         | 2         |
| Medley Device-Naming Conventions .....          | 3         |
| Notation Conventions .....                      | 3         |
| Fonts.....                                      | 3         |
| Packages.....                                   | 3         |
| Prompts .....                                   | 3         |
| Compatibility .....                             | 4         |
| Sysout Compatibility .....                      | 4         |
| Compiled-File Compatibility.....                | 4         |
| End-of-Line Convention .....                    | 4         |
| Release Contents .....                          | 4         |
| Documentation.....                              | 4         |
| Software .....                                  | 5         |
| LispUsers Modules .....                         | 5         |
| <b>2. Software Installation .....</b>           | <b>7</b>  |
| Getting Ready to Install Medley .....           | 7         |
| Ensuring Adequate Swap Space .....              | 8         |
| Installing Medley for Shared Use .....          | 8         |
| Installing Medley Software.....                 | 9         |
| Using the Installation Script's Menu .....      | 9         |
| Getting a Copy Protection Key .....             | 11        |
| Changing Configurations or Adding Options ..... | 11        |



|                                                            |               |
|------------------------------------------------------------|---------------|
| <b>3. Getting Started .....</b>                            | <b>13</b>     |
| Getting Ready.....                                         | 13            |
| Running Medley .....                                       | 13            |
| Running Medley Directly .....                              | 13            |
| Using the Medley Shell Script .....                        | 13            |
| Where Medley Looks for Your Sysout .....                   | 15            |
| Where Medley Looks for Your Site Initialization File ..... | 15            |
| Medley and X Windows.....                                  | 16            |
| Starting X Windows.....                                    | 16            |
| Running Medley Remotely.....                               | 16            |
| The Medley Window .....                                    | 17            |
| Environment Variables .....                                | 17            |
| Keyboard Interpretation .....                              | 17            |
| Sun Type 3 Keyboard .....                                  | 18            |
| Sun Type 4 Keyboard .....                                  | 19            |
| <br><b>4. Using Medley on the Sun Workstation .....</b>    | <br><b>21</b> |
| Setting Up a Site Init File .....                          | 21            |
| Setting Up a Personal Init File .....                      | 22            |
| Saving Your State .....                                    | 22            |
| Sun-Specific Environment Functions.....                    | 24            |
| System Environment Functions and Variables .....           | 24            |
| VM Functions.....                                          | 25            |
| Stopping Lisp Temporarily .....                            | 25            |
| Login Functions .....                                      | 26            |
| Environment Inquiry.....                                   | 26            |
| Display and Keyboard Functions and Variables .....         | 27            |
| Timers and Clocks .....                                    | 27            |
| Miscellaneous Operational Differences .....                | 28            |
| Console Messages .....                                     | 28            |

|                                                 |               |
|-------------------------------------------------|---------------|
| <b>5. Medley File Systems .....</b>             | <b>31</b>     |
| File Naming Conventions .....                   | 31            |
| Hosts that Medley Supports .....                | 31            |
| Using SunOS Files from Medley .....             | 31            |
| Common {DSK} and {UNIX} Naming Conventions..... | 32            |
| {DSK} Naming Conventions.....                   | 33            |
| Version Numbering.....                          | 33            |
| Pathnames.....                                  | 35            |
| {UNIX} Naming Conventions.....                  | 35            |
| Directories.....                                | 36            |
| Directory Enumeration.....                      | 36            |
| Directory Creation .....                        | 36            |
| Directory Deletion.....                         | 37            |
| Open File Limit.....                            | 37            |
| Default Pathname .....                          | 37            |
| File Attributes .....                           | 38            |
| File Variables .....                            | 39            |
| File System Errors .....                        | 39            |
| <br><b>6. Error Recovery .....</b>              | <br><b>41</b> |
| URAIID .....                                    | 41            |
| Entering URAID .....                            | 41            |
| Conventions.....                                | 41            |
| URAIID Commands .....                           | 42            |
| Displaying a Stack .....                        | 42            |
| Viewing Frames from a Stack .....               | 43            |
| Miscellaneous .....                             | 44            |
| Other Fatal Error Conditions .....              | 44            |
| Lisp Errors.....                                | 44            |
| Errors While Running Medley .....               | 44            |
| Xerox Workstation-Specific Errors .....         | 45            |
| Virtual Memory Errors.....                      | 45            |



|                                                                                                                      |            |
|----------------------------------------------------------------------------------------------------------------------|------------|
| <b>Appendix A. Installation Hints</b> .....                                                                          | A-1        |
| Medley Shell Variables .....                                                                                         | A-1        |
| Running on Multiple Workstations .....                                                                               | A-1        |
| Installation for Sites with Sun-3 and Sun-4 Workstations .....                                                       | A-1        |
| Using a "runlde" on Multiple Workstations .....                                                                      | A-2        |
| Configuring the Software.....                                                                                        | A-2        |
| Relinking .....                                                                                                      | A-3        |
| Enabling PUP/XNS Ethernet.....                                                                                       | A-3        |
| Using NIS to Manage the Keys for Multiple Workstations .....                                                         | A-3        |
| <br>                                                                                                                 |            |
| <b>Appendix B. Verifying the Installation Tape's Validity</b> .....                                                  | B-1        |
| Output.....                                                                                                          | B-1        |
| Examples .....                                                                                                       | B-1        |
| <br>                                                                                                                 |            |
| <b>Appendix C. Layout of Installation Tape Files</b> .....                                                           | C-1        |
| Layout of Installation Tape .....                                                                                    | C-1        |
| Font Directories.....                                                                                                | C-3        |
| Manually Extracting Files from the Installation Tape .....                                                           | C-4        |
| <br>                                                                                                                 |            |
| <b>Appendix D. Differences between Xerox Workstations and<br/>                  the UNIX Version of Medley</b> ..... | D-1        |
| Local Disk and Floppy Functions .....                                                                                | D-1        |
| Library Modules Not Supported on the Sun.....                                                                        | D-1        |
| TCP, TCPCHAT, etc. ....                                                                                              | D-1        |
| DLRS232C, DLTTY .....                                                                                                | D-2        |
| KEYBOARD, VIRTUALKEYBOARD.....                                                                                       | D-2        |
| <br>                                                                                                                 |            |
| <b>Glossary</b> .....                                                                                                | GLOSSARY-1 |
| <b>Index</b> .....                                                                                                   | INDEX-1    |

[This page intentionally left blank]

## 2. SOFTWARE INSTALLATION

---

This chapter describes how to install Medley on Sun Workstations.

To install the Medley Release on a Sun Workstation, you need the following:

- Release tape
- Medley documentation kit for the Sun Workstation.

### Getting Ready to Install Medley

---

Medley includes a shell script for automatic installation. The script infers as much as it can about your host and network, but will prompt you for answers when needed. Once it has collected the necessary information, it installs only those parts of Medley that you really need.

Do not worry if you forget something. You can run the installation again, and pick up any items you missed.

Before installing Medley, you should gather some facts about the hardware and network environment on which you will be using Medley. The following checklist will help you.

- Do you have the correct release tape correct for the kind of machine on which you plan to run?

The tape is labeled either Sun-3 or Sun-4/SPARCstation.

- Where is the tape drive you will be using?

Does your Sun have a ¼-inch tape drive? If not, you need a Sun with a ¼-inch tape drive on your network. You will need to know the host name for that machine.

- Does your system have sufficient swap space (45 MB) for Medley?

If you are not sure, see the section below for instructions.

- Are you installing Medley for a single user, or will several users be sharing this copy?

If it is for shared use, you will want to install Medley on a public directory on a shared server. For that, we recommend using `/usr/share/lde` as the directory name. You will need to be running on the server when you do the installation, and you will probably need to log in as `root`. Check with your System Administrator for details.

- Do you have enough disk space free ?

You need to select a file system with enough disk space to install the software. A minimal installation requires approximately 12 MB, and a full installation will require up to 23 MB. Use the UNIX command `df` to find one.

- Have you selected an installation directory?

The directory must be on a file system with enough space. For individual use, we recommend `/usr/local/lde`.

---

### CAUTION

If the installation directory contains a previously installed version of Medley, some of the older files will be replaced with new ones.

---

- Do you have write permission to create the installation directory and to write files into it?
- Will you be running on X Windows?  
If you are, you will need X11 R4 or Motif. If you have X Windows installed, the utility will install the software necessary to run Medley under X11.
- Will you be running XNS (Xerox Network Services) or PUP protocols?
  - If so, you must be logged in as `root` during the installation.
  - If so, be sure you are not running SunOS 3.5 Kernel XNS Ethernet Code ("Alpine Kernel").
- If your host is networked, do you have XNS (Xerox Network Services) servers on it?  
If so, you will want to install XNS-relevant software. Also, if you have XNS Print Services and InterPress printers on your network, you might want to install InterPress fonts, allowing you to use an InterPress printer from within Medley. The default is not to install any XNS-relevant software.
- Is this a new installation, or are you upgrading from an earlier version of Medley?  
If you are upgrading, you only need to install the `sysout`, the appropriate emulator, and library files. If you are making a new installation, you will need at least the display fonts as well.

### Ensuring Adequate Swap Space

Medley requires 45 MB of swap space on top of the normal swap space requirements. Check swap space using the `pstat` command:

```
prompt% /etc/pstat -s
37176k used (3176k text), 12920k free, 1344k wasted, 0k missing
max process allocable = 10224k
avail: 5*2048k 1*512k 4*256k 3*128k 6*64k 7*32k 7*16k 40*1k
4800k allocated + 2520k reserved = 7320k used, 64672k available
```

If you need more swap space, consult the *Sun Software Technical Bulletin, March 1988*, pages 335-36, for information on increasing the amount of available swap space.

### Installing Medley for Shared Use

If several people will be using Medley on different machines, it probably makes sense to install one copy and have people share it.

You will need to find one machine—probably your main file server—with enough disk space. You will also want to have the directory look the same to every user. We recommend calling the installation directory `/usr/share/lde`.

## Installing Medley Software

1. Log in under your username.

```
login yourname
```

2. Put the tape in the tape drive. The script will allow you to install from a tape drive on a remote host. If you are performing a remote installation, put the tape in the tape drive of the host.

3. Retrieve the installation utility from the tape, as follows.

- If the tape drive is on a different host, enter the following:

```
rsh remote-host dd if=/dev/rst0 | tar xf -
```

Replace *remote-host* with the name of the host on your network that has the tape drive you are using. This copies the file `install-medley` to your working directory.

- If the tape drive is local to your machine, type the following:

```
tar xf /dev/rst0
```

This copies the file `install-medley` to your working directory.

If you have any problems during this step of the installation, consult your local UNIX system administrator.

4. Run the installation utility:

```
install-medley
```

Each time you are prompted for information, the script will show you the default in [ ] brackets. Pressing the return key selects the default. To select a different option, type it.

After installing Medley, the script will offer to update two files which must refer to the installation directory. We recommend this. The two files are:

```
medley A script for running Medley easily
```

```
site-init A sample site-init file
```

You can stop the installation process at any time by typing **↑C** (Control-C).

### Using the Installation Script's Menu

To choose an option from the Installation Options Menu, type at least the first three characters of the selection. Most of the menu lists items you might want to install. Choosing one of these options works as a toggle switch, either selecting or deselecting, depending on its previous setting. The other menu items act as commands when you select them. "OS version" lets you pick the SunOS version(s) for which you will need matching emulators. "Directory" lets you specify where to install Medley. "!" lets you use UNIX commands if you need to.

In Figure 2-1, the user has selected for installation the Sysout, Monochrome and X Windows emulators for SunOS 4.1, Display fonts, and Library modules. The menu shows that you need 15052.8 KB of disk space to finish the installation, but only 13002 KB are available. At this point you can either deselect an option to decrease the disk space requirements, or change the installation directory to one that has sufficient disk space.

```

<-----> Installation Options Menu <----->
----- Emulators -----
For one or several OS versions (At least one of monochrome,
color or X11-version is required for new installations)
 x Monochrome - 0.5 MByte
 - Color
 x X11-version - 0.6 MByte
 - XNS - allows use of XNS protocols
 - Object files - allows linking of Medley to other software
 OS version - Change versions. Selected: 4.1
----- Fonts -----
 x Display - 5.5 MByte (recommended)
 - Interpress
----- Sysout, Library & Checksum files -----
 x Sysout - 5.1 MByte (required for new installations)
 x Library modules - 3 MByte (recommended)
 - Checksum files
----- Commands -----
 Directory - Change location of installation directory.
 -- Current: /usr/share/lde 13002 KB
 -- Disk-space(KByte) Available:13002 Needed:15052.8
 !<Unix command> - Execute a Unix command
 ? or Help - Show menu instructions
 Redraw - Redisplay this menu
 None - Unmark all options
 All - Mark all options
 Continue installation
 Quit installation
Select [Directory]:

```

**Figure 2-1. Sample Installation Menu**

Because of the disk space shortage, the script has offered [Directory] as the default next command. If that is what you want, just press Return. Otherwise, type some other command.

For example, to deselect Library modules, type:

```
Select [Directory]: lib
```

Alternately, to find a filesystem with enough disk space, issue the following command:

```
Select [Directory]: !df
```

| Filesystem   | kbytes | used   | avail | capacity | Mounted on |
|--------------|--------|--------|-------|----------|------------|
| /dev/sd0a    | 7548   | 4935   | 1858  | 73%      | /          |
| /dev/sd0g    | 75106  | 51704  | 15891 | 76%      | /usr       |
| /dev/sd1g    | 47999  | 21898  | 21301 | 51%      | /home      |
| king:/shared | 416928 | 349202 | 26042 | 93%      | /share     |

```
Select [Directory]:
```

To change the installation directory, type:

```
Select [Directory]:dir
```

At the new prompt, give the directory name:

```
Where do you want to install Medley? [/usr/share/ldc]: /share/ldc
```

If the directory does not exist, the script will attempt to create it.

---

## Getting a Copy Protection Key

Before starting Medley, you must first obtain a host access key from Venue. This key enables the use of Medley on one workstation. The software cannot run without the key. You must have one key for each host on which you wish to run. Note that your current host access key *will* work if you move from one version of SunOS to another, or if you upgrade from an earlier version of Medley to Medley 2.0.

To obtain a key:

1. Get the host ID of the machine on which you intend to run Medley by typing on that machine

```
prompt% hostid
```

```
310002f6
```

2. Call Venue at 800-228-5325 between 9:00 a.m. and 4:30 p.m. PST. Outside the United States, call your local distributor.
3. Ask for a host access key, giving the Venue representative your host ID. Venue provides you with a host access key, which you need during software configuration, below.
4. If you plan to use the Medley startup script, you will be prompted for the key the first time you invoke it. The script will automatically save the key into a file for future reference.

---

### CAUTION

Depending on your license agreement with Venue, your host access key may have an expiration date. After that date, your key is no longer valid.

---

---

## Changing Configurations or Adding Options

If you forgot to install something or need to add a new emulator, you can rerun the installation, and select only the new things you need. The installation script does not remove things.

This need arises most often when you start running X Windows and need the new emulator.

First, decide what you need to install. Then type **install-medley**. When you get to the Installation Options Menu, select all the new things you need and *deselect* everything else. Continuing the installation from there just adds the new items.

## 2. SOFTWARE INSTALLATION

---

If you have Sun-3s and you just got a SPARCstation, all you need from the new installation are the proper emulators. Everything else is the same.



[This page intentionally left blank]

# LIST of FIGURES

## LIST of TABLES

---

| Figure                                             | Page |
|----------------------------------------------------|------|
| 2-1 Sample Installation Options Menu .....         | 10   |
| 3-1 Sun Type 3 Key Numbering.....                  | 18   |
| 3-2 Sun Type 3 Left Key Pad .....                  | 18   |
| 3-3 Sun Type 3 Right Key Pad .....                 | 18   |
| 3-4 Sun Type 3 Center Key Pad Interpretation ..... | 19   |
| 3-5 Sun Type 4 Key Numbering.....                  | 19   |
| 3-6 Sun Type 4 Left Key Pad .....                  | 20   |
| 3-7 Sun Type 4 Right Key Pad .....                 | 20   |
| 3-8 Sun Type 4 Center Key Pad Interpretation ..... | 20   |

| Table                      | Page |
|----------------------------|------|
| C-1 Font Directories ..... | C-3  |

[This page intentionally left blank]

## 3. GETTING STARTED

---

### Getting Ready

---

To prepare your system to run Medley, use the following checklist:

1. Exit `suntools` or any other windowing system, unless you plan to run under X (in which case you can leave X running). Medley provides its own window system and must not run simultaneously with others.
2. Kill all your user processes (these have console as the control tty). Check to make sure you have killed any "selection\_svc" process. If you do not perform this step, certain error messages from UNIX (e.g., file system full) cause those processes to print to the console, resulting in scrolling of the display.
3. Check for the directory for the software, and add it to your path, if necessary:

```
prompt# set path = ($path /usr/share/lde/install.sunosx)
```

You can also add this to your `.login` file.

### Running Medley

---

#### Running Medley Directly

Invoke Medley by typing the name of the program, e.g.,

```
prompt% lde optional-sysout -k 'thishost-key' [-m memory-size]
```

If you are using either Xerox XNS or Xerox PUP Ethernet protocols, type instead

```
prompt% ldeether optional-sysout -k 'thishost-key' [-m memory-size]
```

This, in turn, runs `lde` and lets it use the Ethernet directly.

*optional-sysout* is the name of a Lisp virtual memory image file (see the section Where Medley Looks for Your Sysout below). *thishost-key* is the key you obtained from Venue for the machine on which you are running.

If the sysout was created on a machine with a different size display, the image will appear garbled for several seconds. After Lisp starts running, it readjusts the display to the current size.

The `-m` flag lets you control the maximum amount of memory Medley will use. *memory-size* is a number in the range 8 through 32, in megabytes. (See the detailed explanation on page 25.)

#### Using the Medley Shell Script

The script will try to find a key, an appropriate emulator, and a sysout. The script relies on information about where the Medley software was initially installed on your system. (The installation script `install-medley` automatically updates this information for you.) The `medley` script assumes that you have not changed the installation subdirectory structure from when it was originally installed.

The script will first try to find a key in the file *Installation directory/.medleyKey.hostname* or in *~/.medleyKey.hostname*. *Installation directory* is where Medley was originally installed on your system. *hostname* is the name of the host for which the license key was issued. If neither file is found, you will be prompted for a key.

`medley` `[[emulator] sysout]` [Command]

---

*emulator*      Given a pathname or a simple name, the command will search for *emulator* as follows:

- If *emulator* is a relative or absolute pathname, e.g., */share/medley/emulators/lde*, it will only try that pathname.
- If *emulator* is a plain file name, e.g., *lde*, the script uses the regular UNIX search path to find it. If it cannot find it, the script looks in the installation directory for Medley at your site, e.g., */usr/share/lde/install.sunos4.1/lde*.
- If you omit *emulator*, the script uses *lde* as the default value, searching for it in the same fashion as above.

*sysout*          The command will search for *sysout* as follows:

- If *sysout* is a relative or absolute pathname, e.g., *../applications/my.sysout*, it will only try that pathname.
- If *emulator* is a plain file name, e.g., *my.sysout*, it will look for it in the following order:
  1. Current working directory *my.sysout*
  2. Installation directory for Medley at your site, e.g., */usr/share/lde/lispsysouts/my.sysout*.
  3. Your home directory, *~/my.sysout*
  4. The *medley* subdirectory in your home directory, *~/medley/my.sysout*.
- If you omit *sysout*, the script looks for it as explained in the Where Medley Looks for Your Sysout section below.

## Examples

- `prompt% medley`  
To start Medley 2.0, a host access key is required.  
Call Venue at (1-800-228-5325) for one,  
and be prepared to give them your workstations host ID#  
Your workstations host ID# is: 51006da3  
Type in key or [^C] to abort: **8bf7723e 459aab34 73491feb**  
Saving key '8bf7723e 459aab34 73491feb' into file  
'*.medleyKey.hostname*' ...  
Trying */usr/share/lde/.medleyKey.hostname* ... Write  
protected !

```
Trying home-directory/.medleyKey.hostname ... Done
Starting up Medley 2.0 ...
..... Medley 2.0 starts
```

If you had Medley installed in `/share/medley` on your system, it would try to run the emulator `/share/medley/install.sunos4.1/lde`, using the `sysout` `/share/medley/lispsysouts/LISP.SYSOUT`.

In this example you are prompted for a key, which is saved into the file: `home-directory/.medleyKey.hostname`

The script tried to save the key into the installation directory but did not have write access there. Instead it was put into your home directory (`~`). `hostname` is the name of the host running medley.

The next time you use the script `medley`, you will not be prompted for the key.

- `prompt% medley application.sysout`

If you had `application.sysout` in your home directory, it would try running the emulator `/share/medley/install.sunos4.1/lde` using `~/application.sysout`.

---

## Where Medley Looks for Your Sysout

If you run Medley directly, the system searches the following places, in order, for the `sysout` to be used:

- *command line*

The name of the `sysout` file can be given on the command line when starting Medley; for example,

```
prompt% lde sysout -k 'thishost-key'
```

- `LDESRCESYSOUT`

If no `sysout` file name is given on the command line, the value of the environment variable `LDESRCESYSOUT` is used as the name of the `sysout` file. For example:

```
prompt% setenv LDESRCESYSOUT my.sysout
prompt% lde -k 'thishost-key'
```

would run the host key `my.sysout`.

- `~/lisp.virtualmem`

Finally, Medley looks for the file `lisp.virtualmem` on your home directory.

---

## Where Medley Looks for Your Site Initialization File

When Medley starts, it reads in a Lisp site initialization file. This site initialization file sets things like pathnames for fonts, site parameters, and the like.

Greeting and initialization are described in the *Interlisp-D Reference Manual*, Section 12.1.

Medley looks for a site initialization file in a number of locations:

- LDEINIT

If the environment variable LDEINIT is set to a complete Lisp file name, Lisp looks there first for the site initialization file:

```
prompt% setenv LDEINIT /usr/lisp/my-site-init.lisp
```

- /usr/share/lde/site-init.lisp

If LDEINIT is not set or there is no file with the name given, Lisp looks for a site initialization file called /usr/share/lde/site-init.lisp. The distribution tape contains a sample site initialization file in the Lisp library directory /usr/share/lde/lisplibrary/site-init. The system administrator should copy site-init into /usr/share/lde/site-init.lisp then customize it for the site. The comments in the sample site-init.lisp describe the parameters it sets and give guidelines for customizing it to your local conditions.

- {DSK}INIT.DFASL, {DSK}INIT.LCOM, {DSK}INIT.LISP

Finally, Lisp looks for a site initialization file on your Medley home directory ({DSK}). Chapter 5, Medley File Systems, describes the {DSK} device.

---

## Medley and X Windows

Medley 2.0-S supports the X Window System, Version 11 Release 4 (X11R4). Medley runs in a single X window; Medley's "screen" is displayed in that window, and you use Medley as usual.

### Starting X Windows

Start the X server on your console. Use the `xinit` command.

If necessary, start a window manager as a client of X (`xinit` often starts a window manager). The window manager provides many window management functions, such as moving, resizing and iconifying the window. Medley has no window management function of its own.

### Running Medley Remotely

You can run Medley on one machine, with the window on some other machine. To do so, perform these steps on the machine whose keyboard and display you will be using:

1. Add the host name to execute the Medley access control list:

```
xhost + hostname
```

2. Open a new xterm and rlogin to the Sun Workstation on which Medley is to run. Set the environment variable DISPLAY to the host name of the server machine:

```
setenv DISPLAY servername:0
```

3. Set the LDEKBDTYPE environment variable to tell Medley what kind of keyboard you will be using. Possible values are:

```
type3 Sun Type 3 keyboard
type4 Sun Type 4 keyboard
rs6000 IBM RS/6000 or PS/2
dec3100 DECstation 3100 or 5000
```

|        |                          |
|--------|--------------------------|
| hp9000 | HP9000 Series 700 or 800 |
| X      | Generic X terminal       |

If you don't set `LDEKBDTYPE`, it will default to `X`. The advantage of specifying a specific keyboard lies in how Medley treats the special function keys. The specific keyboard maps maximize the usefulness of keys marked, e.g., "Find". The generic keyboard code cannot do that reliably.

#### 4. Start up Medley.

A new window for Medley will appear on the X server's screen.

### The Medley Window

Normally, Medley uses the whole screen. Under X, Medley's "screen" appears in a single X window. Medley's screen is slightly smaller than the screen you are using to display it; if you make the X window full-screen-size, you see Medley's entire screen. If it is smaller, you will need to scroll to see parts of the screen.

The scroll bars (at the right and bottom of the X window) control what parts of Medley's screen appears in the window. Use the vertical scrollbar to scroll up and down, and the horizontal scrollbar to scroll left and right. The gravity buttons (at the lower right corner) set the bitgravity of the display window. Click the mouse button on one of these areas. The shade pattern is moved to the clicked area, and the bitgravity is set in the corresponding corner on the display window. The bitgravity determines how reshaping the X window affects what part of the Medley screen is visible.

## Environment Variables

---

Medley on the Sun uses several environment variables. They can be set from the shell with the **setenv** UNIX command. By convention, environment variable names use uppercase rather than lowercase letters, e.g., `LDEDESTSYSOUT`. The Medley environment variables are listed below, with a reference to sections in this *Guide* where further information can be found.

|                             |                                                                               |
|-----------------------------|-------------------------------------------------------------------------------|
| <code>LDEKBDTYPE</code>     | See the Medley and X Windows or Sun Type 4 Keyboard sections in this chapter. |
| <code>LDEINIT</code>        | See the Site Initialization File section in Chapter 4.                        |
| <code>LDESRCESYSOUT</code>  | See the Where Medley Looks for Your Sysout section in this chapter.           |
| <code>LDEDESTSYSOUT</code>  | See the Saving Your State section in Chapter 4.                               |
| <code>LDESHELL</code>       | See the UNIXCHAT section of the <i>Lisp Library Modules</i> .                 |
| <code>LDEFILETIMEOUT</code> | See the File System Errors subsection in Chapter 5.                           |

## Keyboard Interpretation

---

This section describes how Medley interprets the Sun Type 3 and Type 4 keyboards. Except when running under X, Medley performs its own keyboard interpretation, taking raw up/down transitions directly from the keyboard. Medley uses its own key



### 3. GETTING STARTED

numbering scheme; key numbers are used by Lisp functions such as IL:KEYDOWNP and IL:KEYACTION.

These key assignments were chosen to maximize compatibility with both the Xerox workstation keyboard and the normal Sun keyboards. You can attach a Sun Type 3 or Type 4 keyboard template, which also shows the Medley keyboard assignments, to your Sun Type 3 or Type 4 keyboard. Both templates are included with your Medley documentation set.

#### Sun Type 3 Keyboard

Figure 3-1 shows the key number assignments for the Sun Type 3 keyboard. Figures 3 - 2 through 3 - 4 show Medley's key assignments for the Sun Type 3 keypads.

|     |    |    |    |     |    |    |    |    |    |     |    |    |    |     |     |    |    |    |    |    |
|-----|----|----|----|-----|----|----|----|----|----|-----|----|----|----|-----|-----|----|----|----|----|----|
| 61  | 91 | 97 | 99 | 100 |    | 67 |    | 68 |    | 101 |    | 66 |    | 104 | 80  | 13 | 73 | 74 | 75 |    |
| 92  | 63 | 33 | 32 | 17  | 16 | 1  | 0  | 2  | 4  | 53  | 22 | 8  | 10 | 59  | 105 | 45 | 81 | 82 | 83 |    |
| 14  | 62 | 34 | 19 | 18  | 3  | 48 | 49 | 51 | 6  | 23  | 25 | 11 | 58 | 29  | 15  |    |    | 84 | 85 | 87 |
| 111 | 89 | 36 | 21 | 20  | 5  | 35 | 50 | 52 | 38 | 9   | 26 | 43 | 28 | 44  |     |    | 94 | 69 | 70 |    |
| 90  | 46 | 41 |    | 40  | 24 | 37 | 7  | 39 | 54 | 55  | 27 | 42 | 12 | 60  | 71  | 98 |    |    | 76 | 72 |
|     |    | 56 | 31 | 57  |    |    |    |    |    |     |    |    |    | 93  | 47  |    |    |    |    |    |

**Figure 3-1. Sun Type 3 Key Numbering**

|      |        |
|------|--------|
| Stop | Again  |
| Help | Undo   |
| Same | Move   |
| Open | Copy   |
| Find | Delete |

**Figure 3-2. Sun Type 3  
Left Key Pad**

|             |                |              |
|-------------|----------------|--------------|
| Num<br>Lock | Scroll<br>Lock | Break        |
| 7<br>Home   | 8<br>↑         | 9<br>PgUP    |
| 4<br>←      | 5              | 6<br>→       |
| 1<br>End    | 2<br>↓         | 3<br>PgDN    |
| Ins         | DOIT           | Caps<br>Lock |

**Figure 3-3. Sun Type 3  
Right Key Pad**

|        |        |        |        |         |        |           |        |           |        |              |                   |        |              |
|--------|--------|--------|--------|---------|--------|-----------|--------|-----------|--------|--------------|-------------------|--------|--------------|
| Center | Bold   | Italic |        | Case    |        | Strikeout |        | Underline |        | Super<br>Sub | Larger<br>Smaller | Margin | Back<br>Word |
| Esc    | !<br>1 | @<br>2 | #<br>3 | \$<br>4 | %<br>5 | ^<br>6    | &<br>7 | *<br>8    | (<br>9 | )<br>0       | -<br>=            | <br>_  | ~<br>,       |
| Tab    | Q<br>q | W<br>w | E<br>e | R<br>r  | T<br>t | Y<br>y    | U<br>u | I<br>i    | O<br>o | P<br>p       | {<br>[            | }<br>] | Backspace    |
| Ctrl   | A<br>a | S<br>s | D<br>d | F<br>f  | G<br>g | H<br>h    | J<br>j | K<br>k    | L<br>l | :<br>;       | "<br>,            | Return |              |
| Shift  |        | Z<br>z | X<br>x | C<br>c  | V<br>v | B<br>b    | N<br>n | M<br>m    | <<br>, | ><br>.       | ?<br>/            | Shift  | LF           |
| Caps   | Meta   | Space  |        |         |        |           |        |           |        |              |                   | Expand | Next         |

**Figure 3-4. Sun Type 3 Center Key Pad Interpretation**

### Sun Type 4 Keyboard

Figure 3-5 illustrates the keyboard interpretation for the Sun Type 4 keyboard. Figures 3-6 through 3-8 show the keyboard and the left and right key pads for the Sun Type 4 keyboard.

NOTES: In SunOS 4.0, the NEXT (ALT/GRAPH) key on the Type 4 keyboard is inaccessible. Later versions of SunOS fix this.

Medley cannot detect whether it is running on a workstation with a Type 4 keyboard when running SunOS 4.0, 4.0.1, or 4.1. To make it work correctly on your workstation, enter the following before you start running Medley:

```
setenv LDEKBDTYPE type4
```

|     |    |    |    |     |    |    |     |    |     |    |     |     |     |     |    |    |     |    |     |
|-----|----|----|----|-----|----|----|-----|----|-----|----|-----|-----|-----|-----|----|----|-----|----|-----|
| 61  | 91 | 97 | 99 | 100 | 67 | 68 | 101 | 66 | 104 | 80 | 106 | 107 | 108 | 105 | 13 | 75 | 110 | 74 | 73  |
| 109 | 63 | 33 | 32 | 17  | 16 | 1  | 0   | 2  | 4   | 53 | 22  | 8   | 10  | 59  | 15 | 64 | 65  | 95 | 96  |
| 14  | 89 | 34 | 19 | 18  | 3  | 48 | 49  | 51 | 6   | 23 | 25  | 11  | 58  | 29  | 0  | 81 | 82  | 83 | 102 |
| 111 | 62 | 36 | 21 | 20  | 5  | 35 | 50  | 52 | 38  | 9  | 26  | 43  | 28  | 45  |    | 84 | 85  | 87 |     |
| 90  | 46 | 41 | 40 | 24  | 37 | 7  | 39  | 54 | 55  | 27 | 42  | 12  | 60  | 71  |    | 94 | 69  | 70 | 76  |
| 92  |    | 56 | 31 | 86  | 57 |    |     |    |     |    |     |     | 88  | 93  | 47 | 98 | 13  |    |     |

**Figure 3-5. Sun Type 4 Key Numbering**

|       |        |
|-------|--------|
| Stop  | Again  |
| Props | Undo   |
| Same  | Copy   |
| Open  | Move   |
| Find  | Delete |
| Help  |        |

**Figure 3-6. Sun Type 4  
Left Key Pad**

|           |        |                |             |
|-----------|--------|----------------|-------------|
| Break     | PrSc   | scroll<br>lock | num<br>lock |
| =         | /      | *              | -           |
| 7<br>Home | 8<br>↑ | 9<br>PgUP      | +           |
| 4<br>←    | 5      | 6<br>→         |             |
| 1<br>End  | 2<br>↓ | 3<br>PgDN      | DOIT        |
| Ins       |        | Del            |             |

**Figure 3-7. Sun Type 4  
Right Key Pad**

|              |            |              |            |              |             |             |             |              |        |        |        |              |                |        |
|--------------|------------|--------------|------------|--------------|-------------|-------------|-------------|--------------|--------|--------|--------|--------------|----------------|--------|
| F1<br>Center | F2<br>Bold | F3<br>Italic | F4<br>Case | F5<br>Strike | F6<br>Under | F7<br>Super | F8<br>Large | F9<br>Margin | F10    | F11    | F12    | <br>\<br>_   | Delete<br>Word |        |
| Esc          | !<br>1     | @<br>2       | #<br>3     | \$<br>4      | %<br>5      | ^<br>6      | &<br>7      | *<br>8       | (<br>9 | )<br>0 | -<br>_ | +<br>=       | Back<br>Space  |        |
| Tab          | Q<br>q     | W<br>w       | E<br>e     | R<br>r       | T<br>t      | Y<br>y      | U<br>u      | I<br>i       | O<br>o | P<br>p | {<br>[ | }<br>]       |                | Return |
| Ctrl         | A<br>a     | S<br>s       | D<br>d     | F<br>f       | G<br>g      | H<br>h      | J<br>j      | K<br>k       | L<br>l | :<br>; | "<br>, | ~<br>'       |                |        |
| Shift        |            | Z<br>z       | X<br>x     | C<br>c       | V<br>v      | B<br>b      | N<br>n      | M<br>m       | <<br>, | ><br>. | ?<br>/ | Shift        | LF             |        |
| Caps         | Meta       | Left<br>Spc  | Space      |              |             |             |             |              |        |        |        | Right<br>Spc | Expand         | Next   |

**Figure 3-8. Sun Type 4 Center Key Pad Interpretation**

[This page intentionally left blank]

# PREFACE

---

This *Guide* describes Medley release 2.0 for the Sun-3 and Sun-4 workstations and the SPARCstation: the release contents, instructions for installing the release, and information on using it.

## Audience

---

The *Medley For the Sun Workstation® User's Guide* is intended for users familiar with the Medley environment who want to use it on the Sun-3 or Sun-4 workstations or the SPARCstation. The *Guide* assumes that the user is already familiar with UNIX and SunOS concepts. The system administrator of a Sun system or network should read this *Guide* to ensure the correct installation of the Medley software.

Chapter 1 of this manual gives an overview of the product and its internal architecture, and is of interest to all users of the system.

System administrators should read Chapter 2, Software Installation; and Chapter 3, Getting Started. These chapters guide the administrator through the process of installing Medley 2.0 and configuring it on the Sun Workstation. Experienced Lisp users may want to configure the software; this procedure is described in Chapter 4.

Users already familiar with the Lisp environment on Xerox workstations should find Chapter 1 and Chapters 3 through 6 useful. These chapters describe the operation of the system after it has been installed as well as those functions and operations which are specific to the Sun Workstation.

## Using This Manual

---

**Chapter 1, Introduction**, describes the hardware, input/output devices, and software needed to run Medley on a Sun Workstation; describes Medley and how it works with other applications; lists the system components; introduces pertinent SunOS and UNIX conventions used throughout the *Guide*; explains Medley's compatibility; and lists the contents of the release.

**Chapter 2, Software Installation**, contains the installation and software configuration procedures.

**Chapter 3, Getting Started**, explains how to set up a site initialization file and install the X Windows System. It also shows the keyboard configuration and has instructions for getting started in Lisp on the Sun Workstation.

**Chapter 4, Using Medley on the Sun Workstation**, describes how specific Lisp functionality works on the Sun.

**Chapter 5, Medley File Systems**, discusses the file conventions that need to be followed when running in Medley on a Sun Workstation. Differences in Lisp file attributes and variables are also discussed.

**Chapter 6, Error Recovery**, describes the diagnostic error recovery program URAID. This chapter explains how to recover from fatal error conditions and lists specific Lisp errors that may be encountered when running Medley on the Sun.

**Appendix A, Installation Hints**, contains additional notes to help configure Medley, and includes a complete description of the installation script.

**Appendix B, Verifying the Installation Tape's Validity**, tells how to validate the contents of the tar tape.

**Appendix C, Layout of Installation Tape Files**, includes a listing of the tar tape directories and the font directories.

**Appendix D, Differences between Xerox Workstations and the UNIX Version of Medley**, includes functions for controlling device-specific behavior of the Xerox 1100 series workstation disk drives. It also describes the library modules not supported on the Sun.

The **Glossary** provides definitions of SunOS, UNIX, and Lisp terms used in this *Guide*.

Medley is a Venue product which was built on the Xerox Lisp environment. It provides an integrated programming environment consisting of Interlisp-D and Common Lisp, a windowing system, and a set of programs and utilities. Users not already familiar with the Xerox Lisp environment should try to become somewhat familiar with it before attempting serious development work.

## Supporting Documentation

---

The following reference documents are useful to have on hand during the installation process and when working in Medley on the Sun Workstation.

### Sun References

This literature from the Sun documentation set is useful during the installation and when running Medley on a Sun Workstation.

- *Installing UNIX on the Sun Workstation*
- *UNIX Interface Reference Manual*
- *SunOS Reference Manual*
- *Sun Software Technical Bulletin, March 1988*

### Venue Documentation

In addition to this *Guide*, the following documents describe the Medley system:

- Guy Steele, *Common Lisp, the Language, First Edition*, Bedford, MA: Digital Press, 1987
- *Interlisp-D Reference Manual, Volumes 1-3*, Koto Release
- *Xerox Common Lisp Implementation Notes*, Lyric Release
- *Lisp Documentation Tools*, Lyric Release
- *Lisp Library Modules*, Medley Release
- *Lisp Release Notes*, Medley Release

Templates for the Type 3 and Type 4 Sun keyboards are also part of the Medley documentation set.

New users of Medley receive, in the software kit, all the manuals listed above.

Users who are moving the Medley environment from a Xerox workstation to a Sun Workstation receive the following documentation in the software kit:

- *Lisp Release Notes*, Medley Release
- *Lisp Library Modules*, Medley Release
- *Medley for the Sun Workstation® User's Guide*
- Sun Type 3 and Type 4 keyboard templates

The manual *LispUsers' Modules*, Medley Release, which may be purchased separately, supplements the Medley release.

[This page intentionally left blank]



## 4. USING MEDLEY ON THE SUN WORKSTATION

---

Once the system administrator has installed Medley software on the Sun, Lisp users can customize their Medley Lisp environments. This chapter provides basic information to get you started in the Medley environment on a Sun Workstation.

### Setting Up a Site Init File

---

The users at a given site generally print to the same printers, load library files from the same directory, and so on. Medley uses variables to supply defaults for such things. The obvious place to set these variables is in one common initialization file. That is the Site Init File's role.

The Site Init File is a file of Lisp expressions that is loaded when you start Medley with a fresh `LISP.SYSOUT`.

The following Lisp symbols should be set in your site init file:

---

|                          |                   |
|--------------------------|-------------------|
| <b>IL:USERGREETFILES</b> | <b>[Variable]</b> |
|--------------------------|-------------------|

---

A list of templates to search for the place where individuals should find their personal init files. If this is not set in the site init file, no personal init file is used. The list should be similar to the following:

```
(({file-server}< USER >LISP>INIT.LCOM)
 ({file-server}< USER >LISP>INIT)
 ({file-server}< USER >INIT.LISP))
```

---

|                                  |                   |
|----------------------------------|-------------------|
| <b>IL:DISPLAYFONTDIRECTORIES</b> | <b>[Variable]</b> |
|----------------------------------|-------------------|

---

A list of directories to search when the system is looking for display fonts. The site initialization file should set it to a list of strings, each containing a complete pathname for font files, e.g., ("`{UNIX}/usr/local/lde/fonts/display/presentation/`").

---

|                                     |                   |
|-------------------------------------|-------------------|
| <b>IL:INTERPRESSFONTDIRECTORIES</b> | <b>[Variable]</b> |
|-------------------------------------|-------------------|

---

A list of directories to search when the system is looking for Interpress font widths.

---

|                       |                   |
|-----------------------|-------------------|
| <b>IL:DIRECTORIES</b> | <b>[Variable]</b> |
|-----------------------|-------------------|

---

The list of paths to search for files that are not found in the current (Lisp) connected directory.

---

|                                |                   |
|--------------------------------|-------------------|
| <b>IL:LISPUSERSDIRECTORIES</b> | <b>[Variable]</b> |
|--------------------------------|-------------------|

---

The list of paths to search for library and LispUsers' files. Remember that every path in this list should also be in `DIRECTORIES`.

---

|                               |                   |
|-------------------------------|-------------------|
| <b>IL:DEFAULTPRINTINGHOST</b> | <b>[Variable]</b> |
|-------------------------------|-------------------|

---

A list of names of default printers.

**IL:DEFAULTPRINTERTYPE** **[Variable]**

The default printer type, e.g., POSTSCRIPT.

**XCL:\*LONG-SITE-NAME\*** **[Variable]**

The value of the function XCL:LONG-SITE-NAME, e.g., "Frobnitz, Baz and Lispers, Incorporated."

**XCL:\*SHORT-SITE-NAME\*** **[Variable]**

The value of the Common Lisp function XCL:SHORT-SITE-NAME, e.g., "Frobco".

**IL:\BeginDST** **[Variable]**

The day of the year on or before which Daylight Savings Time takes effect (i.e., the Sunday on or immediately preceding this day). Must be set to 98 in the USA if Lisp is to perform time computations correctly (subject, of course, to future legislation). If you are in a region where Daylight Savings Time is not observed, set the value to 367.

**IL:\EndDST** **[Variable]**

The day of the year on or before which Daylight Savings Time ends. Must be set to 305 in the USA.

---

## Setting Up a Personal Init File

Your personal init file keeps track of the location of your home directory and windows layout; it also remembers which library files you always load.

Your personal init file is a file of Lisp expressions that is loaded and run after the site init file. You can create it either as a text file, or have Medley's File Manager help you.

Your initialization file is normally ~/INIT.LCOM

---

## Saving Your State

On the Sun, lde is an ordinary UNIX program that allocates a 45 MB data area, reads into that area several megabytes of data (the sysout), and modifies it there. Under UNIX, that program's data requirements (which include the sysout) are handled by UNIX; all Medley does is modify in "memory" a copy of your original sysout file. UNIX, transparently to Medley, handles all real memory swapping. This has several consequences related to starting, saving, and restarting sysouts.

On Xerox workstations, the virtual memory partition is updated periodically and used to store new pages as they are allocated or flushed from the real memory of the machine. For example, LOGOUT and SAVEVM write out only those pages of data which are different from what might already be in the virtual memory file.

On the Sun Workstation, however, the contents of virtual memory are only written to a file by an explicit call to SAVEVM, LOGOUT, SYSOUT, or MAKESYS. This file is an ordinary SunOS file (normally ~/lisp.virtualmem). The entire virtual memory, which may be many megabytes of data, is written out there.

On the Sun Workstation, starting anew from a saved virtual memory file requires reading it into memory. On the Xerox workstation, it is necessary to first copy the

saved sysout to the virtual memory file and then read it in. Thus, restarting a saved sysout or virtual memory file is significantly faster on a Sun Workstation.

The file that LOGOUT and SAVEVM writes is normally `~/lisp.virtualmem` (i.e., the file `lisp.virtualmem` on the user's home directory). However, the environment variable `LDEDESTSYSOUT` can be used to override this default. For example, you might want to keep virtual memory images on `/user/local`. During a demonstration where you do not want the memory image saved, you can reset `LDEDESTSYSOUT` to `/dev/null`. You can use the C-Shell command `setenv` to do this, e.g.:

```
prompt% setenv LDEDESTSYSOUT "/dev/null"
```

Cursor tracking interferes with writing out the screen bitmap as part of the Medley memory image. For this reason, Medley takes the cursor down before saving a virtual memory image as part of LOGOUT, SAVEVM, SYSOUT, or MAKESYS. When this happens, the message

```
Saving VMem, taking mouse down
```

appears in the prompt window, and cursor tracking is disabled.

Because the virtual memory file need not already exist to run Medley, the functions LOGOUT and SAVEVM can signal the following file errors:

```
File-System-Resources-Exceeded
Protection-Violation
File-Wont-Open
```

Even if some errors occur while saving a virtual memory, the old destination file is safe. Saving does not overwrite the old virtual memory file. The saving virtual memory file is named with "-temp", such as `lisp.virtualmem-temp`. The file is renamed to a specified name, such as `lisp.virtualmem`, at the last sequence of the save.

When the user does not have enough space to save the virtual memory, the old virtual memory file can be overwritten by setting `IL: \LDEDESTOVERWRITE` to `T`. The initial value of `IL: \LDEDESTOVERWRITE` is `NIL`. In some cases, even if the user tries to overwrite, there may still not be enough space.

In Medley, a "page" is 512 bytes. Under SunOS, the page size is variable; some Sun Workstations use 8 Kbyte pages. In general, Medley functions deal only in units of Medley pages, e.g., the `SIZE` attribute of files is in terms of 512-byte pages, (`VMEMSIZE`) returns the number of 512-byte pages in use.

#### **(IL: LOGOUT *FAST*)**

#### **[Function]**

Lets you exit Medley cleanly. The parameter *FAST* indicates whether resumption of the same environment is desirable and in what fashion. Before exiting, disk buffers are written, and network connections subject to timeout are closed.

If *FAST* is `NIL`, LOGOUT first saves your virtual memory in a file. Change the file name by setting the UNIX environment variable `LDEDESTSYSOUT`. If this variable is not set, the file saved is `~/lisp.virtualmem` (i.e., `lisp.virtualmem` on the user's home directory).

If *FAST* is `T`, Medley stops without writing the virtual memory file. It is not possible to resume execution in the same image.

### **(IL:SAVEVM )**

**[Function]**

Saves your state, but does not exit. It causes the current virtual memory image to be written to the location specified by the environment variable `LDEDESTSYSOUT`, if this variable is set; otherwise it is written to `~/lisp.virtualmem`. This allows Lisp to continue. Execution in Medley continues after memory is saved; thus, `SAVEVM` operates as a sort of checkpoint of the current working state. `SAVEVM` can cause the following error:

File-System-Resources-Exceeded.

### **(IL:SYSOUT FILE)**

**[Function]**

Performs the equivalent of `SAVEVM` and then copies the saved image to *FILE* for devices other than {DSK} and {UNIX} (e.g., XNS file servers). (See Chapter 5, Medley File Systems, for further information on {DSK} and {UNIX}.) `SYSOUT` can cause the following error:

File-System-Resources-Exceeded.

## Sun-Specific Environment Functions

---

### System Environment Functions and Variables

These functions, which interrogate the system environment, operate as described below when they are invoked on the Sun Workstation:

### **(IL:REALMEMORYSIZE)**

**[Function]**

On some machines, returns the total amount of real memory available; does not work on a Sun Workstation (i.e., returns a meaningless value).

### **(CL:MACHINE-TYPE)**

**[Function]**

Returns a string identifying the type of computer hardware the system is running under. On the Sun-3 workstation `MACHINE-TYPE` returns "mc68020". On a Sun-4 workstation, `MACHINE-TYPE` returns the string "sparc".

### **(IL:MACHINETYPE)**

**[Function]**

Identifies the generic type of Lisp machine in use. On the Sun Workstation, it returns the symbol `IL:MAIKO`.

### **(CL:MACHINE-VERSION)**

**[Function]**

Returns a string identifying the version of the emulator running; e.g., "Microcode version: 279, memory size: 16384".

### **(CL:MACHINE-INSTANCE)**

**[Function]**

Returns a string containing the workstation host ID (in hexadecimal) and the host name.

**IL:LISP-RELEASE-VERSION****[Variable]**

Identifies the release number within a single major release name. In Medley 2.0, IL:LISP-RELEASE-VERSION is 2.0. While IL:MAKESYSNAME does not change, IL:LISP-RELEASE-VERSION always changes with each new sysout release. This variable did not exist in the Medley 1.0-S sysout.

**IL:\MY.NSADDRESS****[Variable]**

Fills in the fields of the network address with the host ID if Medley is run without the Ethernet enabled. Programs that use the network address as a unique identifier should be aware that the value could vary from session to session depending on whether or not the Ethernet is enabled. (Refer to Chapter 14 of the *Interlisp-D Reference Manual* for further information.)

## VM Functions

The biggest difference is a change in terminology. On Xerox 1100 series workstations, Lisp itself handles all virtual memory operations directly, so the terms "sysout" and "virtual memory image" can be used interchangeably. The running sysout resides in a reserved area on the workstation local disk (the virtual memory partition) that Lisp reads from and writes to as it needs to move pages into and out of physical memory.

**(IL:VMEMSIZE)****[Function]**

Returns the number of 512-byte pages of the Medley virtual memory that are in use. This number is a good estimate of the size of a SYSOUT, MAKESYS, or SAVEVM virtual memory file.

**(IL:VMEM.PURE.STATE ON/OFF)****[Variable]**

Has no effect on the Sun Workstation. The virtual memory file is not modified except by an explicit (LOGOUT) or (SAVEVM).

**IL:BACKGROUNDPAGEFREQ****[Variable]**

Has no effect on the Sun Workstation. The virtual memory file is not modified except by an explicit (LOGOUT) or (SAVEVM).

You can control how much virtual memory Medley uses by using the -m switch, as described below.

**ldeether [<SYSOUT-name>] [-m<memory-size>] [other options]****[Command]**

Allows you to specify an arbitrary virtual memory size for Medley.

-m Specifies the memory size

memory-size 8 through 32 Mbytes

When you use -m, the value of IL:\STORAGEFULLSTATE in the sysout you start should not be 3 or 4. Those values mean it already used more than the 8-Mbyte space in the sysout. Because of the Medley storage management architecture, the virtual memory size cannot be changed after IL:\STORAGEFULLSTATE has been set to 3 or 4. This value can be examined just before (IL:LOGOUT) if you want to specify the virtual memory size during the next start-up.

Example: `ldeether /usr/LISP.SYSOUT -m 16`

This example means 16 Mbytes of virtual space will be assigned for Lisp.

### Stopping Lisp Temporarily

**(IL:SUSPEND-LISP)**

**[Function]**

Suspends, temporarily, the UNIX process running Medley. Using the `fg` C-Shell command, the Medley process can be continued from the C-Shell where it was started. `SUSPEND-LISP` has no effect on Xerox Lisp workstations. This function should not be used during I/O operations (file or network).

## Login Functions

This section describes the interaction between the usernames and passwords in Medley and the SunOS usernames and passwords. The functions `IL:USERNAME`, `IL:SETUSERNAME`, `IL:SETPASSWORD`, and `IL:LOGIN` access the username/password database used by Medley in network operations. (For further information, see Chapter 24 of the *Interlisp-D Reference Manual*.) When Medley is started, this database contains only the SunOS username, with no password. Except for this, there is no interrelation between these Medley functions and SunOS usernames and passwords.

`IL:USERNAME` returns the SunOS login name under which the emulator was started. A subsequent `IL:SETUSERNAME` or `IL:LOGIN` changes `IL:USERNAME`, and the default login name for network access to XNS and PUP hosts. However, it does not change the SunOS login name or access capabilities for files on `{DSK}` or `{UNIX}`. (See Chapter 5, Medley File Systems, for detailed information on `{DSK}` and `{UNIX}`.) Because it doesn't change the SunOS login name, it won't change the author name on SunOS files created from Lisp.

The following functions apply to login activities.

|                                  |                          |
|----------------------------------|--------------------------|
| <b><u>(IL:UNIX-USERNAME)</u></b> | <b><u>[Function]</u></b> |
|----------------------------------|--------------------------|

Returns a string consisting of the username of the SunOS process running Medley. Returns NIL if one of the following conditions apply:

- You are not running under UNIX
- You do not have a full name entered in `/etc/passwd` or the NIS password map
- An error occurs.

|                                  |                          |
|----------------------------------|--------------------------|
| <b><u>(IL:UNIX-FULLNAME)</u></b> | <b><u>[Function]</u></b> |
|----------------------------------|--------------------------|

Returns a string containing the full name of the owner of the SunOS process running Medley. Returns NIL if the user is not running under UNIX or an error occurs.

|                                                 |                          |
|-------------------------------------------------|--------------------------|
| <b><u>(IL:LOGIN HOST FLG DIRECTORY MSG)</u></b> | <b><u>[Function]</u></b> |
|-------------------------------------------------|--------------------------|

Attempts to maintain user IDs and passwords for network as well as local access. If `HOST` is NIL, this function attempts to perform the SunOS `setuid` operation.

Unless you are running as root, this will not change your SunOS login.

## Environment Inquiry

The following functions return the values of UNIX environment variables or machine parameters. They return NIL if run in Medley on Xerox 1100 series workstations.

|                                       |                          |
|---------------------------------------|--------------------------|
| <b><u>(IL:UNIX-GETENV STRING)</u></b> | <b><u>[Function]</u></b> |
|---------------------------------------|--------------------------|

Returns the value of the environment variable with the given name. The argument `STRING` should be the name of a UNIX environment variable. For example, `(UNIX-GETENV "HOME")` might return the user's home directory.

|                                        |                          |
|----------------------------------------|--------------------------|
| <b><u>(IL:UNIX-GETPARM STRING)</u></b> | <b><u>[Function]</u></b> |
|----------------------------------------|--------------------------|

Returns the value of one of a few built-in parameters. The argument `STRING` should be the name of one of the following UNIX environment variables:

| Variable   | If running on<br>this hardware | Returns                                                                 |
|------------|--------------------------------|-------------------------------------------------------------------------|
| "MACH"     | Sun-4                          | "sparc"                                                                 |
|            | Sun-3                          | "mc68000"                                                               |
|            | RS/6000                        | "rs/6000"                                                               |
|            | HP9000                         | "hp9000"                                                                |
|            | DEC3100                        | "mips"                                                                  |
|            | PS/2                           | "i386"                                                                  |
| "ARCH"     | Sun-4                          | "sun4"                                                                  |
|            | Sun-3                          | "sun3"                                                                  |
|            | RS/6000                        | "rs/6000"                                                               |
|            | HP9000                         | "hp9000"                                                                |
|            | DEC3100                        | "dec3100"                                                               |
|            | PS/2                           | "ps/2"                                                                  |
| "HOSTNAME" | All                            | Returns the local host name                                             |
| "HOSTID"   | All                            | Returns the local host identification number<br>as a hexadecimal string |

## Display and Keyboard Functions and Variables

Some Medley display and keyboard functions and variables operate differently on the Sun Workstation.

The following functions have no effect on a Sun Workstation, and always return NIL:

IL:CHANGEBACKGROUND BORDER

IL:VIDEORATE

IL:SETMAINTPANEL

IL:VIDEOCOLOR

The functions IL:BEEPON, IL:BEEPOFF, IL:PLAYTUNE, IL:RINGBELLS generate monotones.

---

**(IL:BEEPON *FREQ*)** **[Function]**

---

Turns on the keyboard tone generator on the Sun Workstation. The *FREQ* argument is ignored.

---

**(IL:BEEPOFF)** **[Function]**

---

Turns off the keyboard tone generator.

---

**(IL:PLAYTUNE *TUNEPAIRS*)** **[Function]**

---

Sounds tones, but ignores the frequencies of the values in *TUNEPAIRS*.

---

**(IL:RINGBELLS)** **[Function]**

---

Causes the machine to beep several times.

## Timers and Clocks

UNIX is a timesharing operating system. When Medley is running, other programs can be running at the same time on the same workstation.



On a Xerox workstation running Lisp, CPU time could be computed exactly from elapsed time after subtracting known system overhead. To allow older Interlisp-D programs to work unchanged, the timer functions were modified to allow programs that accounted for time on Xerox workstations to continue to run. Time is categorized as follows:

|                |                                                                                           |
|----------------|-------------------------------------------------------------------------------------------|
| CPU time:      | The total amount of time spent executing Medley's process in user mode.                   |
| SWAP time:     | The total time spent running other processes (Elapsed time – (CPU time + Disk time)).     |
| Disk I/O time: | The total amount of time spent in the system executing on the behalf of Medley's process. |

The Medley functions `CLOCK`, `TIME`, and the like get the time of day directly from SunOS. The function `SETTIME` has no effect on the Sun Workstation.

**IL: \RCLKMILLISECOND**

**[Variable]**

The number of clock "ticks" in a millisecond. On the Sun Workstation, this value is always 1000. All of the timer functions that deal in clock ticks will do their computation in microseconds. Note, however, that the Sun Workstation does not have that accurate a clock resolution. While clock resolution varies from one operating system version to another, it often has a resolution no better than 1/60th of a second.

## Miscellaneous Operational Differences

The stack and virtual memory handling functions on the Sun Workstation are implemented differently from the way they are on the Xerox workstations. For this reason, the "cursor bars" used on the Xerox workstations are not used on the Sun Workstation.

When working in Medley on a Sun workstation, you should periodically load a fresh sysout. Older Medley sysouts don't run as well as "fresh" sysouts due to a number of factors such as fragmentation of memory, increased working set, more objects taking up various spaces (e.g., gc tables), reduced symbol space.

On Xerox workstations, users are reminded to reload fresh sysouts, because they eventually fill up their sysout partition. With Sun workstations, there is no such limit reminder, so users' sysouts tend to grow to the maximum size (32 MB), and thus run slower and slower.

## Console Messages

Under SunOS, various system processes and operations attempt to log information on the console. Since Medley takes over the screen, console messages are redirected (except when running under X); a background process in Medley causes them to appear in the prompt window.

However, when Medley is run remotely (i.e., not from the console), most console, or operating system, messages are printed in the prompt window. However, some messages may also appear in the middle of the Medley display screen or on the remote tty. This occurs because UNIX is often confused about where to send messages. Note that Medley is normally run remotely only for debugging purposes.

---

##### **CAUTION**

Critical UNIX system processes can hang if the buffer holding console messages fills. Medley uses a temporary file, `/tmp/XXXX-lisp.log`, where `XXXX` is the user's login name, to buffer console messages before printing them. Do not delete this log file while Medley is running. If the log file is deleted, console messages can no longer be printed in the Medley prompt window.

---

[This page intentionally left blank]

## 5. MEDLEY FILE SYSTEMS

---

This chapter discusses the conventions for using files from Medley.

### File Naming Conventions

---

In Lisp, a file name (pathname) consists of a collection of fields: the *host*, *directory*, *name*, *extension* and *version*. These fields are optional. The standard Lisp syntax for these fields is:

`{host}<directory>name.extension;version`

The *directory* field can be a directory path consisting of a sequence of directory and subdirectory components. Slashes (/) and right angle brackets (>) can be used to delimit a directory name; there is no distinction made between them. Square brackets ([ ]) are not acceptable as directory delimiters.

Duplicated directory delimiters are treated as a single delimiter. Thus, the following two file names specify the same file:

```
{DSK}<LISP>USERS>FOO.;1
{DSK}</LISP/USERS/>FOO.;1
```

### Hosts that Medley Supports

---

|                     |                                                                                          |
|---------------------|------------------------------------------------------------------------------------------|
| {CORE}              | Creates "files" in memory; useful for quick temporary files                              |
| {LPT}               | Creates files that are automatically sent to your printer                                |
| {NULL}              | Creates a file that does nothing                                                         |
| {DSK} and<br>{UNIX} | Give you access to the Sun's file systems; the rest of the chapter concentrates on them. |

The above hosts are described in more detail in the IRM.

### Using SunOS Files from Medley

---

You can access any mounted SunOS file system directly from Lisp. The mounted file system is available as an I/O device of the Lisp environment. This file system appears as the local disk of Lisp, even though it may be a remotely mounted file system of networked Sun file servers.

Many of the file devices to which the Medley environment can talk, including PUP, XNS file servers, the {CORE} device, and others, have facilities that are not directly supported by SunOS. For example, many file systems have file version numbers and case insensitive file search conventions.

Medley on the Sun Workstation has two distinct "host" names that can be used to access the SunOS file system. These host names are provided for compatibility with existing applications and tools. They also simultaneously allow natural interaction with the SunOS file system. The names are:

- {DSK}** On the Xerox workstation, {DSK} gave you access to your local hard disk; to use {DSK}, you had to create a directory on each disk partition you wanted to use. On the Sun Workstation, in contrast, the {DSK} device lets you access the file system using similar conventions to those used for {DSK} on the Xerox workstation local disk devices. In particular, {DSK} files have version numbers; {DSK} file name recognition also ignores the case of letters.
- {UNIX}** The {UNIX} device lets you use the mounted file systems with the normal naming conventions of the SunOS file system. {UNIX} files do not have version numbers, and the file name recognition treats lowercase letters as distinct from their uppercase equivalents.

File streams can be opened or closed on both devices. The reason for having both devices is to more easily support the running of applications that were originally developed on a Xerox workstation, while still allowing new applications to interact more naturally with UNIX.

NOTE: Both {DSK} and {UNIX} work as filters. They act as pointers to a device. On 11xs, {DSK}foo is the same as {DSK}<lispfiles>foo. On the Sun, {DSK}foo is the same as \$HOME/foo ~user/foo.

### Common {DSK} and {UNIX} Naming Conventions

- To include a special character (e.g., > or ;) in a file name, precede it with a single quote ('). To include a single quote in a file name, precede it with another single quote. You can quote any of these characters: <, >, ;, ~, and a period (.). The following examples show how the single quote notation on {DSK} and {UNIX} is used.

#### **{DSK} Name From Lisp**

```
foo'>bar.baz;1
foo';bar.baz;1
foo''bar.baz;1
```

#### **File Name From SunOS**

```
foo>bar.baz
foo;bar.baz
foo'bar.baz
```

- {DSK} and {UNIX} do not allow you to use either the slash (/) or the NUL character in file names. Thus, you cannot name files containing these characters.
- Both {DSK} and {UNIX} can handle the following characters, which were defined as special characters in Medley Release 1.1: backslash (\) and tilde (~).
- {DSK} and {UNIX} can distinguish between a file name with a period at the end (e.g., foo.) and a simple file name (e.g., foo). The final period is preceded with a single quote, as shown in the following example:

#### **{DSK} Name From Lisp**

```
foo.;1
foo'..;1
```

#### **File Name From SunOS**

```
foo
foo
```

- On {DSK} and {UNIX}, the C-Shell and SunOS directory notations (~, ., and ..) are supported in the Lisp directory specification. The tilde character (~) is allowed at the very beginning of the directory specification of a pathname. A combination of relative path specifiers (~, ., ..) is supported. The tilde character corresponds to the

user's home directory at login. The period (.) corresponds to the current working directory. Two periods (..) indicates the parent of the current working directory.

- File names are returned by the system (e.g., `INFILEP`) in more canonical form. The function which returns the full file name returns it in the canonical form, as in `{DSK}<usr>etc` rather than `{DSK}/usr/etc/`. This change will make some tools which depend on the conventional file name representation described in the *Interlisp-D Reference Manual* work correctly on the Medley file system (e.g., `COPYFILES`).

## **{DSK} Naming Conventions**

The `{DSK}` device performs the following file name transformation when actually accessing the SunOS file system:

- Mixed case letters are read as such.
- File name searches are done case-sensitive first; if a match is not found, the system does a case-insensitive search.
- The left angle bracket character (<) is translated to a slash (/), the delimiter for the root directory.
- `{DSK}` supports relative pathnames. You can specify relative pathnames by omitting a slash (/) or left angle bracket (<) as the first character in the directory field. For example:

`{DSK}foo.fee` and `{DSK}~/foo.fee` are relative to the user's UNIX home directory (`~/foo.fee`).

`{DSK}./foo.fee` is relative to the user's current working directory (`SunOS./foo.fee`).

`{DSK}../foo.fee` is relative to the parent directory of the user's current UNIX working directory (`../foo.fee`).

The Medley 2.0-S `{DSK}` device supports the notation in which the three meta characters (`'`, `..`, and `~`) are used together, as shown in the following example:

`{DSK}~/../tom/foo.c`

In this example, the `{DSK}` device interprets `tom` as one of the subdirectories of the parent directory of the user's home directory.

`{DSK}` also supports the tilde-name (`~name`) convention. `{DSK}` interprets `{DSK}~tom/foo.c` as a file named `foo` on `tom`'s home directory. In this notation, the user name is case-sensitive (e.g., `~tom` and `~Tom` are treated as different users).

## **Version Numbering**

The UNIX file system does not support version numbers in file names; `{DSK}` emulates versions with a naming convention. (GNU Emacs also uses this convention.) This section explains how `{DSK}` version numbers are represented in the SunOS file system.

- When you create a completely new file, it appears in the SunOS file system without a version number.

**{DSK} Name From Lisp**

**File Name From SunOS**

`bar.baz;1``bar.baz`

- When you create (from Medley) a file with a version other than 1, Medley adds version numbers to that file name, as a trailing number between tildes, e.g., “myfile.~12~” for the twelfth version of myfile.

The following shows some examples of equivalent file names in Lisp and SunOS.

**{DSK} Name From Lisp****File Name From SunOS**`bar.baz;1``bar.baz.~1~``bar.baz;2``bar.baz.~2~``bar.;23``bar.~23~`

- Medley always maintains a versionless file which is hard-linked to the highest extant version of the file (i.e., they are two names for the very same file). This file name does not appear in the {DSK} directory listing.

**From {DSK}****From SunOS**`foo.c;15``foo.c (hard linked with foo.c.~23~)``foo.c;23``foo.c.~15~``foo.c.~23~`

Similarly, a file created in UNIX with no version number is treated by {DSK} as the highest version.

- When you create a new version of a file, the versionless–file link is broken, and the versionless file is hard–linked to the new highest version.

**From {DSK}****From SunOS**`foo.c;15``foo.c (hard linked with foo.c.~24~)``foo.c;22``foo.c.~15~``foo.c;24 (new file)``foo.c.~22~ (no link with foo.c)``foo.c.~24~ (new file, link from foo.c)`

- When you delete the highest version of a file, the versionless file is also deleted. If any older versions of the file remain, a new link is created from the versionless name to the highest version extant. For example, if you have the files

**From {DSK}****From SunOS**`foo.c;1``foo.c (linked to foo.c.~2~)``foo.c;2``foo.c.~1~``foo.c.~2~`

and you delete `foo.c;2` from {DSK}, the resulting files are:

**From {DSK}****From SunOS**`foo.c;1``foo.c (linked to foo.c.~1~)``foo.c.~1~`

- When you rename a file, it works the same as deleting the file under the old name then creating it under the new name. For example, if you have the following {DSK} files

**From {DSK}****From SunOS**`foo.c;1``foo.c (linked to foo.c.~2~)``foo.c;2``foo.c.~1~``fee.c;1``foo.c.~2~`

fee.c;2

fee.c (linked to fee.c.~2~)

fee.c.~1~

fee.c.~2~



and you rename "foo.c" to "fee.c", your renamed {DSK} files and the linked SunOS files would appear as:

**From {DSK}**

foo.c;1  
fee.c;1  
fee.c;2  
fee.c;3

**From SunOS**

foo.c (linked to foo.c.~1~)  
foo.c.~1~  
fee.c (linked to fee.c.~3~)  
fee.c.~1~  
fee.c.~2~  
fee.c.~3~ (renamed file)

- When a file has a name suffix that is not a valid version number (e.g., myfile.~12x~), the suffix is regarded as part of the file name.

**From {DSK}**

myfile.~12x~;1

**From SunOS**

myfile.~12x~

## Pathnames

A pathname on {DSK} is always case insensitive. When the user specifies a file, the {DSK} device handler first searches for the file with the specified name. If no such file is found, it then searches for a file with the same spelling but different case.

Most Lisp functions, such as `FINDFILE` and `INFILEP`, which return pathnames return them with the original case when they are applied on files on {DSK} when `IL:*DSK-UPPER-CASE-FILE-NAMES*` is `NIL`. If `IL:*UPPER-CASE-FILE-NAMES*` is not `NIL`, these functions return only uppercase pathnames. The only exception is the function `DIRECTORY`, which returns a list of pathnames. The case of the pathnames is controlled by the global variable `IL:*UPPER-CASE-FILE-NAMES*` in a similar manner to `IL:*DSK-UPPER-CASE-FILE-NAMES*`.

If a pathname on {DSK} has no directory specification, a tilde-slash combination (`~/`) is used, i.e., the Lisp directory specification {DSK}foo is the equivalent of {UNIX}~/foo.

## {UNIX} Naming Conventions

For the {UNIX} device, file name translation takes place only on the directory. An initial left angle bracket (`<`) is treated as if it were an initial slash (`/`); both signify a path relative to the SunOS file system root directory; if there is no initial left angle bracket or slash, the directory is relative to the current working directory. Initially this is the working directory where Lisp was started; you can change it using the `CHDIR` function, described below. Tilde (`~`) is translated to the user's home directory.

For example, {UNIX}myfile/abc means the file abc on the ./myfile directory.

The {UNIX} device does not recognize version numbers, does not return them, and ignores them for recognition.

No case translation or recognition is done; upper- and lowercase letters are treated as distinct.

Examples:

| <b>{UNIX} Name From Lisp</b> | <b>File Name From SunOS</b> |
|------------------------------|-----------------------------|
| <foo>fee>bar.baz;1           | /foo/fee/bar.baz;1          |
| <foo>fee/bar.;1              | /foo/fee/bar.;1             |
| <foo/fee>                    | /foo/fee/                   |
| </foo/fee/>                  | /foo/fee/                   |
| /foo/fee/bar.~1~             | /foo/fee/bar.~1~            |
| /foo/fee/                    | /foo/fee/                   |

In the first two examples the ;1 is treated as part of the file name, not the version number. In the last two examples that translation is not done.

## Directories

In places where Lisp expects a directory name, {UNIX} paths must end with a slash (/).

### Directory Enumeration

You cannot use the wildcard character, asterisk (\*), in subdirectories for either {DSK} or {UNIX} devices. For example

```
> (DIRECTORY ' {DSK}/users/x*/foo)
NIL
```

Enumeration of files in directories differs between {DSK} and {UNIX} devices. On the {DSK} device, a versionless file which has a link to the highest version file is not enumerated in a directory.

On the {UNIX} device, all files are enumerated in a directory. For instance, if the following SunOS files linked with foo.c.~2~ exist

```
foo.c
foo.c.~1~
foo.c.~2~
```

the {DSK} directory enumeration would look like this:

```
> (DIRECTORY ' {DSK}/users/venue/*)
({DSK}/users/venue/foo.c;1
 {DSK}/users/venue/foo.c;2)
```

The {UNIX} directory enumeration, on the other hand, would look like this:

```
> (DIRECTORY ' {UNIX}/users/venue/*)
({UNIX}/users/venue/foo.c
 {UNIX}/users/venue/foo.c.~1~
 {UNIX}/users/venue/foo.c.~2~)
```

### Directory Creation

{DSK} When you write a new file on {DSK}, if the directory named in a pathname does not exist, the {DSK} device handler creates the directory automatically. This feature is provided for compatibility with other Interlisp-D implementations.

If you try to "connect" to a nonexistent directory (using the `CONN Exec` command or the function `IL:CNDIR`), Medley returns the message

Nonexistent directory

{UNIX}      The {UNIX} device does not support such directory creation. An attempt to create a file on a nonexistent directory results in an error.

{UNIX}/users/venue/foo.c.~2~)

## Directory Deletion

Neither {UNIX} nor {DSK} support automatic directory deletion. To delete a directory you must use the SunOS C-Shell command `rmdir`.

## Open File Limit

The number of simultaneously open {DSK} and {UNIX} files must fall within the SunOS limits for a process. For OS 3.4, this number of open files may be configured, with 30 as the maximum permissible number of open files per process. This means that it is not possible to have more than 30 files open for a process, minus whatever files Medley has open for its own use, at any one time in the Medley system. If you try to open too many files, the system call error number 24, Too many open files, appears in the prompt window.

For OS 4.0, the maximum number of files/processes that can be open at one time is 64, unless your kernel is configured otherwise.

## Default Pathname

If no path is given, the {DSK} device defaults to the user's home directory, tilde-slash (~/). The {UNIX} device defaults to the current working directory. This current working directory can be changed with the `CHDIR` function. The current working directory is also used to resolve the interpretation of the period (.) and double period (..) specifications at the beginning of a {DSK} pathname.

**(IL:CHDIR *PATHNAME*)**

**[Function]**

Changes the current working directory for the current invocation of Lisp. For example,

```
(CHDIR "{DSK}~/subdir/")
(OPENSTREAM "{DSK}./foo" ...)
```

opens the SunOS file `~/subdir/foo`.

When *PATHNAME* does not end with a slash (/) or right angle bracket (>), the whole *PATHNAME* is treated as a directory name:

```
(CHDIR '{DSK}<users>local>')
> "{DSK}<users>local>"
(CHDIR '{UNIX} /usr/local')
> "{UNIX}<usr>local>"
```

If *PATHNAME* is `NIL`, `CHDIR` tries to change the current working directory to the current connected directory. If the directory is connected to devices other than {DSK} or {UNIX}, the error message

Bad Host Name

appears, followed by the host name of the current connected directory.

If the *PATHNAME* does not exist, the error message  
No-Such-Directory  
appears followed by the system echo of the pathname.

## File Attributes

---

This section describes how the various file attributes are treated by Lisp on the Sun Workstation and what they translate to in SunOS.

GETFILEINFO obtains file attributes and SETFILEINFO sets the attributes.

---

### WRITEDATE and CREATIONDATE

[File Attributes]

Resets the date to the current time whenever the contents of a file are modified. This only works for the owner of the file. Since UNIX does not naturally support more than one date for file modification, the WRITEDATE and CREATIONDATE are treated identically by Lisp functions OPENSTREAM, OPENFILE, GETFILEINFO, and by the {DSK} and {UNIX} devices.

---

### TYPE

[File Attribute]

Sets the TYPE property of files; normally either TEXT or BINARY. However, UNIX does not distinguish between TEXT and BINARY files. Normally, programs will infer the type by the file extension, using the Lisp variables DEFAULTFILETYPE and DEFAULTFILETYPELIST. This is the convention used by Medley. If no file extension is given, the value in DEFAULTFILETYPE is used. SETFILEINFO cannot change the TYPE attribute.

---

### EOL

[File Attribute]

Returns the end-of-line convention. Both the {DSK} device and {UNIX} use line feed (LF) as the default EOL convention for text. The EOL for binary files is carriage return (CR). EOL uses the TYPE property of files. (The TYPE property of a file depends on the file extension and the DEFAULTFILETYPE and DEFAULTFILETYPELIST variables). If the TYPE property of a file is TEXT, LF (=10) is used as EOL. If the TYPE property of a file is BINARY, CR (=13) is used as EOL.

NOTE: EOL conventions on {DSK} are not compatible with those on Xerox workstations.

---

### AUTHOR

[File Attribute]

Returns the author of the file, i.e., the login name of the user who created it. This attribute cannot be changed.

---

### PROTECTION

[File Attribute]

Returns file protection attributes. The file protection attributes of files under the SunOS cannot be directly manipulated from inside Lisp. It is necessary to use the UNIX chmod command to change file protection bits.

**SIZE****[File Attribute]**

Returns the file size. For compatibility with other Lisp environments running on Xerox workstations, the SIZE attribute is computed as the length of the file (in bytes) divided by 512 (rounded up).

NOTE: SETFILEINFO lets you change the SIZE attribute of I/O streams and output streams. However, a file cannot be expanded this way.

---

## File Variables

---

This section discusses how certain file variables are used by Medley in SunOS.

**IL:FileTypeConfirmFlg****[Variable]**

The file-type attribute of a file on {DSK} or {UNIX} is decided from its extension, DEFAULTFILETYPEELIST and DEFAULTFILETYPE. Extensions of binary files should be registered in DEFAULTFILETYPEELIST. When this rule is broken, a hardcopy of files on {DSK} and {UNIX} may confuse the printers. So when you try to hardcopy a file whose extension is not registered in DEFAULTFILETYPEELIST, a menu is invoked to confirm the file type. Text or binary can be selected. The invocation of this menu can be stopped by setting IL:FileTypeConfirmFlg to NIL. The default value of IL:FileTypeConfirmFlg is T.

When extensions of binary files are not registered in DEFAULTFILETYPEELIST, copy or rename from a DSK/UNIX device to a non-DSK/UNIX device also may cause file type confusion. This type of copy or rename results in one of the following warning messages in the prompt window, as appropriate:

**Extension of {DSK}foo.fee;1 isn't in DEFAULTFILETYPEELIST. {CORE}foo.fee;1 was copied as TEXT.**

**This message can be stopped by set FileTypeConfirmFlg to NIL.**

**Extension of {DSK}foo.fee;1 isn't in DEFAULTFILETYPEELIST. {CORE}foo.fee;1 was renamed as TEXT.**

**This message can be stopped by set FileTypeConfirmFlg to NIL.**

Either of these messages can be stopped by setting IL:FileTypeConfirmFlg to NIL.

**IL:DEFAULTFILETYPE****[Variable]**

Initially set to TEXT. Used with the file attribute TYPE.

**DEFAULTFILETYPEELIST****[Variable]**

A list of accepted file types. Initially set to ( (NIL . TEXT) (C . TEXT) (H . TEXT) (LISP . TEXT) (LSP . TEXT) (O . BINARY) (OUT . BINARY) (LCOM . BINARY) (DFASL . BINARY) (DCOM . BINARY) (SKETCH . BINARY) (TEDIT . BINARY) (DISPLAYFONT . BINARY) (WD . BINARY) (IP . BINARY) (RST . BINARY) (BIN . BINARY) (MAIL . BINARY) (SYSOUT . BINARY) )

Used with the file attribute TYPE. Binary files, such as Sketch files, InterPress files, or Press files, should have their extensions registered in DEFAULTFILETYPEELIST. This is especially important because UNIX does not support file types.

## File System Errors

Several types of errors may occur in the Medley file system.

When a remotely mounted file system or NFS service is down, or when network traffic is heavy, any attempt to access a file on that file system results in an error. The following error message is printed in the prompt window:

```
File access timed out
```

Medley will wait until the file system responds or until a timeout occurs. If the file system is mounted with the "hard" option, the timeout is controlled by the value of environment variable `LDEFILETIMEOUT`. If the file system is mounted with the "soft" option, the timeout depends on the NFS file system timeout time, and the value of `LDEFILETIMEOUT`. Medley will wait until the shorter of these two times is exceeded. The NFS file system timeout time, retry times, etc., are controlled by the UNIX command **mount**.

If `LDEFILETIMEOUT` is not set, the default value of 10 seconds is used. The variable is inspected at boot time, and a setting between 1 and 100 seconds is appropriate in most cases.

The following error messages may appear when there are Medley file errors:

```
Not owner
Device error:
Protection-violation
File-won't-open
Too-Many-Files-Open
File too large
File-System-Resources-Exceeded
Connection timed out
No-Such-Directory
Bad Host Name
FS-RENAMEFILE-SOURCE-COULDNT-DELETE
```

Another type of error occurs when the user has insufficient access to files. When this happens, Medley will print the following message:

```
File not found
```

The following message then appears in the SunOS prompt window:

```
System call error: open errno=13 Permission denied
```

See the *UNIX Interface Reference Manual*, Intro (2), for descriptions of all OS system call messages.

[This page intentionally left blank]

## 6. ERROR RECOVERY

---

Medley on the Sun Workstation has an error handling system which includes the following:

- The Xerox Lisp error system, described in the IRM
- A diagnostic program, URAID, which handles emulator errors

Occasionally, you may encounter SunOS error messages. Refer to your Sun documentation set for recovery procedures when these errors occur. When running Medley on a Sun Workstation, previous Lisp error handling such as Teleraid and MP errors are no longer available. However, you can still use Teleraid from a Sun Workstation to debug a Xerox 1100 series workstation.

### URAIID

---

The Medley system normally operates as a self-contained environment. In some unusual circumstances Medley may encounter a situation from which it cannot recover. In this case, when an unrecoverable emulator error is encountered, the emulator halts and enters into a small debugger called URAID. URAID allows you to inspect memory, or to look inside the sysout file, and attempt to recover from the error.

If you produce the same type of error condition in Medley on a Sun Workstation as you did on a Xerox workstation, you get a URAID error instead of an MP error.

#### Entering URAID

Normally, the emulator automatically enters URAID when an unrecoverable emulator error occurs. However, there are two additional methods available when you want to enter URAID directly.

- Use the SHIFT-CTRL-DELETE key combinations to enter URAID between opcodes. Note that the DELETE key referred to here is in the L10 position on the left keypad of the Sun keyboards. This sequence allows you to return to Lisp later.
- Use SHIFT-CTRL-NEXT for emergency interrupts only. Note that the NEXT key is in the ALTERNATE key position on the regular Sun keyboard. These combinations are useful for exiting from an opcode infinite loop. SHIFT-CTRL-NEXT does not necessarily enter URAID between opcodes; once you are in URAID mode, another URAID command such as "f" could cause the emulator to crash. At this point it is unlikely that you could return to Lisp. USE WITH CAUTION!

#### Conventions

URAIID uses these display conventions:

- Numbers are displayed in hexadecimal unless otherwise noted.
- The *litatom* should be an uppercase string when used with a package prefix (e.g., XCL:EVAL).
- Symbols are displayed with a package prefix, but with no escape character.



- Symbols in the Interlisp (IL:) package are case-sensitive (e.g., IL:\InterfacePage); symbols in other packages are case-insensitive (e.g., XCL:EVAL).

In addition, these input conventions apply:

- Symbols may only be qualified by their home package.
- A full package name may prefix an input symbol. URAID also supports approved abbreviations of package names (e.g., XCL:, SI:, CL:, XCLC:).

A symbol without a prefix is treated as a symbol in the Interlisp package. For instance, \InterfacePage is the equivalent of IL:\InterfacePage.

- Type-in is uppercase for symbols in any package except the Interlisp package; type-in is in mixed case for IL: package symbols or symbols with no prefix.

## URAID Commands

URAID has a few simple commands which you can use to attempt diagnosis and error recovery. All URAID commands are case-sensitive.

- h** Hard Reset. Attempts to recover by resetting the Lisp stack. Quits URAID and causes Lisp to resume execution. This command should not be used unless you are sure that execution can be resumed.
- e** Exits to SunOS. Medley will end.
- q** Quits URAID and returns to Lisp.

NOTE: An error may occur while the Medley system is running uninterruptably. The following message signals this error:

```
Error in uninterruptable system code -- ^N to continue into
error handler
```

Disregard the ^N command; it is not supported by URAID. Use the **q** command to continue.

## Displaying a Stack

For casual users, the **l** command followed by several **f** commands generally provides the most useful information. Many of the other commands require some knowledge of the internal representation of Lisp objects and stack frames.

- c** Checks all user stack contents; stack inconsistency is displayed.
- k type** Changes the stack link that precedes the **l** command to be *type*, which is either **a** (to follow ALinks) or **c** (to follow CLinks). The default is to trace ALinks. ALinks follow the chain of free variable access.
- l type** Shows the stack as a back trace consisting of a numbered sequence of frame names. The default is the user stack. The argument *type* is a single letter denoting the stack to view. The system has a number of special contexts, which are areas of stack space used by certain system routines. Legal values of *type* are as follows:
  - g** (garbage collect)
  - k** (keyboard handler)
  - m** (miscellaneous)
  - p** (page fault)
  - r** (reset)
  - u** (user stack) - Default

*type* := g|k|m|p|r|u or nil

- C** Checks the contents by scanning all stack space in the sysout. For example:

```
0x11880 BF,[ivar:0x1800]
0x11802: FX for CL:T[]
0x11816 BF,[ivar:0x1816]
0x11818: FX for IL:\TURN.ON.PROCESSES[]
```

## Viewing Frames From a Stack

After displaying a particular stack with the **I** command, the following commands view individual frames from that stack:

- f number** Displays the contents of frame *number* (decimal) with its basic frame, IVars and PVars. The frame is printed in two parts, a basic frame containing the function's arguments and a frame extension containing control information, the function's local (PROG) variables, and dynamic values. On the left side of the printout are the hexadecimal contents of each cell of the frame, with an interpretation, usually as a Lisp value, on the right. The following message appears as you display a frame with the **f** command:
- Press Return (To quit ESC and RET)
- To abort the printing of a frame, first press the ESC key then the RETURN key. The URAID prompt "<" reappears.
- <CR>** Displays the next frame (closer to the root, or bottom, of the stack). This is the same as **f n+1**, where *n* is the number of the frame most recently viewed. Immediately after an **I** command, *n* is zero, so **<CR>** views the first frame.
- a litatom** Displays the top-level value of the *litatom*
- d litatom** Displays the contents of definition cell for the *litatom*. If it is compiled code, this command prints a CCODEP hexadecimal address pointer; for example,
- ```
{CCODEP}0x14ccc4
```
- Otherwise, it prints a Lisp definition; for instance, interpreted code returns
- ```
(LAMBDA () ...)
```
- M** Displays TOS, CSP, PVar, IVar, PC.
- m func1 func2** Moves the definition of *func1* to *func2*.
- t Xaddress** Displays the type of this object.
- p litatom** Displays the contents of the *litatom*'s property list.
- w** Displays the current function name and PC.
- x Xaddress[Xnum]** Prints *Xnum* word (16-bits) of the raw contents of the virtual memory starting at virtual address *Xaddress*. This

is most useful for examining the contents of a datatype which other commands simply print as its virtual address.

**@litatom**[*snumber* | *NIL* | *T*] Sets the TOPVAL of *litatom* to the specified value. *snumber* is a signed smallp number.

**<Xaddress val** Sets the the contents of the word (16-bits) at the *Xaddress* to *val*.

## Miscellaneous

**v filename** Saves the current virtual memory on the *filename*. This file can be examined using the functions READSYS and VRAID in the TeleRaid Lisp Library module, but cannot be used as a sysout file.

NOTE: This sysout cannot be restarted.

**s** Invokes a subshell.

**(num** Sets the print level (default is 2).

**?** Displays this summary.

**!** Prints the error message passed from the emulator.

---

## Other Fatal Error Conditions

Occasionally, other emulator, operating system, or system administration errors may occur from which the URAID program cannot recover. Such error conditions include the process dying, the emulator going into an infinite loop, the keyboard being lost, or the system freezing up.

If any of these emulator errors occur, use the UNIX `kill` command to kill the `lde` process.

---

## Lisp Errors

### Errors While Running Medley

The following Lisp errors may occur when running Medley on the Sun Workstation.

| <u>ERROR MESSAGE</u>  | <u>CAUSE</u>                                                                                                                                                                             |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| File access timed out | Occurs when you try to access a file when the remotely mounted file system or NFS service is down, or when network traffic is heavy. See the File System Errors subsection of Chapter 5. |
| File too large        | Self-explanatory.                                                                                                                                                                        |
| Too-Many-Files-Open   | Occurs when you exceed one of the following: <ul style="list-style-type: none"><li>• SunOS open file limit (see Chapter 5, Medley File Systems)</li></ul>                                |

|                                     |                                                                                                                  |
|-------------------------------------|------------------------------------------------------------------------------------------------------------------|
|                                     | <ul style="list-style-type: none"><li>• System file resources while writing a sysout (using IL:SYSOUT)</li></ul> |
| Nonexistent directory               | Occurs when you try to connect to a nonexistent directory using the IL:CNDIR function or the CONN command.       |
| No-Such-Directory                   | CHDIR                                                                                                            |
| Connection timed out                | Self-explanatory.                                                                                                |
| Bad Host Name                       | Self-explanatory.                                                                                                |
| FS-RENAMEFILE-SOURCE-COULDNT-DELETE | Occurs when you try to rename a file which exists on a directory or which you do not have delete permission.     |

### Xerox Workstation-Specific Errors

These Xerox workstation-specific errors may occur if certain functions are inadvertently used on the Sun Workstation.

| <u>ERROR MESSAGE</u>                     | <u>CAUSE</u>                                                                                                    |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Floppy: No floppy drive on this machine. | Self-explanatory.                                                                                               |
| Device error: {FLOPPY}                   | Occurs when the user tries to enter a Lisp floppy function while running on the Sun Workstation.                |
| Wrong machinetype                        | Occurs when functions controlling Xerox disk drive device-specific behavior are entered while running in SunOS. |

### Virtual Memory Errors

| <u>ERROR MESSAGE</u>           | <u>LISP FUNCTION RESPONSIBLE</u> |
|--------------------------------|----------------------------------|
| File-System-Resources-Exceeded | IL:SYSOUT, IL:LOGOUT, IL:SAVEVM  |
| Protection-Violation           | IL:SYSOUT, IL:LOGOUT, IL:SAVEVM  |
| File-Wont-Open                 | IL:SYSOUT, IL:LOGOUT, IL:SAVEVM  |

[This page intentionally left blank]

## APPENDIX A. INSTALLATION HINTS

---

### Medley Shell Variables

---

The following is a fragment of a `.cshrc` file which you may want to adapt to your own needs. In this example Smythe works in Building 12b (bldg12b), and always wants a fresh sysout, containing Rooms, loaded.

```
=====
Set up various Medley variables.
setenv LDEDESTSYSOUT /user/smythe/sysouts/saved.virtualmem
setenv LDESRCESYSOUT /usr/share/lde/lispsysouts/ROOMS.SYSOUT
setenv LDEINIT /usr/share/lde/site-files/bldg12b-init.lcom

Assuming you are using UNIXChat and VTChat,
configure the Chat window
if ($?LDESHELL == 1) then
 setenv TERM vt100
 stty erase ^H
endif

=====
```

### Running on Multiple Workstations

---

#### Installation for Sites with Sun-3 and Sun-4 Workstations

In Medley 2.0, the only differences between the Sun-3 and Sun-4 distributions are in the `install.sunosX` directories. Thus, during installation the common subdirectories (`lispsysouts`, `lisplibrary`, `fonts`, etc.) might be installed instead to a shared file system, saving 15 MB of unnecessary duplicated space. In the example below, `/sharedserver` is a remote file system mounted on the local machine.

```
prompt% mkdir /sharedserver/lde
prompt% cd /sharedserver/lde
prompt% tar xvfb /dev/rxx0 126 ./lispsysouts ./lisplibrary
 ./fonts
```

If soft links are then left on `/usr/share/lde`, the installation can proceed as before.

```
prompt% ln -s /sharedserver/lde/lispsysouts
 /usr/share/lde/lispsysouts
prompt% ln -s /sharedserver/lde/lisplibrary
 /usr/share/lde/lisplibrary
prompt% ln -s /sharedserver/lde/fonts /usr/share/lde/fonts
```

Otherwise, the site initialization file needs to be changed appropriately.

The install directories are left on /usr/share/lde, since those directories are typically local to a particular processor architecture.

```
prompt% cd /usr/share/lde
prompt% tar xvfb /dev/rxx0 126 ./install.sunos4
```

## Using a "runlde" on Multiple Workstations

The following is an example of a runlde script that might be used for running Medley on different machines.

```
(invokes CSH)
=====
Usage: runlde optional-sysout
#
The script below is for the following machines:
#
Host HostID
---- -
timber 1700319b
gopher 17003016
tree 13003565
=====

switch ("hostid")
 case '1700319b':
 ldeether $1 -k '99e8bfc6 92299f45 9199a409'
 breaksw
 case '17003016':
 ldeether $1 -k '70c5a8d8 7b0498cc 45e35500'
 breaksw
 case '13003565':
 ldeether $1 -k 'ce7627bf b5b61ac8 2f990cc0'
 breaksw
 default:
 echo "Sorry, host 'hostname' is not in this shell
 script"
endsw
```

---

## Configuring the Software

The software comes in these two forms:

- An executable binary image for users who have not modified the Sun kernel too extensively
- An object file that can be relinked for your particular system.



If you want to use the executable that Venue supplies, skip to the `Enabling PUP/XNS Ethernet` subsection below.

## Relinking

If you have tried the prelinked software and it doesn't work, link the object code with the Sun libraries. To do this, you need the `suntool`, `sunwindow`, and `pixrect` libraries, and `make`, `cc`, etc., available on your search path. To configure the system, connect (`cd`) to the directory `usr/share/lde/install.sunosx` (where `x` is the version of SunOS that you are running, e.g., SunOS 4.0 in the following), and type `make`.

```
prompt% cd /usr/share/lde/install.sunos4
prompt% rm lde ldeether; make
```

This procedure replaces the two executable programs, `lde` and `ldeether`. The program `ldeether` enables access to Xerox network protocols from Lisp.

## Enabling PUP/XNS Ethernet

If you intend to use the PUP or XNS Ethernet directly from Medley, you need to change file ownership and permissions of `ldeether`. Note that you do this on the server where `ldeether` is actually residing (root permission must be on the server). Log in to the machine where `ldeether` resides. To find out where `ldeether` resides, type:

```
prompt% df filename
```

where *filename* is the pathname of `ldeether`. The system responds with the name of a file system (e.g., `/dev/sd0g`) for a local file, or with a machine name and directory (e.g., `python:/user1`) for an NFS file.

Now you can change the `ldeether` file ownership and permissions.

```
prompt% rlogin server
server% su
server# cd /usr/share/lde/install.sunos4
server# /etc/chown root ldeether
server# chmod 4755 ldeether
server# exit
```

If you are using the Ethernet, substitute `ldeether` whenever `lde` appears in the instructions below.

---

## Using NIS to Manage the Keys for Multiple Workstations

Here is an example how to handle several Medley licenses on a network, by using the Sun Network Information Service (NIS).

Create a file containing an association list of hostnames vs. license keys, for each host that has a Medley license. For example:

```
medley-keys.by-hostname
=====
king 6a1c33bf 11dc1a48 a4c34080
sidewinder 7b636e98 55a26cd4 26b80560
hognose 190750c0 17c658e0 08060ac0
boa 8334d182 00793e07 4903890b
asp c90faa4f d3477c53 d304b85b
rattler 70b8fd18 2d79f344 c30051c0
```

NOTE that the following commands should all be run as root.

On your NIS master server, create an NIS database of hostname vs. Medley keys:

```
prompt% /usr/etc/yp/makedbm ./medley-keys.by-hostname \
 /var/yp/your-domain/medley-keys
```

Replace *your-domain* with the name of your NIS domain. The output is put in the directory containing your master NIS maps.

If you have NIS slave servers serving your domain, you will need to update each one manually the first time the map is created. Thereafter, they will be updated automatically. On each NIS slave server do the following:

```
% /usr/etc/yp/ypxfr -f -h your-NIS-master medley-keys
```

Replace *your-NIS-master* with the name your NIS master server.

After updating all NIS slave servers, you now need to propagate the NIS map to your NIS clients. On your NIS master, type:

```
% /usr/etc/yp/yppush medley-keys
```

From now on, any changes made to the `medley-keys.by-hostname` file will only require the propagation of the map to your NIS clients. The following steps are required:

1. Create a new NIS map using the `makedbm` command as described above.
2. Propagate the changes to your NIS clients using the `yppush` command as described above.

You can now use the newly created map. Below is an example of a `runlde` script that uses the newly created NIS map.

```
#!/bin/csh -f
=====
Usage : runlde [sysout]
#
Script for running Venue Medley software.
#
=====
if ($#argv > 1) then
 echo "Usage : runlde [sysout]"
endif

set SYSOUT = "$1"
set HOSTNAME = `/bin/hostname`
set KEY = `/bin/ypmatch $HOSTNAME medley-keys`
```

```
if ! $status then
 ldeether $SYSOUT -k "$KEY"
endif
```

Consult the *Sun Network and Communications Administration* manual for more details about NIS and how to add the new map to the `/var/yp/Makefile`.

[This page intentionally left blank]

## APPENDIX B. VERIFYING THE INSTALLATION TAPE'S VALIDITY

---

If you encounter inexplicable problems shortly after you install Medley, they may be due to files being corrupted — the release tape may have been damaged, errors may have occurred while the tape was being read, etc. If you have unexplained problems, we recommend that you verify the checksums of your installed files.

The script generates checksum files named `*.check` and compares them to the released `*.sum` residing in the `/checksumdir` subdirectory.

The checksum script reports inconsistent files, the correct checksum values for the files, and an error message. The checksum of individual files can be generated with the UNIX command `sum filename`.

| <code>ldchecksum [-cg] medleydir [ dir   dirgroup ]</code> | [Command]                                                                                                                                                                                    |
|------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-c</code>                                            | Generates checksums for your installed files and compares them with correct values. This is the default action.                                                                              |
| <code>-g</code>                                            | Generates checksums for the files specified.                                                                                                                                                 |
| <code>medleydir</code>                                     | Name of the Medley installation directory. Default is <code>/usr/share/ldc</code> .                                                                                                          |
| <code>dir</code>                                           | Any specific directory residing under <code>medleydir</code> . Only relative pathnames with respect to <code>medleydir</code> are accepted.                                                  |
| <code>dirgroup</code>                                      | The directory group, either <b>all</b> (the default) or <b>lisp</b> , which includes the <code>X/install.xxxx</code> , <code>X/lisplibrary</code> and <code>X/lispysouts</code> directories. |

### Output

As it begins checking each directory, the script prints a message in the form:

```
Checking directory: /usr/share/ldc/subdir
```

Error and warning messages may be in one of two forms:

```
< E > 32711 494045XLPSTREAM.DFASL
```

indicates that file `4045XLPSTREAM.DFASL` is erroneous or does not exist in the directory. The correct checksum of `32711`, together with the size (`49 Kbytes`) of the file, are shown.

```
< W > /usr/share/ldc/fonts/display/chinese : Directory not installed
```

indicates that Chinese fonts were not installed or were removed after Medley was installed.

### Examples

```
prompt% ldchecksum /usr/share/ldc
```

All files in the installed Medley directories in `/usr/share/ldc` are checked.

```
prompt% ldechecksum /usr/share/somedir/lde lisp
```

This example checks all files in:

```
 /usr/share/somedir/lde/install.xxxx
 /usr/share/somedir/lde/lisplibrary
 /usr/share/somedir/lde/lispsysouts
```

```
prompt% cd/usr/share/lde
```

```
prompt% ldechecksum -c . fonts/display
```

This example checks only the display font directories. The period (.) is used because you are positioned under the current Medley installation directory.

[This page intentionally left blank]

# APPENDIX C. LAYOUT OF INSTALLATION TAPE FILES

## Layout of Installation Tape

Below follows the layout of the Medley Installation Tape with a description of the individual files.

|        |        |        |
|--------|--------|--------|
| FILE 1 | FILE 2 | FILE 6 |
|--------|--------|--------|

| File | Contents                                                                                           | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------|----------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1    | ./install-medley                                                                                   | The Medley installation utility                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| 2    | ./medley                                                                                           | The Medley startup script                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 3    | ./install.sunos3/ (only on the Sun3 installation tape)<br>./install.sunos4/<br>./install.sunos4.1/ | Each subdirectory contains:<br><br>lde      Used as a bootstrapper to load the right emulator, depending on the frame-buffer of your host and whether X Windows is running.<br><br>ldeether      Used when you want to use the XNS protocol from within Medley on a Sun. It will set up your system to intercept XNS and PUP packets and then immediately runs <b>lde</b> .<br><br>ldesingle      The emulator used to run Medley on a workstation with a monochrome display or one with a color frame-buffer of type cg2, cg4, or cg9.<br><br>ldemulti      The emulator used to run Medley on a workstation with a color frame-buffer of type cg3 or cg6.<br><br>ldex      The emulator used to run Medley on a workstation where an X Windows server is running.<br><br>ldesingle.o<br>ldemulti.o<br>ldex.o      These object files are used when recompiling the emulators to either include your own C subroutines or when problems arise.<br><br>makefile<br>usersubrs.c      Used when you wish to link your own C subroutines into the emulator (a non-documented feature).<br><br>ldeether.c      The source code for the <b>ldeether</b> . Its only purpose is to allow you to recompile the ethernet set-up code should you run into any problems. |



|   |                                 |                                                                                                                                             |
|---|---------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| 4 | <code>./lisplibrary</code>      | Contains all the Medley 2.0 Lisp Library files                                                                                              |
| 5 | <code>./checksumdir</code>      | Contains <code>ldechecksum</code> , <code>checksum</code> and <code>X.sum</code> checksum files (See Appendix B for a detailed explanation) |
|   | <code>./lispsysouts</code>      | Contains the <code>sysout</code> , <code>lisp.sysout</code>                                                                                 |
| 6 | <code>./fonts/display</code>    | Contains the display fonts (See Table C-1 for a detailed description of the individual font files)                                          |
|   | <code>./fonts/interpress</code> | Contains the Interpress printer fonts (See Table C-1 for a detailed description of the individual font files)                               |

## Font Directories

Table C-1 shows the organization of the font directories, as well as the descriptions and contents of the directories.

**Table C-1. Font Directories**

| Directory Name                                                    | Description                                                                          | Font Families                                                             | Font Types                                                                                                                                                                               |
|-------------------------------------------------------------------|--------------------------------------------------------------------------------------|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ./fonts/display/presentation<br>./fonts/interpress/presentation   | All presentation fonts for display and user interface applications                   | Helvetica<br>Gacha<br>Times Roman                                         | Sans serif<br>Monospace screen font in 8, 10, 12 MRR<br>Serif                                                                                                                            |
| ./fonts/display/publishing<br>./fonts/interpress/publishing       | All publishing fonts for character sets, foreign characters, and technical alphabets | Classic<br>Modern<br>Terminal                                             | Serif; in all character sets, sizes, faces<br>sans serif; in all character sets, faces, but with selected sizes<br>Monospaced, in all character sets, faces, but with selected sizes     |
| ./fonts/display/printwheel<br>./fonts/interpress/printwheel       | All printwheel fonts for word processing applications                                | BoldPS<br>LetterGothic<br>Titan                                           | Proportional serif<br>Monospaced sans serif<br>Monospaced serif                                                                                                                          |
| ./fonts/display/JIS1<br>./fonts/interpress/JIS1                   | Japanese Kanji fonts, character set 1                                                | Classic                                                                   | Point sizes 8 through 24                                                                                                                                                                 |
| ./fonts/display/JIS2<br>./fonts/interpress/JIS2                   | Japanese Kanji fonts, character set 2                                                | Classic                                                                   | Point sizes 8 through 24                                                                                                                                                                 |
| ./fonts/display/chinese<br>./fonts/interpress/chinese             | Chinese character fonts                                                              | Classic<br>Modern                                                         | Point sizes 12 and 24<br>12 point                                                                                                                                                        |
| ./fonts/display/miscellaneous<br>./fonts/interpress/miscellaneous | Miscellaneous fonts for nonstandard and rare applications                            | ClassicThin<br>Hippo<br>Logo<br>Math<br>OldEnglish<br><br>Symbol<br>Tonto | Brackets and parentheses in point sizes 16, 20, 26, and 30<br>Greek or Latin<br>Xerox logo<br>Math symbols<br>Point sizes 10 and 18<br><br>Math symbols<br>Thick monospaced 14 point MRR |
| ./fonts/press                                                     | All metric information for Press printers.                                           |                                                                           |                                                                                                                                                                                          |

## Manually Extracting Files from the Installation Tape

---

You can manually extract individual files or directories from the Medley installation tape. For example, if you want to extract the X-window emulator **ldex** for SunOS release 4.1 from the tape do the following:

```
prompt% mt -f /dev/nrst0 rewind
```

Ensures that the tape is positioned at the beginning of the tape.

```
prompt% mt -f /dev/nrst0 fsf 2
```

Positions the tape at the beginning of the third file on the tape. The **n** in the **/dev/nrst0** makes sure the tape is not rewound after the command has been completed.

```
prompt% tar xvf /dev/nrst0 ./install.sunos4.1/ldex
```

Extracts **ldex** from the Medley installation tape and puts it in your current working directory.

## APPENDIX D. DIFFERENCES BETWEEN XEROX WORKSTATIONS AND THE UNIX VERSION OF MEDLEY

---

### Local Disk and Floppy Functions

---

The functions for controlling device-specific behavior of the Xerox 1100 series workstation disk drives are not supported. These functions signal the error

Wrong machinetype

if called when running under UNIX. These functions include

IL:PURGEDSKDIRECTORY  
IL:CREATEDSKDIRECTORY  
IL:VOLUMESIZE  
IL:DISKFREEPAGES  
IL:DISKPARTITION  
IL:SCAVENGEDSKDIRECTORY  
IL:FILENAMEFROMID

The following functions for controlling the Xerox 1100 series workstation floppy disk drive also signal an error under UNIX:

IL:FLOPPY.FORMAT, IL:FLOPPY.NAME, IL:FLOPPY.TO.FILE,  
IL:FLOPPY.FROM.FILE, IL:FLOPPY.ARCHIVE, IL:FLOPPY.UNARCHIVE,  
IL:FLOPPY.MODE, IL:FLOPPY.FREE.PAGES, IL:FLOPPY.CAN.READP,  
IL:FLOPPY.CAN.WRITEP, IL:FLOPPY.WAIT.FOR.FLOPPY,  
IL:FLOPPY.SCAVENGE

These functions signal the error

Floppy: No floppy drive on this machine. Device error:  
{FLOPPY}

The following functions have no effect and always return NIL on UNIX:

IL:VOLUMES  
IL:LISPDIRECTORYP  
IL:DSKDISPLAY

### Library Modules Not Supported on the Sun

---

The following modules listed in the manual *Lisp Library Modules*, Medley Release, are not supported on the Sun Workstation running Medley.

#### TCP, TCPCHAT, etc.

Because SunOS supports TCP/IP directly, TCP packets cannot be routed to Medley. For this reason, the TCP library modules are not supported on the Sun Workstation.

## **DLRS232C, DLTTY**

The DLRS232C and DLTTY library modules are specific to the hardware devices available on the Xerox 1100 series workstations. Serial lines and other devices can be accessed from Medley either through sub-shells, or by using the {UNIX} file device, e.g., writing to {UNIX}/dev/ttya or {UNIX}/dev/ttyb.

The following library modules are normally used with equipment attached to the Xerox 1186 RS232 serial lines:

FX-80DRIVER  
4045XLPSTREAM  
KERMIT  
RS232CHAT.

## **KEYBOARDEDITOR, VIRTUALKEYBOARD**

Medley does not include versions of KEYBOARDEDITOR or VIRTUALKEYBOARD library modules that know about the Sun keyboards.

VIRTUALKEYBOARD lets you bring up keyboard images that give you access to special characters via the mouse. The keyboard itself is unaffected.

[This page intentionally left blank]

## MEMORANDUM

FROM: John Sybalsky  
DATE: September 15, 1991  
RE: Release 2.0 of Medley for the Sun Workstation

Enclosed is the software and documentation for Release 2.0 of Medley for the Sun Workstation. The package consists of the following:

- Tape containing the revised software.
- Release Notes, providing warnings and information important to the successful running of the software, followed by fixed bugs.
- *Medley for the Sun Workstation User's Guide*, encompassing release contents, instructions for installing Release 2.0, and information on using it. This *Guide* has been completely reorganized, and information about using the new installation script has been added.
- *Lisp Library Modules* revised pages, reflecting additions to the prior issue (replace the old sections with the corresponding new pages).

# GLOSSARY

---

|                              |                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>access permissions*</b>   | Determines what operations can be performed on a file.                                                                                                                                                                                                                                                                                                                                    |
| <b>alias*</b>                | <p>A user-created C-Shell command defined in terms of other commands or programs. For example, if you type (or put in your <code>.cshrc</code> file)</p> <pre>alias runlde "lde ~/sysout -k xx"</pre> <p>then when you type <code>runlde</code> to the C-Shell, it acts as if you had typed</p> <pre>lde ~/sysout -k xx</pre>                                                             |
| <b>backing store</b>         | A Xerox 1100 series workstation file, the virtual memory partition. This file stores pages as they are allocated or flushed from real memory.                                                                                                                                                                                                                                             |
| <b>byte code emulator</b>    | A byte-code instruction interpreter. Executes the Interlisp-D virtual machine instruction set compatibly with microcode for the Xerox workstations.                                                                                                                                                                                                                                       |
| <b>chmod*</b>                | A program used to change access permissions of a file.                                                                                                                                                                                                                                                                                                                                    |
| <b>chown*</b>                | A program used to change ownership of a file.                                                                                                                                                                                                                                                                                                                                             |
| <b>{DSK}</b>                 | A host device name allowing users to access the SunOS file system. Uses conventions (e.g., version numbers and file name recognition which ignores the case of letters) similar to those used by the Xerox 1100 series workstation local disk device ({DSK}).                                                                                                                             |
| <b>environment variable*</b> | A name/value pair that is passed to subprocesses. Can be set from the shell with the <code>setenv</code> command. By convention, environment variable names use uppercase rather than lowercase letters, e.g., <code>LDEDESTSYSOUT</code> . The Medley environment variables are <code>LDESRCESYSOUT</code> , <code>LDEDESTSYSOUT</code> , <code>LDEINIT</code> , <code>LDESHELL</code> . |
| <b>home directory*</b>       | The working directory when a user logs in.                                                                                                                                                                                                                                                                                                                                                |
| <b>host access key</b>       | A special code which must be entered to Medley to run Medley software on the Sun Workstation.                                                                                                                                                                                                                                                                                             |
| <b>lde</b>                   | Lisp development environment.                                                                                                                                                                                                                                                                                                                                                             |
| <b>ldeether</b>              | A program produced during the software startup procedure; runs <code>lde</code> after enabling access to Xerox network protocol.                                                                                                                                                                                                                                                          |
| <b>.login*</b>               | The name of a file in the home directory that is read by the shell when a user first logs in. Contains C-Shell commands.                                                                                                                                                                                                                                                                  |
| <b>Medley</b>                | The Venue programming environment; also, the name of the release. Supports Common Lisp and Interlisp; a library of utilities, graphics packages, applications; a complete windowing system; network protocols. Runs on both Xerox and Sun workstations.                                                                                                                                   |



**NFS\*** Network File System; the way SunOS handles remote file systems.

---

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>pathnames*</b>               | <p>In UNIX, a position identifier of a file or directory within the file system tree structure.</p> <p>An <i>absolute</i> pathname gives the position, beginning with the root directory, of the file or directory in the file system hierarchy. Each directory in the pathname is delimited by a slash (/).</p> <p>A <i>relative</i> pathname locates the position of the desired file or directory from the working directory. Again, all directories in this pathname are delimited by the slash (/).</p> |
| <b>root directory*</b>          | <p>The root of the directory tree. Designated by a slash (/) at the <i>beginning</i> of an absolute pathname. Slashes elsewhere in a pathname are simply delimiters.</p>                                                                                                                                                                                                                                                                                                                                     |
| <b>shell*</b>                   | <p>Command interpreter (akin to the Medley Exec).</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>shell script*</b>            | <p>A file that contains shell commands. Can be run by typing the file name provided the user has execute permission on the file.</p>                                                                                                                                                                                                                                                                                                                                                                         |
| <b>site initialization file</b> | <p>A Lisp file, used when Medley is started up. Contains standardized information about the site environment such as pointers to fonts and site parameters.</p>                                                                                                                                                                                                                                                                                                                                              |
| <b>SunOS</b>                    | <p>Sun's version of UNIX.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>suntools</b>                 | <p>A Sun system window-based program tool. A program that allows all of the Sun window-based tools to run on the screen.</p>                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>tar</b>                      | <p>A program for copying data to and from magnetic tape.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>{UNIX}</b>                   | <p>A host device name allowing users to access the SunOS file system using UNIX naming conventions. Files on the {UNIX} device have no version numbers and file name recognition distinguishes between upper- and lowercase letters.</p>                                                                                                                                                                                                                                                                     |

**\* Indicates a UNIX term. See UNIX documentation for full definition.**

[This page intentionally left blank]

# INDEX

## A

Access key 11,13  
Asterisk 36  
**AUTHOR** (*File Attribute*) 38

## B

Back trace 40,42  
**BACKGROUNDPAGEFREQ** (*Variable*) 25  
**BEEPOFF** (*Function*) 27  
**BEEPON** (*Function*) 27  
**\BeginDST** (*Variable*) 12,22  
Binary files 4,38,40  
Binary image, executable A-2; 13  
Brackets  
    left angle 33  
    right angle 31  
    square 31

## C

C-Shell 32  
Carriage return 4,38,40  
Case sensitivity 33,35,39,40,41,42  
**CHANGEBACKGROUND** (*Function*) 27  
Characters, special 32  
**CHDIR** (*Function*) 35,37,44,45  
checksum 1,3, B-1  
**chmod** (*UNIX Command*) 38  
**CLOCK** (*Function*) 28  
Clocks 27  
**CNDIR** (*Function*) 37  
Compatibility  
    compiled-file 4  
    end-of-line convention 4  
    sysout 4  
Configuration  
    changing 11  
    software A-2  
**CONN** (*Command*) 37  
Console messages 28  
Conventions  
    common {DSK} and {UNIX} 32  
    {DSK} naming 33  
    fonts 3  
    Medley devices 3,17,35  
    notation 3  
    URAIID 39,41  
Copy protection 11  
**CREATIONDATE** (*File Attribute*) 38  
.cshrc file 13, A-1

## D

Daylight Savings Time  
    setting values for 12,22  
**DEFAULTFILETYPE** (*Variable*) 38,39  
**DEFAULTFILETYPEPELIST** (*Variable*) 38,39  
**DEFAULTPRINTERTYPE** (*Variable*) 12,21  
**DEFAULTPRINTINGHOST** (*Variable*) 12,21  
**DIRECTORIES** (*Variable*) 12,21

## Directory

    changing 32,37  
    creation 36,37  
    deletion 37  
    enumeration 36  
    home 32,37  
    name delimiting 31  
    parent 32  
**DISKFREEPAGES** (*Function*) D-1  
**DISKPARTITION** (*Function*) D-1  
Display functions 27  
Display fonts, how to find 12,21  
**DISPLAYFONTDIRECTORIES** (*Variable*) 12,21  
DLRS232C D-2  
DLTTY D-2  
{DSK} 26,32,36, 38  
    special characters 32  
    naming conventions 33  
{DSK}INIT. 12, 16  
**DSKDISPLAY** (*Function*) D-1

## E

Emulator 2  
**\EndDST** (*Variable*) 13,22  
End-of-line convention 4,38,40  
Environment variable A-1  
    LDEDESTSYSOUT 23  
    LDEINIT 11,16  
    LDESOURCESYSOUT 14,15  
    obtaining value of 26  
**EOL** (*File Attribute*) 38  
Errors  
    fatal 44  
    file system 39,41  
    Lisp 44

## F

**fg** (*UNIX Command*) 25  
File attributes 38  
File name  
    conventions 31  
    recognition 32  
File protection bits, changing 38  
file resources, exceeding 44  
File streams 32  
File system errors 39  
File types 39  
File variables 39  
**FILENAMEFROMID** (*Function*) D-1  
Files  
    binary 4,38,40  
    finding 12,21  
    open 37, 44  
    text 38  
    transfer 38  
    versionless 34  
**FINDFILE** (*Function*) 35  
**FLOPPY.ARCHIVE** (*Function*) D-1  
**FLOPPY.CAN.READP** (*Function*) D-1  
**FLOPPY.CAN.WRITEP** (*Function*) D-1  
**FLOPPY.FORMAT** (*Function*) D-1  
**FLOPPY.FREE.PAGES** (*Function*) D-1

**FLOPPY.FROM.FILE** (Function) D-1  
**FLOPPY.MODE** (Function) D-1  
**FLOPPY.NAME** (Function) D-1  
**FLOPPY.SCAVENGE** (Function) D-1  
**FLOPPY.TO.FILE** (Function) D-1  
**FLOPPY.UNARCHIVE** (Function) D-1  
**FLOPPY.WAIT.FOR.FLOPPY** (Function) D-1  
Font directories C-3  
Fonts 2  
    font conventions 3  
    Interpress 12,21  
Frames, viewing 41,43  
Functions  
    display and keyboard 27  
    environment inquiry 26  
    Lisp-stopping 25  
    login 26  
    system environment 24  
    timer and clock 27  
    VM 25

**G**

**GETFILEINFO** (Function) 38

**H**

Hardware, requirements 1,24  
Host access key 13  
Host ID 11  
    identifying 24  
Host name, identifying 24  
Hosts supported by Medley  
    {CORE} 31  
    {DSK} 31  
    {LPT} 31  
    {NULL} 31  
    {UNIX} 31

**I**

Input/output devices, requirements 1  
install.sunosX 1  
Installation  
    preparation 7  
    script 9  
    software 9  
    tape layout C-1  
        extracting files from C-4  
Installation Options Menu 9  
Interlisp package 39,41  
InterPress files 39  
InterPress fonts, finding 12,21  
**INTERPRESSFONTDIRECTORIES** (Variable) 12,21

**K**

Kermit 38  
**KEYACTION** (Function) 15,17  
Keyboard functions 27  
Keyboard template  
    Sun 3 15,18  
    Sun 4 15,18  
Keyboard tone generator 27  
**KEYBOARDEDITOR** D-2  
**KEYDOWNP** (Function) 15,17

**kill** (UNIX Command) 42,44

**L**

lde 13,22, A-3  
    killing 42,44  
**ldechecksum** (Command) 1, B-1  
**LDEDESTSYSOUT** (Variable) 23  
ldeether 13, A-3  
**LDEFILETIMEOUT** (Variable) 40  
**LDEINIT** (Variable) 11,16  
**LDEKBDTYPE** (Variable) 16,19  
**LDEKBDTYPE** (Variable) 17  
**LDESRCESYSOUT** (Variable) 14,15  
Left angle bracket 33,35  
Library files, finding 12,21  
Line feed 4,38,40  
Lisp symbols  
    set in site initialization file 12,21  
**LISP-RELEASE-VERSION** (Variable) 24  
LispUsers' Modules 5  
    finding 12,21  
**LISPUSERSDIRECTORIES** (Variable) 12,21  
litatom 39,41  
**LOGIN** (Function) 26  
Login functions 26  
**LOGOUT** (Function) 22,23  
**\*LONG-SITE-NAME\*** (Variable) 12, 22  
**LONG-SITE-NAME** (Function) 12,22

**M**

**MACHINE-INSTANCE** (Function) 24  
**MACHINE-TYPE** (Function) 24  
**MACHINE-VERSION** (Function) 24  
**MACHINETYPE** (Function) 24  
Medley, exiting 23  
Memory, requirements 1  
MP errors 39,41  
**\MY.NSADDRESS** (Variable) 24

**N**

Naming onventions 32, 33  
Network address, identifying 24  
NFS service 40

**O**

Object file A-2  
**OPENFILE** (Function) 38  
**OPENSTREAM** (Function) 38  
Operating system requirements 2  
Options, adding 11

**P**

Packages 3  
Passwords, maintaining for access 26  
Pathname, Lisp 31,35  
Period  
    single 32,37  
    double 32,37  
Personal init file, set up 22  
**PLAYTUNE** (Function) 27

Postscript 1  
 Postscriptstream Module 5  
 Press files 39  
 Printers 1  
     default 12,21  
**PROTECTION** (*File Attribute*) 38  
 Protocol  
     Ethernet 2  
     PUP 13  
**pstat** (*UNIX Command*) 8

PUP protocol 13, 38  
     installation 3,8  
**PURGEDSKDIRECTORY** (*Function*) D-1

## R

**\RCLKMILLISECOND** (*Variable*) 28  
**READSYS** 42,44  
**REALMEMORYSIZE** (*Function*) 24  
 Relative pathnames 33  
 Release contents 4, 5  
 Relinking A-3  
 Right angle bracket 31  
**RINGBELLS** (*Function*) 27  
**rmdir** (*UNIX Command*) 37  
**root** (*UNIX Command*) 13  
 RPC 5

## S

**SAVEVM** (*Function*) 22,23  
**setenv** (*UNIX Command*) 23  
**SETMAINTPANEL** (*Function*) 27  
**SETPASSWORD** (*Function*) 26  
**SETTIME** (*Function*) 28  
**setuid** (*UNIX Command*) 26  
**SETUSERNAME** (*Function*) 26  
**\*SHORT-SITE-NAME\*** (*Variable*) 12, 22  
**SHORT-SITE-NAME** (*Function*) 12,22  
 Site initialization file 11,15  
     how to find 12,21  
 site-init.lisp 11,16  
**SIZE** (*File Attribute*) 38  
 Sketch files 39  
 Slash 31  
 Software requirements 2, A-2  
 Special characters 32  
 Square brackets 31  
 Stack 40,42  
 Sun type 3 keyboard 17,18  
 Sun type 4 keyboard 17, 19  
 Sun Workstations, sharing 2  
 SunOS  
     versions supported 2,37,38,39  
     Type 4 keyboard 16,19  
     console messages 28  
     directory notations 32  
     file system 31  
     username 26  
 SunOS process  
     identifying username of 26  
**SUSPEND-LISP** (*Function*) 25  
 Swap space, allocating additional 8  
 Symbols  
     set in site initialization file 12,21  
 Sysout 2,13,39,41  
     files 37, 39  
     locations of 13,15  
**SYSOUT** (*Function*) 24,45  
 System administrator 9,13

## T

TCP D-1  
 TCPCHAT D-1  
 TCP/IP 38

Teleraid 39,41

## Template

Sun 3 keyboard 15,18  
Sun 4 keyboard 15,18

Text files 4,38,40

tilde 32

tilde-slash 35,37

**TIME** (Function) 28

Timers 27

/tmp/XXXX-lisp.log 29

**TYPE** (File Attribute) 38,39

## U

{UNIX} 26,36,37,38

naming conventions 35

**{UNIX}** (Function) 32

UNIX process, suspending 25

**UNIX-FULLNAME** (Function) 26

**UNIX-GETENV** (Function) 26

**UNIX-GETPARG** (Function) 26

**UNIX-USERNAME** (Function) 26

URAIID 39,41

commands 40,42

quit 40,42

/usr/share/lde 1, 2, B-1

User IDs, maintaining for access 26

**USERGREETFILES** (Variable) 12,21

**USERNAME** (Function) 26

## V

### Version

identifying machine 24

numbering 33

numbers 32,35

**VIDEOCOLOR** (Function) 27

**VIDEORATE** (Function) 27

VIRTUALKEYBOARD D-2

### Virtual memory

saving 23,24,25

saving with URAID 42,44

VM functions 25

**VMEM.PURE.STATE** (Variable) 25

**VMEMSIZE** (Function) 23,25

**VOLUMES** (Function) D-1

**VOLUMESIZE** (Function) D-1

VRAID 42,44

## W

**WRITEDATE** (File Attribute) 38

## X

### X Windows 16

Medley window 17

preparing to run Medley 13,16

running Medley remotely 16

starting 17

XNS Ethernet protocol 2, 8,13

enabling A-3

## [

[] 31

## \

**\BeginDST** (Variable) 12,22

**\EndDST** (Variable) 13,22

**\MY.NSADDRESS** (Variable) 24

**\RCLKMILLISECOND** (Variable) 28

## {

{CORE} 31

{DSK} 26,32,36,38

special characters 32

naming conventions 33

{DSK}INIT. 12,16

{LPT} 31

{NULL} 31

{UNIX} 26,36,37,38

naming conventions 35

**{UNIX}** (Function) 32

## ~

~ 32

~/ 35,37

~/lisp.virtualmem 14,15,22,23

## \*

\* 36

**\*LONG-SITE-NAME\*** (Variable) 12,22

**\*SHORT-SITE-NAME\*** (Variable) 12,22

## .

. 32,37

.. 32,37

.cshrc file A-1;13

.login file 13

## /

/ 31

/install.sunosx A-3

/usr/share/lde 1, 2, B-1

## <

< 33,35

## >

> 31



*Venue*

*Lisp Library Modules*



Address comments to:  
Venue  
User Documentation  
1549 Industrial Road  
San Carlos, CA 94070  
415-508-9672

---

## LISP LIBRARY MODULES

Medley Release 2.0

Copyright © 1990, 1991 by Venue.

All rights reserved.

Medley is a trademark of Venue.

Xerox® is a registered trademark and InterPress is a trademark of Xerox Corporation.

Sun® is a registered trademark of Sun Microsystems, Inc.

Sun, Sun-2, Sun-3, Sun-4, SPARCstation, and SunOS are trademarks of Sun Microsystems, Inc.

DEC, VAX, VMS, and VT100 are registered trademarks of Digital Equipment Corporation.

UNIX® is a registered trademark of UNIX System Laboratories.

PostScript is a trademark of Adobe Systems.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

---

The information in this document is subject to change without notice and should not be construed as a commitment by Venue. While every effort has been made to ensure the accuracy of this document, Venue assumes no responsibility for any errors that may appear.

Text was written and produced with Venue formatting tools using Xerox printers to produce text masters. The typeface is Classic.

# TABLE OF CONTENTS

---

| Section .....          | Release ..... | Page |
|------------------------|---------------|------|
| PREFACE .....          |               | vii  |
| INTRODUCTION.....      |               | ix   |
| 4045XLPstream .....    | 1.0 .....     | 1    |
| Browser .....          | 1.15 .....    | 11   |
| Cash-File .....        | 1.0 .....     | 15   |
| Centronics .....       | 1.0 .....     | 17   |
| CharCodeTables .....   | 1.0 .....     | 19   |
| Chat.....              | 1.15 .....    | 21   |
| CmlFloatArray .....    | 1.0 .....     | 29   |
| Color.....             | 1.2 .....     | 33   |
| CopyFiles .....        | 1.15 .....    | 45   |
| DataBaseFns .....      | 1.0 .....     | 49   |
| DEdit .....            | 1.0 .....     | 51   |
| EditBitMap .....       | 1.15 .....    | 61   |
| EtherRecords .....     | 1.0 .....     | 69   |
| FileBrowser .....      | 1.0 .....     | 71   |
| FontSample .....       | 1.0 .....     | 95   |
| FTPServer .....        | 1.0 .....     | 97   |
| FX-80Driver .....      | 1.0 .....     | 99   |
| GChax .....            | 1.0 .....     | 105  |
| Grapher .....          | 1.15 .....    | 111  |
| GraphZoom .....        | 1.0 .....     | 125  |
| Hash.....              | 1.0 .....     | 127  |
| Hash-File .....        | 1.0 .....     | 133  |
| HRule .....            | 1.0 .....     | 139  |
| Kermit and Modem ..... | 1.0 .....     | 143  |
| Keyboard Editor .....  | 1.0 .....     | 149  |
| MasterScope .....      | 1.15 .....    | 157  |
| Match .....            | 1.0 .....     | 183  |
| MatMult .....          | 1.0 .....     | 193  |
| MiniServe .....        | 1.0 .....     | 199  |
| NSMaintain.....        | 1.0 .....     | 203  |

## TABLE OF CONTENTS

---

|                         |            |         |
|-------------------------|------------|---------|
| Press .....             | 1.0 .....  | 213     |
| ReadNumber .....        | 1.15 ..... | 215     |
| RS232 .....             | 1.0 .....  | 219     |
| SameDir .....           | 1.0 .....  | 231     |
| Spy .....               | 1.0 .....  | 233     |
| SysEdit .....           | 1.0 .....  | 241     |
| TableBrowser .....      | 1.0 .....  | 243     |
| TCP-IP .....            | 1.15 ..... | 253     |
| TeleRaid .....          | 1.15 ..... | 279     |
| TExec .....             | 1.15 ..... | 287     |
| TextModules .....       | 1.15 ..... | 289     |
| UNIXChat .....          | 2.0 .....  | 303     |
| UNIXComm .....          | 2.0 .....  | 305     |
| Virtual Keyboards ..... | 1.2 .....  | 307     |
| Where-Is .....          | 1.2 .....  | 313     |
| INDEX .....             |            | INDEX-1 |

# LIST OF FIGURES

---

| Figure                                   | Page |
|------------------------------------------|------|
| 1. FX-80 printer drivers .....           | 99   |
| 2. FX-80 DIP switch settings .....       | 100  |
| 3. Hash File format .....                | 135  |
| 4. Horizontal rules .....                | 140  |
| 5. Built-up rules .....                  | 141  |
| 6. Character display .....               | 151  |
| 7. Character sets .....                  | 152  |
| 8. Virtual keyboard editing window ..... | 155  |
| 9. Keyboard layout display .....         | 309  |

[This page intentionally left blank]

# PREFACE

---

The *Lisp Library Modules* manual describes the library modules. These modules can be loaded into your sysout to provide additional functionality to your Lisp environment.

## Organization of This Manual

---

For ease of reference, the library modules appear in alphabetical order by software module name. This makes it easy to find the instructions and description of a particular module without having to consult the index or table of contents.

Each module includes the following:

- General description of the module
- Requirements to run the module
- Installation instructions
- User interface description
- Functions that are part of the module
- Information on implementation of the module
- Any limitations
- Samples where appropriate.

The descriptions of many library modules are related to functions, variables and concepts documented in the *Interlisp-D Reference Manual*, or *IRM*. Because of the extensive changes in this release, any advice to "see the *IRM*" means that you should also check the *Lisp Release Notes* for the latest, most accurate information.

## Conventions

---

Conventions used in the *Lisp Library Modules* manual include the following:

Names of Interlisp functions, macros and variables are shown in UPPERCASE Titan; their arguments are in *Classic ITALICS*.

Names of Common Lisp functions, macros and variables are shown in lowercase Titan; their arguments are in *Classic italics*.

A backslash (\) character preceding a function or variable name signifies that it is a system-internal name.

Examples are shown in Titan.

Messages displayed in the prompt window are shown in **Titan bold**.

Revision bars in the right margin indicate information that has been added or modified since the last release.

References to the *Interlisp-D Reference Manual*, or *IRM*, are used throughout this manual.

## References

---

We recommend that you use the *Lisp Library Modules* manual with the following publications:

*Interlisp-D Reference Manual*, Volumes I through III, Koto Release, 1985.

*Common Lisp, the Language, Second Edition*, by Guy L. Steele Jr., Digital Press, 1990.

*Common Lisp Implementation Notes*, Lyric Release, 1987.

*Kermit: A File Transfer Protocol*, by F. DaCruz, Digital Press, 1987.

*Lisp Documentation Tools*, (includes "A User's Guide to TEdit" and "A User's Guide to Sketch"), Lyric Release, 1987.

*Lisp Release Notes*, Medley Release, 1988.

*Medley 2.0 User's Guide*, Medley Release, 1991.

[This page intentionally left blank]



# INTRODUCTION

---

This information pertains to the library modules as a whole, as well as the interaction among modules. This section contains changes that have occurred in the library modules since the Lyric release. Changes for Medley Release 1.2 are indicated with revision bars in the right margin.

Primary change:

- The new Color module has been added at the end of the binder.

## What You Should Look For

---

Before running the modules, pay particular attention to the file dependencies section below. Most modules use other files, which then must be accessible to the system, and often you'll also need files that are not loaded automatically.

Load the library modules with the FILESLOAD function, using the module's name without an extension. Most modules on the distribution contain the extensions .LCOM or .DFASL in their names to designate their compiled form. The exceptions are ETHERRECORDS and SYSEDT, which are currently distributed in source form. We have attempted to include the correct extension in the descriptions of each module contained in this manual. However, (FILESLOAD...) normally loads the correct files, regardless of their form.

## Modules that are New, Moved, or Replaced

---

### Modules Added to the Library

Color

## Details of Changes

---

The following module was added to the library with the Medley Release 1.2.

### Color

Now supports color Sun Workstations.

## File Dependencies

---

Some modules require that another module be loaded into the sysout in order to run. Automatic dependencies are implemented in the source code, so that the module will load the files it depends on. Contingent dependencies are those files that you may need to load via commands typed into the executive window.

Some modules also depend of specific FONT files. As of this writing the best advice we can give to help you avoid difficulty is that you should make sure all your English font

## INTRODUCTION

---

files are loaded in an accessible directory and that your DISPLAYFONTDIRECTORIES and INTERPRESSFONTDIRECTORIES variables are set accordingly.

| <b>LIBRARY MODULE</b> | <b>AUTOMATIC DEPENDENCIES</b><br>(not including system files) | <b>CONTINGENT DEPENDENCIES</b><br>(of module)                                                                                                                                                                                                                                                                                                                                                      |
|-----------------------|---------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                       | 4045XLPSTREAM                                                 | DLRS232C or DLTTY or CENTRONICS                                                                                                                                                                                                                                                                                                                                                                    |
|                       | BROWSER<br>GRAPHER                                            | MASTERSCOPE                                                                                                                                                                                                                                                                                                                                                                                        |
|                       | CASH-FILE<br>HASH-FILE                                        |                                                                                                                                                                                                                                                                                                                                                                                                    |
|                       | CHAT<br>DMCHAT<br>CHATTERMINAL                                | PUPCHAT or NSCHAT or RS232CHAT or<br>TTYCHAT<br>and<br>DMCHAT (default) or<br>TCPCHAT or VTCHAT or TEDITCHAT<br>and the corresponding explicit dependencies:<br>PUPCHAT<br>CHAT<br>RS232CHAT<br>DLRS232C<br>see "RS232C" below<br>CHAT<br>TTYCHAT<br>DLTTY<br>see "RS232C" below<br>RS232CHAT<br>see above<br>CHAT<br>DMCHAT<br>CHATTERMINAL<br>TCPCHAT<br>see "TCP-IP" below<br>VTCHAT<br>VT100KP |
|                       | DATABASEFNS                                                   | MASTERSCOPE                                                                                                                                                                                                                                                                                                                                                                                        |
|                       | DEDIT<br>DEDITPP                                              |                                                                                                                                                                                                                                                                                                                                                                                                    |
|                       | EDITBITMAP<br>READNUMBER                                      | SCALEBITMAP                                                                                                                                                                                                                                                                                                                                                                                        |
|                       | FILEBROWSER<br>TABLEBROWSER                                   | Printer drivers, fonts<br>TEDIT, SEDIT, DEDIT                                                                                                                                                                                                                                                                                                                                                      |
|                       | FONTSAMPLE                                                    | FONTSHEETx.IP                                                                                                                                                                                                                                                                                                                                                                                      |
|                       | FX-80DRIVER                                                   | DLRS232C or DLTTY                                                                                                                                                                                                                                                                                                                                                                                  |
|                       | GRAPHZOOM                                                     |                                                                                                                                                                                                                                                                                                                                                                                                    |

GRAPHER

---

## INTRODUCTION

---

|                                                                                                                                                                                                                                                         |                                     |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|
| HRULE                                                                                                                                                                                                                                                   | IMAGEOBJ<br>EDITBITMAP<br>TEDIT     |
| KERMIT<br><br>KERMITMENU<br>KERMIT                                                                                                                                                                                                                      | CHAT<br>RS232 or TCP protocols      |
| KEYBOARDEDITOR<br>VIRTUALKEYBOARDS<br>see below                                                                                                                                                                                                         |                                     |
| MASTERSCOPE<br>MSPARSE<br>MSANALYZE<br>MSCOMMON<br>MS-PACKAGE                                                                                                                                                                                           | BROWSER, DATABASEFNS, a Lisp editor |
| MINISERVE                                                                                                                                                                                                                                               | NS or PUP or XNS                    |
| NSMAINTAIN<br>DES                                                                                                                                                                                                                                       |                                     |
| PRESS<br>PUPPRINT                                                                                                                                                                                                                                       | FONT.WIDTHS                         |
| RS232<br>RS232 port :<br>DLRS232C<br>DOVERS232C<br><br>RS232CMENU<br>DLRS232C<br>see above<br>RS232CHAT<br>see above<br>TTY port :<br>DLTTY<br>DOVERS232C<br>see above<br>DLRS232C<br>see above<br>TTYCHAT<br>see CHAT above<br>RS232CMENU<br>see above | KERMIT                              |

---

---

 SPY

GRAPHER

READNUMBER

IMAGEOBJ

EDITBITMAP

see above

---

 SYSEDT

MASTERSCOPE

EXPORTS.ALL

CMLARRAY-SUPPORT

---

 TCP-IP

TCP

TCPLLIP see below

TCPCHAT

TCP see above

CHAT see above

TCPCONFIG

TCPDEBUG

TCP see above

TCPFTP

TCPNAMES

TCP see above

TCPFTPSRV

TCPFTP see below

TCPHTE

TCPLLAR

TCPLLICMP

TCPLLIP

TCPHTE

TCPLLICMP

TCPLLAR

TCPNAMES

TCPTFTP

TCPUDP see below

TCPUDP

TCPLLIP see above

---

 TELERAID

REMOTEVMEM

READSYS

RDSYS

VMEM

---

 TEXEC

TEDIT

---

 VIRTUALKEYBOARDS

DANDELIONKEYBOARDS or

DAYBREAKKEYBOARDS or

DORADOKEYBOARDS or

DOVEKEYBOARDS

---

**WHERE-IS****HASH-FILE**

---

**COLOR****MAIKOCOLOR****LLCOLOR****TAKEBIGBM**

---

---

## Additional Notes

---

DEdit is not error-protected. Doing a ↑ in a DEdit break window closes the DEdit window, too.

In addition, the modules work under all Lisp environments (Interlisp-D, Common Lisp, Xerox Common Lisp). However, many of the functions and variables used within modules are those of Interlisp-D, and therefore you will have to make sure that you use the IL: prefix when you are not in Interlisp .

---

## Koto CML Library Module

---

If you have files that used the Koto CML library module, with its package-style symbol naming conventions, you will need to convert them to use the correct symbols in Lyric and Medley. The procedure is briefly as follows: see the *Lyric Common Lisp Implementation Notes*, Chapter 11, "Reader Compatibility Feature," for complete details on this mechanism:

First, set the global variable LITATOM-PACKAGE-CONVERSION-ENABLED to T. Then for each of your files, do

```
(LOAD file 'PROP)
```

```
(MAKEFILE file 'NEW)
```

Afterwards be sure to set the global variable LITATOM-PACKAGE-CONVERSION-ENABLED back to NIL.

[This page intentionally left blank]

# INDEX

## A

Abort a Print Job 101,102  
Aborting Commands 76  
Access Functions 251  
Add New Keyboards to the List 310  
**add-form** (Function) 299  
**ADD.BORDER.TO.BITMAP** (Function) 65  
**ADD.PROCESS** (Function) 266  
Adding FileBrowser Commands 92  
Address space of the internetwork 261  
**ADJUSTCOLORMAP** (Function) 37  
Administrator Commands for NSMaintain 206  
**ADVISE** (Function) 283  
Aliases 203  
**ALLOCATE.ETHERPACKET** (Function) 270  
Analyzing user Functions 157  
Application layer 261  
ARPANET 253  
Array Functions 193  
Array operations 29  
Assignments 189

## B

**BACKWARD FIND** (Command) 288  
**BEGIN DST** (Variable) 199  
**BIN** (Function) 222  
**BIT.IN.COLUMN** (Function) 65  
**BIT.IN.ROW** (Function) 65  
**BITBLT** (Function) 116  
**BITMAPCREATE WIDTH HEIGHT BITS PER PIXEL**  
(Function) 38  
**BITS PER PIXEL BITMAP** (Function) 38  
**BOUT** (Function) 222  
**BREAK** (Function) 283  
Broadcast address 262  
Browser 111  
    describe window 12  
    printout window 12  
BROWSER window 74  
**BROWSERBOXING** (Variable) 12  
**BROWSERFORMAT** (Variable) 12

## C

C-Shell 303  
C4 printed circuit board 1  
**CACHE/NODE/LABEL/BITMAPS** (Variable) 116  
Call graph 233  
Call stack 233  
Calling the Keyboard Editor 150  
**CALLS** (Function) 175  
**CALLSCCODE** (Function) 175  
Cash-File 15  
Centronics 17  
    cable 17  
    port 1  
**CENTRONICS.RESET** (Function) 17  
**CH.DEFAULT.DOMAIN** (Variable) 206  
**CH.DEFAULT.ORGANIZATION** (Variable) 206  
Changing an Existing Keyboard 150  
Character framing errors 224  
CharCodeTables 19

Chat 21,303  
**CHAT** (Function) 22,303  
Chat connections 22  
Chat Menu 23  
Chat window 146  
**CHAT.ALLHOSTS** (Variable) 25  
**CHAT.DISPLAYTYPES** (Variable) 24  
**CHAT.EMACSCOMMANDS** (Variable) 25  
**CHAT.FONT** (Variable) 25  
**CHAT.IN.EMACS?** (Variable) 26  
**CHAT.INTERRUPTS** (Variable) 24  
**CHAT.KEYACTIONS** (Variable) 24  
**CHAT.PROTOCOLTYPES** (Variable) 26  
**CHAT.TTY.PROCESS** (Variable) 25  
**CHAT.WINDOW.REGION** (Variable) 25  
**CHAT.WINDOW.SIZE** (Variable) 25  
Chatting to a host 258  
**CHECKSAMEDIR** (Function) 231  
Clean-Up After Copying Files 46  
Clearinghouse 26  
Clearinghouse service 203  
**CLOSE** (Function) 305  
**close-hash-file** (Function) 134  
**CLOSECHATWINDOWFLG** (Variable) 25  
**CLOSEF** (Function) 9,222  
**CLOSEHASHFILE** (Function) 128  
CmlFloatArray 29  
**CMYCOLORMAP** (Function) 36  
**COLLECTINUSE** (Function) 108  
**COLLISIONS** (Function) 108  
**COLORDISPLAY** (Function) 33  
**COLORDISPLAYP** (Function) 33  
**COLORIZEBITMAP** (Function) 44  
**COLORLEVEL** (Function) 37  
**COLORMAPCOPY** (Function) 36  
**COLORMAPCREATE** (Function) 36  
**COLORNAMES** (Association List) 34  
**COLORNUMBERP** (Function) 35  
**COLORSCREEN** (Function) 39  
**COLORSCREENBITMAP** (Function) 38  
COMMAND menu 74  
Common Lisp 288  
**COMPILE** (Command) 86  
Conjunctions of Sets 168  
Connecting to a host 258  
**Control-E** (Command) 76  
**Control-F** (Command) 53  
**Control-O** (Command) 139  
**Control-P** (Command) 280  
**Control-Q** (Command) 75,87  
**Control-S** (Command) 53  
**Control-Z** (Command) 53  
Conversation Mode 45  
**convert-loaded-files** (Variable) 292  
**COPY** (Command) 78  
Copy-Selection 53,77  
**COPYFILE** (Function) 46,267  
CopyFiles 45  
**COPYFILES** (Function) 45  
**COPYHASHFILE** (Function) 130  
CPE FP 17  
CPE-FP upgrade 1  
Create a Key Pad for Repeated Use 216  
**CREATE-PROCESS-STREAM** (Function) 305  
**CREATE.NUMBERPAD.READER** (Function) 216  
**CREATEHASHFILE** (Function) 128



**CREATEW** (*Function*) 41

Creating

- 4045XLP Master Files 4
- a graph 112
- a key pad 215
- a new bitmap 64
- a new keyboard 149
- a TableBrowser 247
- horizontal rules 139
- HOST.TXT file 254
- new keyboard configurations 153
- objects 206
- the local IP.INIT file 256

Creation dates 46

Cumulative mode 238

**CURSORPOSITION** (*Function*) 44

**CURSORSscreen** (*Function*) 44

Customizing

- Chat 24
- FileBrowser 89

Cyclic graphs 111

**D**

DataBaseFns 49

Databases 313

- using 313,314

Datamedia 2500 21

Daughter 114

Debuggee 279

Debugger 279

**DECREASING.FONT.LIST** (*Variable*) 125

DEdit 51,307

- Command menu 54

- Functions 54

- idioms 59

- parameters 57

- window 52

**DEDITLINGER** (*Variable*) 57

**DEDITTYPEINCOMS** (*Variable*) 57

**DEFAULT-CLEANUP-COMPILER** (*Variable*) 86

**DEFAULT.GRAPH.WINDOWSIZE** (*Variable*) 117

**DEFAULTCHATHOST** (*Variable*) 22,25

**DEFAULTPRINTERTYPE** (*Variable*) 2

**DEFAULTPRINTINGHOST** (*Variable*) 2,213

**defdefiner-macros** (*Variable*) 292

**DEFINEKEYBOARD** (*Function*) 311

Defining a Virtual Keyboard 310

**DEGREES-TO-RADIANS** (*Function*) 196

**DELETE** (*Command*) 77

Deleting objects 206

Device Errors 17

Differences between TExec and TEdit 287

Digital VT100 21

Directed acyclic graph 113

**DIRECTORIES** (*Variable*) 2,258

Directory-only lines 75

**DIRECTORYNAME** (*Function*) 231

Display-Only a Keyboard 309

Displayer 233

**DISPLAYFONTDIRECTORIES** (*Variable*) 100

**DISPLAYGRAPH** (*Function*) 117

Displaying

- a graph 116

- a stack 280

Displaystream graphics 288

DM2500 Chat 27

**DODIP.HOSTP** (*Function*) 272

Domain 203  
     manipulating 208  
**DOMAIN.GRAPH** (Function) 276  
**DOMAIN.INIT** (Function) 276  
**DOMAIN.LOOKUP** (Function) 276  
**DOMAIN.LOOKUP ADDRESS** (Function) 276  
**DOMAIN.LOOKUP NAMESERVER** (Function) 276  
**DOMAIN.TRACE** (Function) 276  
**DRAWBETWEEN** (Function) 43  
**DRAWLINE** (Function) 43  
**DRAWTO** (Function) 43  
**DSPBACKCOLOR** (Function) 42  
**DSPCOLOR** (Function) 42  
**DT.EDITMACROS** (Variable) 58  
**DUMPDATABASE** (Function) 177  
**DUMPDB** (Function) 50  
**DUMPGRAPH** (Function) 121  
 Dvorak keyboard 311  
 Dynamic structure 233

## E

E-30 option kit 1  
 E30 option 219  
**EDIT** (Command) 84  
 Edit buffer window 52  
**EDIT.BITMAP** (Function) 61  
 EditBitMap 61  
     sub-menu 62  
     window 62  
**EDITBM** (Function) 61  
**EDITCOLORMAP** (Function) 37  
**EDITCONFIGURATION** (Function) 154  
**EDITEMBEDTOKEN** (Variable) 57  
**EDITGRAPH** (Function) 121  
 EDITGRAPHMENU 119  
**EDITGRAPHMENUMCOMMANDS** (Variable) 119  
 Editing  
     a bitmap in a document 64  
     editing a graph 117  
     existing bitmap 63  
     keyboard configuration 154  
**EDITKEYBOARD** (Function) 150  
**EDITMODE** (Function) 52  
**EDITMODE** (Variable) 84  
 Effecting MasterScope Analysis 171  
 Element patterns 185  
 Embedding and extracting 59  
**ENDDST** (Variable) 199  
 Entering DEdit Commands 54  
 Environmental and Lisp mappings 95  
**EOLCONVENTION** (Variable) 272  
 Error Condition Reporting 224  
**ERRORPUP** (record) 70  
**ERRORXIP** (record) 69  
 Establishing a Connection 143  
 et FX-80 Fast Mode 101  
**ETHERHOSTNUMBER** (Variable) 4,280  
 Ethernet 260  
 Ethernet packet 264  
 EtherRecords 69  
**EVALUATE-POLYNOMIAL** (Function) 31  
 expr definitions of Functions 171  
**EXPUNGE** (Command) 86  
 Extend the selection 75  
 Extended Processor board 17  
 Extended processor option 29  
 External Communication Service 27

## F

**FASSOC** (Function) 121  
 Fast mode FX-80 101  
**FASTFX80-DEFAULT-DESTINATION** (Variable) 100  
**FB** (Command) 90  
**FB.ALLOW.ABORT** (Function) 94  
**FB.BROWSERFONT** (Variable) 92  
**FB.DEFAULT.EDITOR** (Variable) 84,92  
**FB.DEFAULT.INFO** (Variable) 87,90  
**FB.DEFAULT.NAME.WIDTH** (Variable) 91  
**FB.FETCHFILENAME** (Function) 93  
**FB.HARDCOPY.DIRECTORY.FONT** (Variable) 92  
**FB.HARDCOPY.FONT** (Variable) 92  
**FB.ICONFONT** (Variable) 91  
**FB.INFO.FIELDS** (Variable) 91  
**FB.INFO.MENU.ITEMS** (Variable) 91  
**FB.MENU.ITEMS** (Variable) 92  
**FB.PROMPTFONT** (Variable) 92  
**FB.PROMPTLINES** (Variable) 92  
**FB.PROMPTW.FORMAT** (Function) 93  
**FB.PROMPTW.PRINT** (Function) 93  
**FB.PROMPTFORINPUT** (Function) 94  
**FB.SELECTEDFILES** (Function) 93  
**FB.TABLEBROWSER** (Function) 93  
 File Manager functions, changed 313  
 File name translation Function 269  
 File transfer using RS232 226  
**FILEBROWSE** (Command) 84  
 FileBrowser 71  
     Functions 89  
     Variables 90  
**FILEBROWSER** (Function) 89  
**filecom-specifier** (Function) 299  
 Filing operations 267  
**FILING.ENUMERATION.DEPTH** (Variable) 92  
**FIND-ARRAY-ELEMENT-INDEX** (Function) 31  
**FINDPOINTER** (Function) 109  
**FINDPOINTERS.OF.TYPE** (Function) 109  
**FLIPNODE** (Function) 120  
 Floating-point vector 29  
**FNT.DISPLOOK** (Function) 96  
**FNT.DISPTBLE** (Function) 96  
**FNT.MAKEBOOK** (Function) 95  
 Font mappings 95  
 FontSample 95  
**FORCEOUTPUT** (Function) 222  
 Forest 113  
**form-specifier** (Function) 299  
**FORWARD FIND** (Command) 288  
**FREEVARS** (Function) 175  
**FROM.SCREEN.BITMAP** (Function) 66  
 FTP service 268  
**FTPDEBUG** (Function) 268  
 FTPserver 97  
**FTPSEVER** (Function) 4,97  
**FTPSEVER.DEFAULT.HOST** (Variable) 97  
 Function-calling structures 111  
 Functions for saving work 283  
 Functions for writing routines 176  
 FX Printer Compatibility 102  
 FX-80  
     DIP Switch Settings 100  
     driver 99  
     family 99  
     serial Interface 99  
**FX80-PRINT** (Function) 102

**G**

GapTelnet 26  
 Garbage collector 284  
 Gateway Access Protocol 26  
 GCHax 105  
 General Purpose Records 69  
 Get, Set Parameters via Inspector Window 7  
**get-cash-file** (Function) 15  
**GETBOXSCREENPOSITION** (Function) 40  
**GETBOXSCREENREGION** (Function) 41  
**GETFILEINFO** (Function) 267  
**GETHASHTEXT** (Function) 129  
**GETHASHVILE** (Function) 129  
**GETRELATION** (Function) 176  
**GETSCREENPOSITION** (Function) 40  
**GETTEMPLATE** (Function) 174  
**GETTERMTABLE** (Function) 288  
 Getting Hardcopy Directory Listings 77  
 Ghost boxes 238  
 Graph 11  
     data structure 120  
     nodes 111  
**GRAPH** (record) 121  
**GRAPH/HARDCOPY/FORMAT** (Variable) 117  
 Grapher 111,233  
 Grapher image objects 120  
**GRAPHEROBJ** (Function) 120  
**GRAPHERPROP** (Function) 121  
 Graphics interface 99  
**GRAPHNODE** (record) 122  
**GRAPHREGION** (Function) 120  
 GraphZoom 125  
**GRAYCOLORMAP** (Function) 36  
 Groups 207

**H**

HANZON Universal Card 99  
**HARDCOPY** (Command) 81  
**HARDCOPYGRAPH** (Function) 117  
 Hash tables 178  
 Hash-File 133  
**HASHFILEDEFAULTSIZE** (Variable) 130  
**HASHFILEDTBL** (Variable) 130  
**HASHFILENAME** (Function) 129  
**HASHFILEP** (Function) 129  
**HASHFILEPLST** (Function) 130  
**HASHFILEPROP** (Function) 129  
**HASHLOADFACTOR** (Variable) 130  
**HFGROWTHFACTOR** (Variable) 131  
**HLS** (Record) 35  
**HLSP** (Function) 35  
**HOST&DIRECTORYFIELD** (Function) 231  
**HOSTS.TEXT.DIRECTORIES** (Variable) 276  
 HOSTS.TXT files, parsing 276  
**HPRINT** (Function) 121  
**HQFX80-DEFAULT-DESTINATION** (Variable) 100  
**HQFX80-FONT-DIRECTORIES** (Variable) 100  
 HRule 139  
**HRULE.CREATE** (Function) 139  
**HTE.READ.FILE** (Function) 276  
 Hue-Lightness-Saturation Triples 35

**I**

I/O processor 226  
**IDENTITY-3-BY-3** (Function) 194  
**IDENTITY-4-BY-4** (Function) 194  
**il:typesof** (Function) 313  
**il:whereis** (Function) 313  
 Imagestream 5  
 Include all files, both deleted and undeleted 75  
 Individual moe 239  
  
 INFO menu 74  
 Info Options window 87  
 Input conventions for FileBrowser commands 75  
 Inserting a segment 59  
**install-form** (Function) 299  
**INTENSITIESFROM** (Function) 36  
**INTERACT&ADD.BORDER.TO.BITMAP** (Function) 66  
**INTERACT&SHIFT.BITMAP.DOWN** (Function) 66  
**INTERACT&SHIFT.BITMAP.LEFT** (Function) 66  
**INTERACT&SHIFT.BITMAP.RIGHT** (Function) 66  
 Interactive File Transfers With Kermit or Modem 146  
 Interactive Terminal Service 27  
 Internal fields 241  
 Internet layer 260  
**INTERPRESSFONTDIRECTORIES** (Variable) 19  
 Interpreter 239  
 Interrupt character 234  
**INVERT.BITMAP.B/W** (Function) 65  
**INVERT.BITMAP.DIAGONALLY** (Function) 65  
**INVERT.BITMAP.HORIZONTALLY** (Function) 65  
**INVERT.BITMAP.VERTICALLY** (Function) 65  
 Invisible characters 288  
 IP  
     addresses 262  
     networks primer 261  
     packet building 274  
     packet sending 275  
     socket access 272  
**IP.ADD.PROTOCOL** (Function) 273  
**IP.APPEND.BYTE** (Function) 275  
**IP.APPEND.CELL** (Function) 275  
**IP.APPEND.STRING** (Function) 275  
**IP.APPEND.WORD** (Function) 275  
**IP.CLOSE.SOCKET** (Function) 274  
**IP.DELETE.PROTOCOL** (Function) 274  
**IP.OPEN.SOCKET** (Function) 274  
**IP.SETUPIP** (Function) 275  
**IP.TRANSMIT** (Function) 275  
**IPHOSTNAME** (Function) 273  
**IPINIT** (Function) 272  
**IPTRACE** (Function) 273  
 Iterative statement operator 177

**J**

**join-comments** (Variable) 292

**K**

Kermit 144,226  
 Kermit menu 146  
**KERMIT.RECEIVE** (Function) 145  
**KERMIT.SEND** (Function) 144  
**KEYACTION** (Function) 153  
 Keyboard editor menus 150

KeyboardEditor 149

## L

Landscape mode 1  
 Landscape printing 103  
 Lattices 111  
 Laying out a graph for display 112  
**LAYOUTGRAPH** (Function) 112  
**LAYOUTSEXPR** (Function) 115  
**LDESHELL** (Variable) 303  
 Library module changes 5-9  
     summary 5-9  
 Library module dependencies 9  
 Linguistic tree 120  
 Link layer 260  
 Lisp interrupts 22  
**LISPUSERSDIRECTORIES** (Variable) 258  
**LISPXREAD** (Function) 53  
 List of nodes 112  
 List structure editor 51  
**LISTFILES** (Function) 3,213  
**LOAD** (Command) 84  
**LOAD** (Function) 49  
 Load New Keyboards 310  
**load-textmodule** (Function) 290  
**LOADDB** (Function) 50  
**LOADDBFLG** (Variable) 50  
**LOADFROM** (Function) 49,85  
 Loading TCP 258  
**LOOKUPHASHFILE** (Function) 129  
 Lost characters 224  
 Low-level TCP Functions 272

## M

Macro 177  
 Macro Expansion 170  
**MAINSCREEN** (Function) 39  
 Maintenance panel halt 279  
**make-cash-file** (Function) 15  
**make-hash-file** (Function) 133  
**MAKE-HOMOGENEOUS-3-BY-3** (Function) 193  
**MAKE-HOMOGENEOUS-3-VECTOR** (Function) 193  
**MAKE-HOMOGENEOUS-4-VECTOR** (Function) 194  
**MAKE-HOMOGENEOUS-N-BY-3** (Function) 193  
**MAKE-HOMOGENEOUS-N-BY-4** (Function) 194  
**make-specifier** (Function) 297  
**make-textmodule** (Function) 291  
**MAKEFILE** (Function) 49,78,231  
**MAKEFILEFORMS** (Function) 231  
 Making a sysout that contains TCP-IP 259  
 Manipulating  
     domains 208  
     groups 207  
     remote Vmem 284  
**MAP-ARRAY** (Function) 29  
**MAPGC** (Function) 110  
**MAPHASHFILE** (Function) 130  
**MAPOFACOLOR** (Function) 37  
**MAPRELATION** (Function) 176  
**MASTERSCOPE** (Function) 175  
 MasterScope 11,49,157  
     Commands 158  
     database 157  
     entries 175

relations 161  
 set specifications 164  
 templates 164

Match 183  
 MatMult 193  
**MATMULT-133** (Function) 195  
**MATMULT-144** (Function) 195  
**MATMULT-331** (Function) 195  
**MATMULT-333** (Function) 195  
**MATMULT-441** (Function) 195  
**MATMULT-444** (Function) 196  
**MATMULT-N33** (Function) 195  
**MATMULT-N44** (Function) 196  
 Matrix Creation Functions 193  
 Matrix multiplication 193  
 Matrix Multiplication Functions 195  
 Merged node 237  
**MIGRATIONS** (Variable) 231  
 MiniServe 199  
 mlFloatArray 196  
 Modem 145,226  
**MODEM.RECEIVE** (Function) 145  
**MODEM.SEND** (Function) 145  
 Moving an expression 59  
**MSMACROPROPS** (Variable) 170  
**MSMARKCHANGED** (Function) 177  
**MSNEEDUNSAVE** (Function) 176  
 Multiple DEdit commands 58  
 Multiple streams 9  
**MY.NSHOSTNUMBER** (Variable) 200

## N

Network addresses 261  
 Network protocols 21,26  
**NETWORKOSTYPES** (Variable) 257  
 NEW INFO FileBrowser Command 87  
**NODECREATE** (Function) 112  
 Notecards 9  
 Noticing changes that require recompiling 177  
 NS characters 213  
 NS Chat 26  
 NS Records 69  
**NS.TO.PUP.ALIST** (Variable) 200  
**NS.TO.PUP.FILE** (Variable) 200  
**NSHOSTNUMBER** (record) 69  
 NSMaintain 203  
**NSMAINTAIN** (Function) 204  
**NSTIMESERVER** (Function) 199  
**INTERACT&SHIFT.BITMAP.UP** (Function) 66  
**NUMBERPAD.READ** (Function) 216

## O

Objects 203  
     creating 206  
     deleting 206  
 Obtaining information in NSMaintain 204  
 Obtaining network addresses 254  
**open-hash-file** (Function) 133  
**OPENHASHFILE** (Function) 128  
**OPENIMAGESTREAM** (Function) 9,101  
 Opening  
     a 4045 Stream 7  
     a Centronics Stream 17  
     a Chat Connection 22  
**OPENP** (Function) 222  
**OPENSTREAM** (Function) 222

**OPENWINDOWS** (Function) 42  
 Operations on Multiple Items 249  
 Organization 203  
**OVERFLOWS** (Function) 108

**P**

Packages 124, 286  
 Parents 114  
 parity errors 224  
**PARSERELATION** (Function) 176  
**PATLISPCHECK** (Variable) 184  
 Pattern  
   elements 185  
   match compiler 184  
   match expressions 185  
**PATVARDEFAULT** (Variable) 185  
**PERSPECTIVE-4-BY-4** (Function) 195  
 Place Markers 189  
 Portrait mode 1  
**PORTSTRING** (Function) 4,280  
**PPTCB** (Function) 266  
 Press 213  
**PRESSFONTWIDTHFILES** (Variable) 213  
 Primitive relationship 178  
**PRINTERMABLE** (Variable) 288  
**PRIN3** (Function) 124  
 Print  
   a file 101,102  
   source code in high-quality mode 103  
   source code on fast FX-80 103  
   TEdit file in fast FX-80 mode 103  
   TEdit file in high-quality mode 104  
   text and graphics in high-quality mode 103  
**print-filecom** (Function) 299  
 Printer drivers 99  
 Printer's point 139  
 Printing  
   in fast mode 101  
   in high-quality mode 102  
   source or TEdit Files 3  
   speed 9  
   via FTPserver 4  
   windows 4  
 Programmer's Assistant 288  
**PROJECT-AND-FIX-3-VECTOR** (Function) 196  
**PROJECT-AND-FIX-4-VECTOR** (Function) 196  
**PROJECT-AND-FIX-N-BY-3** (Function) 196  
**PROJECT-AND-FIX-N-BY-4** (Function) 196  
 PROMPT window 73  
 Properties 203  
 Protocol number 273  
**PUP** (record) 70  
 PUP  
   Chat 26  
   FTP 97,232  
   ID service 200  
   records 70  
   time service 200  
**PUP.ID.SERVER** (Function) 199  
**PUPADDRESS** (record) 70  
**PUPNUMBER** (Variable) 200  
**PUPTIMESERVER** (Function) 199  
**PUTHASHFILE** (Function) 128  
**PUTHASHTEXT** (Function) 129

**Q**

**QABLEITEM** (record) 69  
 Query Mode 45  
 Queue multiple arguments 60  
 Quick-scrolling 74  
 Quitting the FileBrowser 76

**R**

**RANDACCESSP** (Function) 272  
**READ-BYTE** (Function) 219  
**READ-CHAR** (Function) 219  
**READGRAPH** (Function) 121  
 Reading the Remote Vmem 284  
 ReadNumber 215  
 ReadNumber window 66  
**READP** (Function) 222  
**READSYS** (Function) 284  
**REALFRAMEP** (Function) 239  
**RECOMPILEDEFAULT** (Variable) 178  
**RECOMPUTE** (Command) 86  
 Reconstruction 190  
 Record declaration 177  
 Recursive loads 97  
 Redefine Existing Keyboards 310  
**REDUCE-ARRAY** (Function) 30  
 Red-Green-Blue Triples 34  
**REFCNT** (Function) 108  
 Reference counting 284  
**REHASHFILE** (Function) 130  
**REHASHGAG** (Variable) 131  
 Relations between sets 157  
**RELDRAWTO** (Function) 43  
 Remote  
   Kermit in Server Mode 144  
   Kermit not in Server Mode 144  
   system administration 27  
   system executive 27  
 Removing Keyboards From the Men 310

**RENAME** (Command) 81  
**REPACKFILENAME.NEW.TRANSLATION**  
   (Function) 269  
**REPACKFILENAME.OSTYPE.TABLE** (Variable)  
   269  
**REPACKFILENAME.STRING** (Function) 268  
 Replace All Known Keyboards 310  
 Replacements 190  
**RESET/NODE/BORDER** (Function) 120  
**RESET/NODE/LABELSHADE** (Function) 120  
**RESETDEDIT** (Function) 54  
 Resetting 4045XLPstream 8  
 Restarting MiniServe 201  
**RGB** (Record) 34  
**RGBCOLORMAP** (Function) 36  
**RGBP** (Function) 35  
**RNUMBER** (Function) 215  
 Ross referencing user programs 157  
**ROTATE-3-BY-3** (Function) 194  
**ROTATE-4-BY-4-ABOUT-X** (Function) 194  
**ROTATE-4-BY-4-ABOUT-Y** (Function) 194  
**ROTATE-4-BY-4-ABOUT-Z** (Function) 194  
**ROTATE.BITMAP.LEFT** (Function) 65  
**ROTATE.BITMAP.RIGHT** (Function) 65  
**ROTATECOLORMAP** (Function) 37  
 RS232 143,219  
 RS232

Chat 27  
 port 219  
 RS232C port 1  
**RS232C.CLOSE-STREAM** (Function) 222  
**RS232C.DEFAULT.INIT.INFO** (Variable) 220  
**RS232C.ERROR.STREAM** (Variable) 224  
**RS232C.GET.PARAMETERS** (Function) 221  
**RS232C.INIT** (Function) 220  
**RS232C.OTHER.STREAM** (Function) 222  
**RS232C.OUTPUT.PACKET.LENGTH** (Function) 222  
**RS232C.READP.EVENT** (Function) 223  
**RS232C.REPORT.STATUS** (Function) 224  
**RS232C.SET.PARAMETERS** (Function) 220  
**RS232C.SHUTDOWN** (Function) 221  
 RS232CHAT 225  
 RS232CMENU 225,228  
**RS232MODEMCONTROL** (Function) 223  
**RS232MODEMHANGUP** (Function) 224  
**RS232MODEMSTATUSP** (Function) 223  
**RS232MODIFYMODEMCONTROL** (Function) 223  
**RS232SEENDBREAK** (Function) 223  
 RS232TRACE 224

## S

S-expression 115  
 SameDir 231  
 Sampler 233  
**SAVEDBFLG** (Variable) 50  
 Scale factors 9  
**SCALE-3-BY-3** (Function) 194  
**SCALE-4-BY-4** (Function) 195  
**SCALEDBITBLT** (Function) 9  
**SCREENBITMAP** (Function) 39  
**SCREENCOLORMAP** (Function) 36  
**SCREENPOSITION** (Record) 39  
**SCREENREGION** (Record) 39  
 Scroll bar 74  
**SEE** (Command) 82  
 Segment Patterns 186  
 Segment selection and manipulation 59  
 Selecting Files 74  
 Selecting Objects and Lists 52  
 Selection stack 53  
 Send text output to fast FX-80 103  
**SEND.FILE.TO.PRINTER** (Function) 3,213  
 Serial interface card 99  
 Server mode 144  
**SET DEPTH** (Command) 88  
 Set  
   destination 102  
   determiners 167  
   FX-80 Destination 101  
   FX-80 Page Size 101  
   HQ Mode 102  
   page Size 102  
   specifications by Blocktypes 167  
   specifications by Relation 166  
   types 167  
**SETCOLORINTENSITY** (Function) 37  
**SETSYNONYM** (Function) 175  
**SETTEMPLATE** (Function) 174  
**SETTIME** (Function) 199  
**SHIFT.BITMAP.DOWN** (Function) 65  
**SHIFT.BITMAP.LEFT** (Function) 66  
**SHIFT.BITMAP.RIGHT** (Function) 66  
**SHIFT.BITMAP.UP** (Function) 66

SHOW PATHS 180  
**SHOW PATHS** (Command) 169  
**SHOW PATHS** (Function) 11  
**SHOW.CLOSED.WINDOWS** (Function) 110  
**SHOWCIRCULARITY** (Function) 110  
**SHOWCOLORTTESTPATTERN** (Function) 44  
**SHOWCOMMONSETS** (Function) 20  
**SHOWCSET** (Function) 20  
**SHOWCSETLIST** (Function) 20  
**SHOWCSETRANGE** (Function) 20  
**SHOWGC** (Function) 107  
**SHOWGRAPH** (Function) 116,125  
**SHOWZOOMGRAPH** (Function) 125  
 Simple item operations 248  
**SINGLEFILEINDEX** (Function) 10  
 Sketch 10  
**SORT** (Command) 89  
 Special characters 307  
**specifiers** (Variable) 297  
 Specifying the files to browse 72  
 Spy 233  
**SPY.BORDERS** (Variable) 239  
**SPY.BUTTON** (Function) 234  
**SPY.END** (Function) 234  
**SPY.FONT** (Variable) 239  
**SPY.FREQUENCY** (Variable) 239  
**SPY.LEGEND** (Function) 239  
**SPY.MAXLINES** (Variable) 239  
**SPY.NOMERGEFNS** (Variable) 239  
**SPY.START** (Function) 234  
**SPY.TOGGLE** (Function) 234  
**SPY.TREE** (Function) 235  
**SPY.TREE** (Variable) 239  
 Stack 280  
 Stack architecture 58  
 Stacking several rules in a single object 140  
 Starting FileBrowser 71  
 Starting TExec 287  
**STARTMINISERVE** (Function) 199  
 Static structure 233  
 stderr 305  
 stdin 305  
 stdout 305  
**STOPIP** (Function) 272  
**STORAGE** (Function) 105  
 Store keyboards 309  
**SUBNETMASK** (Variable) 262  
 Subnetworks 262  
 Sun,TCP to a 254  
 SunOS, interface to 305  
 Switch & Display a Keyboard 308  
 Switch A-2 2  
 Switch Keyboards 308  
 Switch settings 2  
**SWITCHKEYBOARDS** (Function) 311  
 SysEdit 241  
**SYSHASHFILE** (Variable) 131  
**SYSHASHFILELST** (Variable) 131  
 System sources 241

## T

TableBrowser 243  
**TABLEBROWSER** (record) 244  
**TABLEITEM** (record) 243  
 Tally window 73  
 Target-source-Command 59  
**TB.BROWSER.BUSY** (Function) 248

**TB.CLEAR.LINE** (Function) 249  
**TB.COLLECT.ITEMS** (Function) 250  
**TB.DELETE.ITEM** (Function) 248  
**TB.FIND.ITEM** (Function) 250  
**TB.FINISH.CLOSE** (Function) 248  
**TB.INSERT.ITEM** (Function) 249  
**TB.ITEM.DELETED?** (Function) 251  
**TB.ITEM.SELECTED?** (Function) 251  
**TB.MAKE.BROWSER** (Function) 247  
**TB.MAP.DELETED.ITEMS** (Function) 250  
**TB.MAP.ITEMS** (Function) 250  
**TB.MAP.SELECTED.ITEMS** (Function) 250  
**TB.NORMALIZE.ITEM** (Function) 249  
**TB.NTH.ITEM** (Function) 250  
**TB.NUMBER.OF.ITEMS** (Function) 249  
**TB.REDISPLAY.ITEMS** (Function) 249  
**TB.REMOVE.ITEM** (Function) 249  
**TB.REPLACE.ITEMS** (Function) 248  
**TB.SELECT.ITEM** (Function) 248  
**TB.SET.FONT** (Function) 248  
**TB.UNDELETE.ITEM** (Function) 248  
**TB.USERDATA** (Function) 251  
**TB.WINDOW** (Function) 251  
**TCP**  
     and directory enumeration 254  
     Chat 27  
     debugging aids 277  
     segment 264  
**TCP-IP** 253  
     protocol 47  
     protocol layers 260  
**TCP.BYE** (Function) 268  
**TCP.CLOSE.SENDER** (Function) 265  
**TCP.DEFAULT.RECEIVE.WINDOW** (Variable) 265  
**TCP.DEFAULT.USER.TIMEOUT** (Variable) 265  
**TCP.DEFAULT.FILETYPE** (Variable) 267  
**TCP.ECHO.SERVER** (Function) 266  
**TCP.ECHOTEST** (Function) 266  
**TCP.FAUCET** (Function) 267  
**TCP.INIT** (Function) 265  
**TCP.OPEN** (Function) 264  
**TCP.OTHER.STREAM** (Function) 265  
**TCP.SINK.SERVER** (Function) 266  
**TCP.STOP** (Function) 265  
**TCP.URGENT.EVENT** (Function) 265  
**TCP.URGENT.MARK** (Function) 265  
**TCP.URGENTP** (Function) 265  
**TCPCHAT.TELNET.TTY TYPES** (Variable) 269  
**TCPCHAT.TRACEFILE** (Variable) 269  
**TCPCHAT.TRACEFLG** (Variable) 269  
**TCPFTP.DEFAULT.FILETYPES** (Variable) 267  
**TCPFTP.EOL.CONVENTION** 267  
**TCPFTP.INIT** (Function) 268  
**TCPFTP.SERVER** (Function) 268  
**TCPFTP.SERVER.USE.TOPS20.SYNTAX** (Variable) 268  
**TCPTTRACE** (Function) 266  
**TEdit** 9,21,139,287,307  
**TEdit Chat** 28  
**TeleRaid** 279  
**TELERAID** (Function) 280  
**TeleRaid Commands** 280  
**Teletype editor** 51  
**TELNET** protocol 261,269  
**Ten-key calculator pad** 215  
**Terminal emulators** 21  
**Testing the connection between two Xerox Lisp machines** 228

**TESTRELATION** (Function) 176  
**TExec** 287  
**TEEXEC** (Function) 287  
**TEXTMODULES** 289  
**TFTP.CLOSEFILE** (Function) 272  
**TFTP.GET** (Function) 272  
**TFTP.OPENFILE** (Function) 272  
**TFTP.PUT** (Function) 271  
**TFTP.SERVER** (Function) 272  
**Thumbing** 74  
**TIMEZONECOMP** (Variable) 200  
**Tracing and test Functions** 265  
**Trailer encapsulation** 277  
**Transferring files** 143,258  
**TRANSLATE-3-BY-3** (Function) 195  
**TRANSLATE-4-BY-4** (Function) 195  
**Translating between the file-naming conventions** 268  
**Transport layer** 260  
**Transport control protocol** 264  
**Trees** 111  
**TRIM.BITMAP** (Function) 66  
**Trivial file transfer protocol** 271  
**Troubleshooting Problems With FileBrowser** 94  
**TTY port** 1,219  
**TTY.DEFAULT.INIT.INFO** (Variable) 226  
**TTY.GET.PARAMETERS** (Function) 228  
**TTY.INIT** (Function) 226  
**TTY.SET.PARAMETERS** (Function) 227  
**TTY.SHUTDOWN** (Function) 228  
**TTYCHAT** 228  
**Typing Characters to DEdit** 53

## U

**UDP.APPEND.BYTE** (Function) 271  
**UDP.APPEND.CELL** (Function) 271  
**UDP.APPEND.STRING** (Function) 271  
**UDP.APPEND.WORD** (Function) 271  
**UDP.CLOSE.SOCKET** (Function) 270  
**UDP.EXCHANGE** (Function) 271  
**UDP.GET** (Function) 270  
**UDP.INIT** (Function) 270  
**UDP.OPEN.SOCKET** (Function) 270  
**UDP.SEND** (Function) 271  
**UDP.SETUP** (Function) 270  
**UDP.SOCKET.EVENT** (Function) 270  
**UDP.SOCKET.NUMBER** (Function) 270  
**UDP.STOP** (Function) 270  
**UNCOLORIZEBITMAP** (Function) 44  
**UNDELETE** (Command) 77  
**Unechoed input mode** 288  
**UNIX** 232,272  
**UNIX-STREAM-CLOSE** (Function) 305  
**UNIXChat** 303  
**UNIXComm** 305  
**Unread** 54  
**UNSAVEFNS** (Function) 177  
**UPDATECHANGED** (Function) 177  
**UPDATEFN** (Function) 177  
**Updating the MasterScope Data Base** 174  
**upgrade-semicolon-comments** (Variable) 292  
**User Commands for NSMaintain** 204  
**User datagram protocol** 270  
**USERWORDS** (Function) 288  
**Using**  
     4045XLP stream 8  
     4045 as a default printing host 3

FileBrowser window 73  
 Keyboard editor 150  
 modems 223  
 RS232 streams 222  
 TExec 288  
 TTY port 226  
 TTY streams 228

## V

Verifying TCP connections 258  
 Version Control 46  
**VFIND.PACKAGE** (Function) 286  
**VFIND.SYMBOL** (Function) 286  
**VGETBASE0** (Function) 285  
**VGETDEFN** (Function) 285  
**VGETPROPLIST** (Function) 285  
**VGETTOPVAL** (Function) 285  
**VGETVAL** (Function) 285  
 Viewing  
   an existing bitmap 64  
   frames from a stack 281  
   system stack 282  
 Violation of the IP standard 262  
 Virtual graph nodes 113  
 Virtual memory 279  
 Virtual terminal I/O 269  
 VirtualKeyboards 149,307  
**VKBD.CONFIGURATIONS** (Variable) 154  
**VKBD.KNOWN-KEYBOARDS** (Variable) 311  
**VLOADFNS** (Function) 283  
**VLOADVAR** (Function) 283  
**VPUTBASE0** (Function) 286  
**VRAID** (Function) 284  
**VSAVEWORK** (Function) 283  
**VSETTOPVAL** (Function) 285  
 VT100 Chat 28  
**VTYPENAME** (Function) 285  
**VUNSAVEDEF** (Function) 283  
**VVAG2** (Function) 285  
**VYANKDEF** (Function) 283  
**VICOPY** (Function) 285  
**VIUNCOPY** (Function) 285

## W

Weitek floating-point chip set 29  
 When To Copy 46  
 Where-Is 313  
 Wild cards 72  
**WINDOWPROP** (Function) 42  
**WORPCURSOR** (Function) 43  
 Wrapper Functions 193

## X

XCVR interface cable 253  
 Xerox 2700-II laser printer 1  
 Xerox 4045 Laser CP 1  
 Xerox Character Codes 19  
 Xerox/Diablo 630 1  
**XIP** (record) 69  
 XNS host number 200  
 XNS Time Service 199

## (

**(GETSCREENREGION** (Function) 40  
**(MAKE-HOMOGENEOUS-4-BY-4** (Function) 194

## 4

4045 configuration cartridge 2  
 4045 Emulation Mode Selection 2  
 4045 Fonts 9  
 4045 Parameter Names and Values 5  
 4045 Port Initialization 3  
 4045 Port Selection 2  
 4045 PROM and Software Compatibility 1  
**4045XLP.CHANGE.MODE** (Function) 5  
**4045XLP.DEFAULTS** (Variable) 5  
**4045XLP.GET.PARAMETERS** (Function) 7  
**4045XLP.PARAMETERS** (record) 5  
**4045XLP.SET.PARAMETERS** (Function) 6  
 4045XLPStream 1  
 4045XLPstream Options 5  
**4045XLPSTREAM.VERSION** (Variable) 1

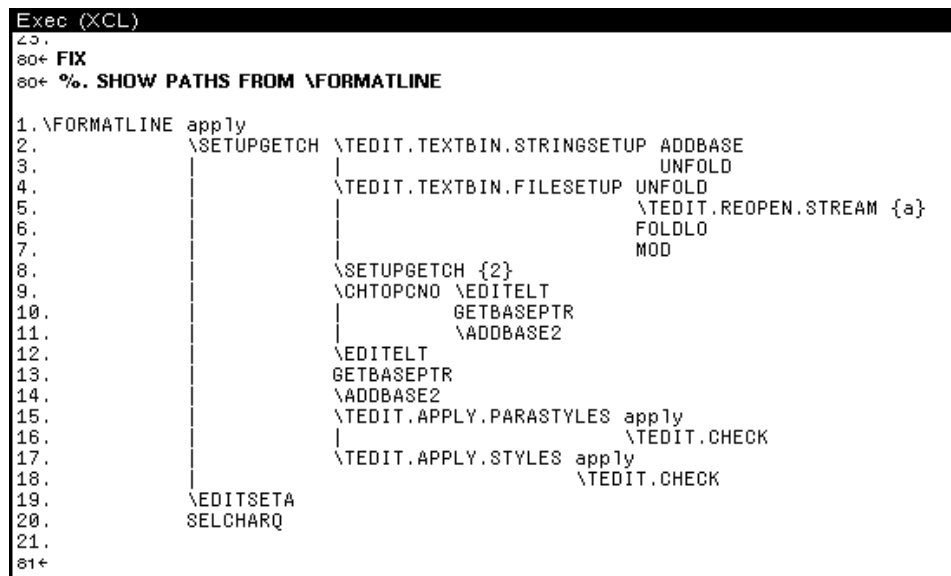


# BROWSER

4/26/2023 Larry Msinter: added BROWSERMAX to limit depth of tree, and made left-button on a node on the browser graph offer all formatting / control.

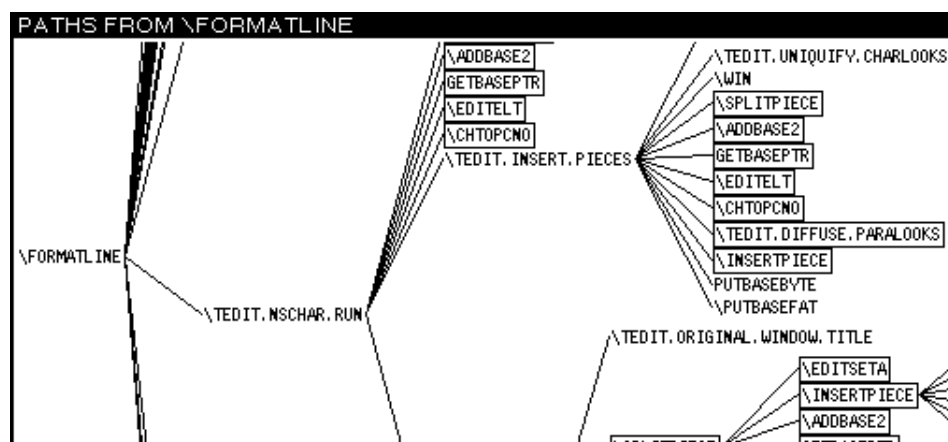
Browser modifies the SHOW PATHS command of MasterScope so that the output of the command is displayed as an undirected graph.

Browser makes it easier to use the MasterScope SHOW PATHS command by making it easier to read the result. This is how SHOW PATHS looks before Browser is loaded:



This shows that FORMATLINE calls SETUPGETCH, which calls EDITELT, etc.

And this is how SHOW PATHS looks after Browser is loaded:



This shows that FORMATLINE calls TEDIT.NSCHAR.RUN, which calls EDITELT, etc. (on another part of the calling tree).

You can reshape or scroll the window to see more of the result.

## Requirements

---

MASTERSCOPE, GRAPHER

## Installation

---

Load `BROWSER.LCOM` and the other required modules from the library.

## User Interface

---

Browser creates a new window for each `SHOW PATHS` command, but will reuse a window if that window has an earlier instance of the same `SHOW PATHS` command displayed in it.

The windows can be reshaped and scrolled with the normal window menu commands (which pop up when the right button is pressed in the window title bar).

The windows are active in the sense that nodes in the graph (i.e., functions) can be selected using the mouse. Clicking with the left button over the name of a function causes a menu to appear with menu items: `Display` will pretty-printed in the Browser printout window. `Describe` will describes the function, using the MasterScope `DESCRIBE` command, in the Browser describe window. The `Edit` menu item will calls the editor on that function. There are two menu items that will show you a new graph rooted at the function selected: `CallsFrom` and `CallsTo`.

The Browser graph is not updated automatically when you edit a function; you must give the `SHOW PATHS` command again to see the changes.

## Browser Variables

---

BROWSERFORMAT [Variable]

Controls the layout of the graph. It is set initially to display the graph horizontally, with the root to the left, and links running left to right.

**Example:**

`BROWSERFORMAT= (VERTICAL REVERSE)`

Lays out the graph with the root at the bottom and links running from bottom to top. For more details, see the description of the `LAYOUTGRAPH` function in the `GRAPHER Library Module`.

BROWSERMAX [Variable]

Maxumum depth of the graph. Initially 6.

BROWSERBOXING [Variable]

Controls how duplicate nodes in the graph are displayed. Its initial setting boxes functions that occur more than once in the graph.

**Example:**

BROWSERBOXING = ( (MARK BORDER 5) COPIES/ONLY)

Boxes copies of the function with a border of 5. For more details, see the description of the LAYOUTGRAPH function in the GRAPHER Library Module.

## Functions

---

`(BROWSER T)` turns the Browser on.

The Browser calls `LAYOUTFOREST` (in `Grapher`) to generate a graph showing the calling hierarchy.

`SHOWGRAPH` displays the graph.

The Browser modification to `MasterScope` can be undone by calling `(BROWSER NIL)`, restoring the teletype-oriented output of `SHOW PATHS`.

## Limitations

---

Browser works only with `MasterScope`.

## Examples

---

Type the following into an Interlisp Exec window:

```
.ANALYZE ANY ON MY-MODULE
```

```
.SHOW PATHS FROM MY-FUNCTION
```

[This page intentionally left blank]

# CASH-FILE

---

Cash-File is a front end to Hash-File which uses a hash table to cache accesses to hash files. This can provide a significant performance improvement in applications which access a small number of keys repeatedly. For example, the Where-Is library module uses this module to achieve acceptable interactive performance.

Cash-File is similar to but not compatible with the LispUsers' module, HASHBUFFER.

All of the code for Cash-File is in a package called Cash-File. Throughout this document Lisp symbols are printed as though in a package which uses the packages Cash-File, Hash-File, and Lisp.

## Installation

---

Load CASH-FILE.DFASL and HASH-FILE.DFASL from the library.

## Functions

---

The functional interface is designed to closely resemble that of Hash-File, which was in turn designed to resemble the Common Lisp hash table facility.

(make-cash-file *file-name* *size* *cache-size*) [Function]

Creates and returns an empty cash file in *file-name*. *Size* is passed as the *size* argument to make-hash-file, while *cache-size* is passed as the *size* argument to make-hash-table and determines the maximum number of entries to be cached.

(get-cash-file *key* *cash-file* &optional *default*) [Function]

Just like get-hash-file and gethash. Retrieves the value stored under *key* in *cash-file* or *default* if there is none. Also returns a second value which is true if a value was found for *key*.

A setf method is also defined for get-cash-file.

(open-cash-file *file-name* *cache-size* &key *direction*) [Function]

Open the existing hash file in *file-name* in *direction* (:input or :io). *Cache-size* is passed as the *size* argument to make-hash-table and determines the maximum number of entries which will ever be cached.

(rem-cash-file *key* *cash-file*) [Function]

Like rem-hash-file and remhash. Deletes key from the hash file and the cache. Returns true if and only if there was a value stored under *key*.

(cash-file-p *object*) [Function]

Returns true if and only if *object* is a cash file.

(cash-file-p *object*) ≡ (typep *object* 'cash-file)

(cash-file-hash-file *cash-file*)

[Function]

Returns the hash file object to which *cash-file* is a front end.

There are no cash file specific equivalents for `close-hash-file`, `map-hash-file` and `hash-file-count`. For these use the hash file functions on the `cash-file-hash-file`.

---

## Implementation Notes

A queue is maintained to enable cache deletion when the cache is full. This queue is implemented as a list. Each time a key is accessed, it is moved to the head of the queue. The last element of the queue is deleted when a new key is accessed and the queue is full.

---

## Limitations

The cache time is not constant but grows linearly with the size of the cache. For this reason, huge caches are not recommended.

[This page intentionally left blank]



# CENTRONICS

---

The Centronics module implements a stream interface to an industry-standard Centronics printer port. This port is designed to drive Centronics-compatible devices, typically printers. The module allows you to send bytes over the parallel port, and notifies you of any device error conditions.

## Requirements

---

The Centronics port is found on the Xerox 1109, which is an 1108 equipped with the Extended Processor board (marked CPE FP). It is the upper of the two connectors on the board.

The Centronics cable from the port to the printer should be wired as shown in the Introduction of this manual.

CENTRONICS.LCOM implements a general byte output stream. It is typically used in conjunction with a printer driver module, such as 4045XLPStream, though it can also run by itself.

## Installation

---

Load CENTRONICS.LCOM from the library.

## User Interface

---

### Functions

(CENTRONICS.RESET)

[Function]

The only user-callable function in the module, it initializes the parallel port and any attached device. It should be called after the printer is powered on.

### Opening a Centronics Stream

To open a stream to the Centronics port, evaluate a form similar to the following:

```
(SETQ CENTRONICS.STREAM (OPENSTREAM '{CENTRONICS} 'OUTPUT))
```

All bytes BOUTed to CENTRONICS.STREAM are sent to the attached printer. You may only have one stream open to the parallel port at one time; attempts to open others yield an error.

### Device Errors

When a device error is detected (e.g., printer offline, out of paper, etc.), a break window will pop up.

After resetting the device, type **RETURN** (the word, not the key) to continue. Type **STOP** to abort.

## **Limitations**

---

The port is available on Xerox 1109 workstations only.

[This page intentionally left blank]

# CHARCODETABLES

---

CharCodeTables prints character code charts for a given font, showing the characters that exist in a printer in that font. It is useful for illustrating the characters that are available to the user of an application.

Each table is a grid, specific to a font and to the printer to which the output is sent. Across the top are the high-order 8 bits of the character code; down the left are the low-order 8 bits. At 255 of the 256 intersections, a character is printed (code 377 is reserved).

If a particular character doesn't exist on the printer, a black rectangle is shown in its stead.

The tables are printed two to a page, in landscape form. To avoid problems with printer limitations, a group of charts is broken into documents no longer than five pages each for actual printing.

## Xerox Character Codes

---

The Xerox Character Code Standard specifies a 16-bit character code space containing all the characters in a given font.

For example, there are over 60,000 possible characters in the font Modern 10 Bold; however, many of those character codes have not yet been assigned. Moreover, a given printer may not have all the characters for a given font that the standard calls for.

The character code space is divided into 255 character sets (numbered 0–376 octal) of 255 characters each (codes 0–377 octal). Character code 377 is reserved as the character-set switching marker in the Xerox file representation of the characters, thereby rendering character set 377 unavailable as well.

Generally, each character set or range of character sets is reserved for a particular use or language. Character set 0, for example, contains the ASCII characters in its lower half, and a variety of common international and commercial symbols in its upper half (corresponding to the 8-bit ISO 6937 standard). Character set 46 is reserved for the Greek alphabet and Greek-specific punctuation marks.

Code assignments are described in detail in Xerox System Integration Standard *Character Code Standard*, XNSS 058605, May 1986, version XC1-2-2-0.

## Requirements

---

The variable `INTERPRESSFONTDIRECTORIES` must be set to a list of directories which contain font metric files. These files have names of the form

`{ERIS}<LISP>FONTS>MODERN08-BIR-C#.WD`

where B = bold, I=italic, and # represents the character set number in octal.

## Installation

---

Load `CHARCODETABLES.LCOM` from the library.

## Functions

---

(SHOWCSETLIST *CSETS FONT*) [Function]

Prints code tables for the character sets in the list *CSETS*, for the given font specification *FONT*.

*CSETS* is a list of one or more numbers that identify the character sets; *FONT* is the name of a font.

(SHOWCSETRANGE *FIRSTCSET LASTCSET FONT*) [Function]

Prints code tables for the character sets from *FIRSTCSET* through *LASTCSET* for the given *FONT*.

(SHOWCOMMONCSETS *FONT*) [Function]

Prints code tables for the most common character sets in the given *FONT*.

(These are character set 0, Greek, Cyrillic, Katakana, Hiragana, and various special symbols)

(SHOWCSET *FONT*) [Function]

Prints code tables for every character set defined in the Xerox Character Code Standard.

## Limitations

---

This module works only with InterPress fonts.

## Examples

---

```
(SHOWCSETLIST ' (0 238 239) ' (MODERN10))
```

or

```
(SHOWCSETLIST ' (0 #0356 #0357) ' (MODERN10))
```

Prints the code tables for character sets 0, 356 and 357 (octal) that correspond to the Modern 10-point font.

```
(SHOWCSETRANGE 24 26 ' (MODERN 10 ITALIC))
```

or

```
(SHOWCSETRANGE #030 #032 ' (MODERN 10 ITALIC))
```

Prints the code tables for character sets 30 and 32 (octal) and the Modern 10 italic font.

**Note:** When typing the character set number, remember that the character sets are identified by octal numbers. Therefore you must type either the decimal equivalent of that octal number (e.g., to represent 41 octal, type 33) or the octal number directly, typed as #041 (where O is the letter O).

[This page intentionally left blank]

# CHAT

---

Chat is a remote terminal facility that allows you to communicate with other machines while inside Lisp. Chat sets up a Chat connection to a remote machine, so that everything you type is sent to the remote machine, and everything the remote machine prints is displayed in a Chat window.

Chat is an extensible terminal emulation facility. Its core supplies both terminal- and network-protocol- independent functionality; new terminal types and new Chat protocols, based on this core, can be added to Chat at any time. You can choose any terminal type to be used with any network protocol type.

There are currently terminal emulators for the following terminals:

Datamedia 2500

DEC VT100

TEdit (this is actually a TEdit-based Chat window, supporting scrolling and copy-select operations as in standard TEdit).

A number of different network protocol interfaces can be used with Chat. The following protocols are available:

PUP Chat

NS Chat (using the GAP protocol)

TCP (ARPANET) TELNET

RS232 Chat (using either the RS232 or TTY ports of the 1108 and 1186 processors)

Each of these is available by loading the corresponding module.

## Requirements

---

DMCHAT

CHATTERMINAL

One of the network protocols: PUPCHAT or NSCHAT or RS232CHAT or TTYCHAT or TCPCHAT.

One of the terminal emulators: DMCHAT or VTCHAT or TEDITCHAT.

The applicable file dependencies enumerated in the Introduction of this manual.

## Installation

---

Load CHAT.LCOM from the library.

In addition, you must load at least one of the Chat network protocol modules.

If you want a terminal emulator different from the default DM2500, you must also load it.

## User Interface

---

Chat prompts for a new window for each new connection. It saves the first window to reuse once the connection in that window is closed (other windows just go away when their connections are closed).

Multiple, simultaneous Chat connections are possible. To switch between typing to different Chat connections, press the left button within the Chat window you want to use.

### Opening a Chat Connection

The simplest way to open a Chat connection is to select the CHAT option of the right-button (background) menu. The first time you do this, you are prompted in the system's prompt window for the name of a host to which to connect. Subsequently, you are prompted with a menu of all hosts to which you have opened Chat connections; the last entry in this menu is OTHER, and provides a way for you to connect to new Chat hosts.

The other method of opening a Chat connection is to call the CHAT function directly:

(CHAT *HOST* *LOGOPTION* *INITSTREAM* *WINDOW*) [Function]

Opens a Chat connection to *HOST*, or to the value of DEFAULTCHATHOST. If *HOST* requires login, Chat supplies a login sequence.

You may alternatively specify one of the following values for *LOGOPTION*:

    Login   Always perform a login.

    Attach   Always perform an attach (this is likely to be useful only when opening Chat connections to hosts running the Tops-20 or Tenex operating systems). This fails if you do not have exactly one detached job.

    None   Do not attempt to log in or attach.

**Note:** It is important that you supply information about the types of hosts to which you chat by setting the variable NETWORKOSTYPES (see IRM) or DEFAULT.OSTYPE (see *Lisp Release Notes*), as CHAT uses that information to determine whether and how to log in. An incorrect login sequence can inadvertently expose your password.

If *INITSTREAM* is supplied, it is either a string or the name of a file whose contents are read as type-in. When the string/file is exhausted, input is taken from the keyboard.

If *WINDOW* is supplied, it is the window to use for the connection; otherwise, you are prompted for a window.

While Chat is in control, all Lisp interrupts are turned off, so that control characters can be transmitted to the remote host. Chat does not turn off interrupt characters until after creating the Chat window, so you can abort the call to Chat by typing Control-E while specifying the Chat window region.

If you press the left button in an Executive window, the system's focus-of-attention is switched to that window. At the same time, keyboard interrupts, such as Control-E, are reenabled. Whenever you select an open Chat window, the focus-of-attention is returned to the Chat window, and keyboard interrupts are disabled.



## Chat Menu

Commands can be given to an active Chat connection by pressing the middle mouse button in the Chat window to get a command menu.

**Note:** The left mouse button, when pressed inside an active Chat window, holds output as long as the button is down. Holding down the middle button coincidentally does this too, but not on purpose; since the menu handler does not yield control to other processes, it is possible to kill the connection by keeping the menu up too long.

|           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CLOSE     | Closes this connection. Once the connection is closed, control is handed over to the main Lisp Executive window. Closes the Chat window unless it is the primary Chat window.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| SUSPEND   | Same as CLOSE, but always leaves the window open.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| NEW       | Closes the current connection and prompts for a new host to which to open a connection in the same window.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| FREEZE    | Holds type-out from this Chat window. Pressing a mouse button in the window in any way releases the hold. This is most useful if you want to switch to another, overlapping window and there is type-out in this window that would compete for screen space.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| DRIBBLE   | Opens a typescript file for this Chat connection (closing any previous dribble file for the window). You are prompted for a file name. If you want to close an open dribble file (without opening a new one), just type a carriage return.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| INPUT     | Prompts for a file from which to take input. When the end of the file is reached, input reverts to the keyboard.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| CLEAR     | Clears the window and resets the simulated terminal to its default state. This is useful if undesired terminal commands have been received from the remote host that place the simulated terminal into an indeterminate state.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| EMACS     | Turns on or off the Chat EMACS feature, which provides a convenient way to use the workstation's mouse to move the cursor on the remote machine when using the EMACS text editor. When this feature is turned on, pressing the left mouse button in the Chat window causes a sequence of commands to be sent to the remote machine that cause EMACS to move its cursor to the mouse location.<br><br>Use of this feature assumes you know the keystrokes to perform cursor-moving commands; see <code>CHAT.EMACSCOMMANDS</code> if your EMACS does not use the standard ones. Also, it assumes that you are pointing where there is actually text in your document (not white space beyond the end of a line) and that there are no tabs in your text; otherwise, the cursor position may not be where you expect. |
| RECONNECT | In an inactive Chat window, pressing the middle mouse button brings up a menu of one item, RECONNECT, whose selection reopens a connection to the same host as was last in the window. This is the primary motivation for the SUSPEND menu command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

`<EMULATOR>MODE` The Chat menu also contains a command of this form for each terminal emulator that you have loaded. The `<EMULATOR>MODE` commands are intended to let you dynamically switch between terminal emulators.

However, this feature is currently defective and should not be used. You must also choose your emulator type, by setting `CHAT.DISPLAYTYPES`, before opening the Chat connection.

---

## Customizing Chat

---

`CHAT.DISPLAYTYPES` [Variable]

This variable contains a list that assigns the terminal emulators to be used with the hosts. Each entry on the list is of the form:

`(<HostName><TerminalTypeNumber><TerminalEmulator>)`

`HostName` When Chat opens a connection, it scans `CHAT.DISPLAYTYPES` to find an entry whose `HostName` field matches the name of the Chat host. If no matching entry is found, it scans the list again, looking for an entry whose `HostName` field is `NIL`.

`TerminalTypeNumber` Is only important when the Chat protocol in use is PUP Chat. This number identifies the terminal type to the Chat host's operating system. Currently, only Tops-20 and Tenex hosts make use of this facility; if the Chat host does not support this feature, the number in the `TerminalTypeNumber` field is ignored.

`TerminalEmulator` Chat uses this field of the entry it finds to choose which terminal type to emulate. Typical terminal emulator names are `DM2500`, `VT100`, and `TEDIT`.

`CHAT.KEYACTIONS` [Variable]

This variable controls the remapping of the keyboard when the system's focus-of-attention is an active Chat window. The format of this list is:

`((KEYNAME . ACTIONS) (KEYNAME . ACTIONS) ... )`

For example, if you prefer the backspace key to send the rubout character (octal 177), you would set `CHAT.KEYACTIONS` to be:

`((BS (177Q 177Q NOLOCKSHIFT) . IGNORE))`

The key actions are assigned when a Chat process is initiated; i.e., changing `CHAT.KEYACTIONS` only affects new Chat connections.

`CHAT.INTERRUPTS` [Variable]

A list of interrupts to pass to `INTERRUPTCHAR` to assign keyboard interrupts; e.g., `((177Q. HELP))` causes the `DELETE` character (code 177) to run the `HELP` interrupt.

Like `CHAT.KEYACTIONS`, this variable only affects new Chat connections.

CHAT.ALLHOSTS [Variable]

A list of host names, as uppercase symbols, to which you want to chat. Chatting to a host not on the list adds it to the list. These names are placed in the menu used by the background Chat command prompts.

CLOSECHATWINDOWFLG [Variable]

If true, every Chat window is closed on exit. If NIL, the initial setting, then the primary Chat window is not closed.

DEFAULTCHATHOST [Variable]

The host to which CHAT connects when it is called with no HOST argument.

CHAT.FONT [Variable]

If non-NIL, the font used to create Chat windows. If CHAT.FONT is NIL, Chat windows are created with (DEFAULTFONT 'DISPLAY).

Note: To work well with the DM2500 and VT100 terminal emulators, you should use fixed-width fonts (e.g., Gacha or Terminal).

CHAT.WINDOW.SIZE [Variable]

This variable is either NIL or a dotted pair of (WIDTH . HEIGHT). The value of the WIDTH field indicates the desired width of the Chat window, in pixels. The value of the HEIGHT field indicates the desired HEIGHT of the window, also in pixels.

Note: Before a new value of CHAT.WINDOW.SIZE is used, CHAT.WINDOW must be set to NIL or NOBIND. If CHAT.WINDOW.SIZE is changed after chat has already been called, and chat is then called, the window is not changed because the information is cached. CHAT.WINDOW must be set to NIL and the window recreated anew before this takes place.

CHAT.WINDOW.REGION [Variable]

This variable is either NIL or an instance of a REGION. When CHAT.WINDOW.REGION is non-NIL, its value is used as the region in which to create the first Chat window.

Subsequent windows are created by prompting for the position of a window of CHAT.WINDOW.SIZE dimensions, or, if that variable is NIL, for an arbitrary window region.

CHAT.TTY.PROCESS [Variable]

When you start up CHAT, it takes the TTY immediately if the value is T. (The initial value is T.)

CHAT.EMACSCOMMANDS [Variable]

A list of five character codes; initially the value of (CHARCODE (^U ^P ^N ^F ^A)). These character codes are used by the EMACS Argument command in changing the position of the cursor:

- Up one line
- Down one line
- Forward one character

Backward one character

Beginning of line

CHAT.IN.EMACS?

[Variable]

The initial state of the EMACS feature when a Chat connection is started.  
Initially NIL, meaning the feature is off.

CHAT.PROTOCOLTYPES

[Variable]

Each Chat emulator (TTYCHAT, RS232CHAT, PUPCHAT ...) adds an entry onto CHAT.PROTOCOLTYPES which recognizes host names for the appropriate protocol.

For example, loading PUPCHAT adds an entry (PUP . PUPCHAT.HOST.FILTER) and TCPCHAT adds an entry (TCP . TCP.HOST.FILTER).

Site administrators of complex networks may want to reorganize these entries when there are hosts which are running multiple servers, each running different protocols.

---

## Network Protocols

For the most part, you should not notice too many differences in the behavior of Chat when using one network protocol versus another. The following are unique features of each of the Chat network protocols.

### PUP Chat

PUP Chat is in the file PUPCHAT.LCOM. Implementations of PUP Chat servers exist for Tops-20, Tenex, VAX/UNIX, and VAX/VMS operating systems. The PUP Chat protocol contains provisions for automatically setting your terminal type, width, and height whenever you establish a connection or reshape your Chat window.

### NS Chat

The NS Chat protocol (also known as GAP, or Gateway Access Protocol) is used to communicate with hosts running GapTelnet service, including VAX/UNIX and the VAX/VMS service XNS/DEC VAX, and also with Xerox 8000-series network services such as 8040 print servers or 8030 file servers. This protocol is contained on the file NSCHAT.LCOM. The NS Chat protocol differentiates among a number of virtual terminal services. When you chat to an NS host, the NS Chat module queries the Clearinghouse for information about the specified host. This information permits the NS Chat module to determine which of the following virtual terminal services are appropriate for the host.

The NS Chat module uses a small set of heuristics to choose which virtual terminal service to invoke, based on information returned by the Clearinghouse. If the Clearinghouse information indicates that only one service type is possible, NS Chat opens a connection to the Chat host and invokes the proper virtual terminal service.

If the Clearinghouse returns information indicating that more than one virtual terminal service is supported by the specified host, you are prompted to choose a service from a menu of the possible service types.

If NS Chat guesses an incorrect service type, or you choose an incorrect service type, you are prompted to choose a service from a menu of all known virtual service types. If this fails, NS Chat abandons its attempts to connect to the specified host.

## Remote System Administration

This service lets you log onto print servers and clearinghouse servers, and issue appropriate commands. NS Chat automatically chooses this service when the specified host is registered in the Clearinghouse as any type of server machine.

## Remote System Executive

This service is currently supported by VAX/VMS systems running XNS/DEC VAX, by UNIX systems running GapTelnet service, by Lisp workstations running CHATSERVER from the library, and by XDE workstations.

## Interactive Terminal Service

The ITS is a TTY-based interface to NS mail.

## External Communication Service

The External Communication Service (ECS) enables Chat connections to external hosts accessible only by use of a modem. When you open a Chat connection to an ECS, you are prompted for a telephone number; the ECS dials that number and completes the connection if a compatible modem answers.

ECS hosts typically support a variety of modem connection characteristics (specific combinations of parity, character length, baud rate, and flow control settings). Each connection type is known by a different Chat host name; check with your system administrator to determine the Chat host name you should use to connect to a particular external host.

## TCP Chat

TCPCHAT.LCOM is the interface to the TCP-based TELNET protocol, which is the protocol in use throughout the ARPANET. It loads and initializes the TCP-IP module, if necessary. Read the TCP-IP module in this manual for more information.

## RS232 Chat

RS232 Chat is contained on the files RS232CHAT.LCOM and TTYCHAT.LCOM. RS232 Chat enables use of the 1108, 1185, and 1186 RS232 ports; TTY Chat enables use of the 1108, 1185, and 1186 TTY ports. Read the RS232 module in this manual for more information.

---

## Terminal Emulators

### DM2500 Chat

The Datamedia 2500 terminal emulator is contained in DMCHAT.LCOM. To use it, load DMCHAT.LCOM and add entries to CHAT.DISPLAYTYPES in the form:

```
(<HOSTNAME> <TERMINALTYPENUMBER> DM2500)
```

## VT100 Chat

The VT100 emulator is contained in VTCHAT. To use it, load VTCHAT.DFASL and add entries to CHAT.DISPLAYTYPES in the form:

```
((<HOSTNAME> <TERMINALTYPENUMBER> VT100)
```

Currently, the VT100 emulator does not emulate the following features of the actual Digital VT100 terminal:

- Dual-width or dual-height characters

- Graphics character set

- Remotely initiated switching between 80- and 132-column mode

## TEdit Chat

TEdit Chat supplies a “glass TTY” terminal emulator with a TEdit stream storing all characters received during the Chat session. As a result, you can scroll back and forth through a transcript of your session, and you can use the standard TEdit copy-select command to copy blocks of characters from the Chat window to another TEdit window, a Lisp Executive, etc.

To use TEdit Chat, load TEDITCHAT.LCOM, and add entries to CHAT.DISPLAYTYPES in the form:

```
((<HOSTNAME> <TERMINALTYPENUMBER> TEDIT)
```

Note that since TEdit already uses the middle mouse button, you must click in the window's title bar in order to get the usual Chat menu.

[This page intentionally left blank]



---

---

## CLIPBOARD

---

---

Written by Ron Kaplan, 2020-2021

A small package that implements copy and paste to the system clipboard.

It arms meta-C for copy to the clipboard from the current selection of an application that has been armed (TEDIT, SEDIT), and also meta-X (TEDIT) for extraction (copy followed by delete). (SEdit currently assigns "expand" to meta-X).

Meta-V is defined as an interrupt character that pastes the current clipboard contents into whatever process currently has input focus.

The information in the clipboard can be provided from or to external (non-Medley) applications (mail, emacs, etc.) in the usual way. For example, a form selected in SEDIT can be copied to the clipboard and pasted into an Outlook or Mac Mail email message.

It assumes that the external format of the clipboard is determined by (SYSTEM-EXTERNALFORMAT), and characters will be converted to and from the Medley internal character encoding.

The name of the clipboard stream may differ from platform to platform. On the Mac, the paste stream is "pbpaste" and the copy stream is "pbcopy". Those names are used if "darwin" is a substring of (UNIX-GETENV "ostype"). Otherwise both stream-names default to "xclip". The functions CLIPBOARD-COPY-STREAM and CLIPBOARD-PASTE-STREAM perform this selection.

# CMLFLOATARRAY

---

CmlFloatArray implements high-speed floating-point vector and array operations. Although optimized for the case of arrays of element-type single-float, the array operations are generic and will operate on arrays of any element-type.

CmlFloatArray uses special purpose microcode that exploits the full capabilities of the Weitek floating-point chip set, available on 1109s, for doing arithmetic operations on floating-point arrays. On machines without the Weitek floating-point chip set, such as the 1186, these operations will still usually be more efficient than the corresponding scalar implementation.

The functions described here operate on Common Lisp arrays, and may be thought of as extensions of the general Common Lisp sequence functions.

## Requirements

---

1186 or 1109 (1108 with the extended processor option) and the Weitek floating-point chip set.

## Installation

---

Load CMLFLOATARRAY.LCOM from the library.

## Functions

---

(MAP-ARRAY *RESULT* *MAPFN* *ARRAY1* *ARRAY2* . . .  
*ARRAYN*)

[Function]

MAP-ARRAY is a general mapping function over arrays and scalars.

Arrays of dimension greater than one are treated as vectors in row-major order; that is, an array *A* with dimension (2 2) is treated as a vector of length 4 and elements '#(aref *a* 0 0), (aref *a* 0 1), (aref *a* 1 0), (aref *a* 1 1)). All array arguments must be conformable; that is, of the same dimensions.

Scalars (non-arrays) are extended to the common dimension of the other array arguments by copy-on (that is, a scalar is treated as a vector of the appropriate length, each of whose elements is the scalar).

For example, a call to MAP-ARRAY with two arrays of dimensions (4 4) and one scalar and the function MAX as map function will invoke MAX 16 times, with the scalar the third argument in each call.

If *RESULT* is NIL, the map is for effect only (i.e., no array result is returned; MAP-ARRAY is of interest only due to a side effect).

If *RESULT* is a valid element-type, an array of the appropriate dimensionality and element-type will be created to hold the map results.

If *RESULT* is an array, it must be conformable with the other array arguments and it will be side effected by the mapping operation.

*MAPFN* is an arbitrary n-ary Lisp function; i.e., it takes as many arguments as there are arrays passed to *MAP-ARRAY*. It is unary (takes one argument) if one array is passed to *MAP-ARRAY*, binary if two arrays are passed, etc.. In the case of unary or binary operations, *MAP-ARRAY* recognizes certain functions and executes the corresponding operation particularly efficiently.

If the single array argument is of element-type single-float and the result array is of element-type single-float, the following unary operations are recognized and executed in microcode:

|           |                                                                                                                                                                                                                                                    |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| - (MINUS) | Negates each element of the array argument.                                                                                                                                                                                                        |
| ABS       | Computes the absolute value of each element of the array argument.                                                                                                                                                                                 |
| TRUNCATE  | The single array argument must be of element-type single-float, but the result array may be any element-type which will accomodate the integer results. Truncates (converts to integer, rounding towards zero) each element of the array argument. |
| FLOAT     | The single array argument must be of element-type (unsigned-byte 16), and the result array may be of element-type single-float.<br><br>Converts each element of the array argument to a single precision floating point number.                    |

If both arguments are of element-type single-float and the result array is of element-type single-float, the following binary operations are recognized and executed in microcode.

|              |                                                                          |
|--------------|--------------------------------------------------------------------------|
| + (PLUS)     | Computes the element-wise (element by element) sum of the two arguments. |
| - (MINUS)    | Computes the element-wise difference of the two arguments.               |
| * (TIMES)    | Computes the element-wise product of the two arguments.                  |
| / (QUOTIENT) | Computes the element-wise quotient of the two arguments.                 |

(REDUCE-ARRAY *REDUCTION-FUNCTION* *ARRAY* &OPTIONAL  
*INITIAL-VALUE*)

[Function]

*REDUCE-ARRAY* is similar to the sequence function *REDUCE* but is generalized for arrays of arbitrary dimensionality; that is, the binary mapping function is applied to each element of the single array argument, each time being passed the result of the previous application as well as the current array element. Arrays of dimensionality greater than one are treated as vectors in row-major order. The result of *REDUCE-ARRAY* is always a scalar.

If *INITIAL-VALUE* is provided, it is used as the starting value of the reduction operation, otherwise the first element of *ARRAY* is the starting value. In the degenerate case of arrays of size zero or one, the use of *INITIAL-VALUE* parallels that of the sequence function *REDUCE*. *REDUCE-ARRAY* recognizes certain mapping functions and executes the corresponding operation particularly effeciently.

If the single array argument is of element-type single-float, the following reduction operations are recognized and Executed in microcode.

|                        |                                                      |
|------------------------|------------------------------------------------------|
| <code>+</code> (PLUS)  | Computes the sum of all the array elements           |
| <code>*</code> (TIMES) | Computes the product of all the array elements       |
| <code>MIN</code>       | Returns the smallest array element                   |
| <code>MAX</code>       | Returns the largest array element                    |
| <code>MIN-ABS</code>   | Returns the smallest array element in absolute value |
| <code>MAX-ABS</code>   | Returns the largest array element in absolute value  |

(EVALUATE-POLYNOMIAL *X* *COEFFICIENTS*) [Function]

This function calculates the value of a polynomial at the point *X*. The polynomial is described by a vector of coefficients, *COEFFICIENTS*, where *COEFFICIENTS*[0] corresponds to the coefficient of highest degree. If *COEFFICIENTS* is a vector of element-type single-float, then this operation is Executed in microcode.

(FIND-ARRAY-ELEMENT-INDEX *ELEMENT* *ARRAY*) [Function]

Returns the index of the first element of *ARRAY* that is EQL to *ELEMENT*, or NIL if there is no such element.

---

## Limitations

This version of CmlFloatArray does not support the FFT functionality of previous versions.

[This page intentionally left blank]



# COLOR

---

Color is the software for driving color displays. To run COLOR, you need either a Sun (3, 4 or SPARCstations) with CG4, CG6 color hardware and color display.

The machine independent color graphics code is stored in the library files `LLCOLOR.LCOM` and `COLOR.LCOM`. The Sun color driver resides in the file `MAIKOCOLOR.LCOM` which loads `LLCOLOR.LCOM`, `COLOR.LCOM`, and `TAKEBIGBM.LCOM`.

The CG4,CG6 device supports 8 bpp (bits per pixel) at a resolution of 1152 by 900.

Note: You cannot use COLOR if you are running Medley under X.

## Software Screens

---

To support the various hardware configurations and external display devices, the software has a special datatype, a "screen". A screen represents and controls a physical hardware display. It contains windows and icons, and tracks the mouse. There are two distinct types of screens: a black and white screen, and a color screen.

On workstations with a single hardware display, the display is shared by both the black and white screen and the color screen. It can be changed by moving the mouse cursor. The screen mode may be changed by moving the cursor out of the screen.

## Turning the Color Display Software On and Off

---

The color display software can be turned on and off. While the color display software is on, and a small amount of processing time is used to drive the color display.

(`COLORDISPLAYP`) [Function]

Returns T if the color display is on; otherwise it returns NIL.

(`COLORDISPLAYONOFF TYPE`) [Function]

Turns off the color display if *ONOFF* is set to OFF. If *ONOFF* is set to ON, it turns on the color-display-allocating memory for the color screen bitmap. *TYPE* should be `MAIKOCOLOR`, if you are using a Sun Workstation. The usual sequence of events is to `LOAD MAIKOCOLOR.LCOM` to drive GC4/GC6 color card and then to call `COLORDISPLAY` with the appropriate *TYPE* (`MAIKOCOLOR` for Sun Workstations) to turn the software on. For example,

```
(LOAD 'MAIKOCOLOR.LCOM)
(COLORDISPLAY 'ON 'MAIKOCOLOR)
```

Calling `COLORDISPLAY` allocates memory for the color screen bitmap in the `sysout`. Turning on the color display requires allocating the memory necessary to hold the color display screen bitmap. Turning off the color display does not free this memory; it still exists in the `sysout`.

## Colors

---

The number of bits per pixel (bpp) determines the number of different colors that can be displayed at one time. When there are 8 bpp, 256 colors can be displayed at once. A table, called a *color map*, determines what color actually appears for each pixel value. A color map gives the color in terms of how much of the three primary colors (red, green, and blue) is displayed on the screen for each possible pixel value.

A color can be represented as a number, an atom, or a triple of numbers. The color map provides the final interpretation of how much red, blue, and green a color represents. A color map maps a color number ( $[0 \dots 2^{n_{\text{bits}}}-1]$ ) into the intensities of the three color guns (primary colors red, green, and blue). Each entry in the color map has 8 bits for each of the primary colors, allowing 256 levels per primary or  $2^{24}$  possible colors (not all of which are distinct to the human eye). Within Xerox Lisp programs, colors can be manipulated as numbers, red-green-blue triples, names, or hue-lightness-saturation triples. Any function that takes a color accepts any of these different representations.

If a number is given, it is the color number used in the operation. It must be valid for the color bitmap used in the operation. (Since all of the routines that use a color need to determine the color's number, it is fastest to use numbers for colors. `COLORNUMBERP`, described below, provides a way to translate into numbers from the other representations.)

The following sections describe other ways to represent colors.

### Red-Green-Blue Triples

A red-green-blue (RGB) triple is a list of three numbers, each of which can be between 0 and 255. The nine digits are divided as follows:

- First 3 digits = red intensity
- Second 3 digits = green intensity
- Third 3 digits = blue intensity

When an RGB triple is used, the current color map is searched to find the color with the correct intensities. If none is found, an error is generated. (That is, no attempt is made by the system to assign color numbers to intensities automatically.) An example of an RGB triple is (255 255 255), which gives the color white.

RGB

[Record]

A record defined as (RED GREEN BLUE); it can be used to manipulate RGB triples.

COLORNAMES

[Association list]

Maps names into colors. The CDR of the color name's entry is used as the color corresponding to the color name. This can be any of the other representations.

Note: It can even be another color name. The system does not check for loops in the name space, such as would be caused by putting ' (RED . CRIMSON) and ' (CRIMSON . RED) on COLORNAMES.

Some color names are available in the initial system and allow color programs written by different users to coexist. These are:



| Name    | RGB           | Number in default color maps |     |
|---------|---------------|------------------------------|-----|
| BLACK   | (0 0 0)       | 15                           | 255 |
| BLUE    | (0 0 255)     | 14                           | 252 |
| GREEN   | (0 255 0)     | 13                           | 227 |
| CYAN    | (0 255 255)   | 12                           | 224 |
| RED     | (255 0 0)     | 3                            | 31  |
| MAGENTA | (255 0 255)   | 2                            | 28  |
| YELLOW  | (255 255 0)   | 1                            | 3   |
| WHITE   | (255 255 255) | 0                            | 0   |

## Hue–Lightness–Saturation Triples

A hue–lightness–saturation triple is a list of three numbers:

- The first number (HUE) is an integer between 0 and 355. It indicates a position in degrees on a color wheel (blue at 0, red at 120, and green at 240).
- The second (LIGHTNESS) is a real number between zero. It indicates how much total intensity is in the color.
- The third (SATURATION) is a real number between zero and one. It indicates how disparate the three primary levels are.

HLS

[Record]

A record defined as (HUE LIGHTNESS SATURATION); it is provided to help you manipulate HLS triples.

Example: the color blue is represented in HLS notation by (0 .5 1.0).

(COLORNUMBERP *COLOR* *BITSPERPIXEL* *NOERRFLG*)

[Function]

Returns the color number (offset into the screen color map) of *COLOR*. *COLOR* can be one of the following:

- A positive number less than the maximum number of colors
- A color name
- An RGB triple
- An HLS triple

If *COLOR* is one of the above and is found in the screen color map, its color number in the screen color map is returned. If not, an error is generated unless *NOERRFLG* is non-NIL, in which case NIL is returned.

(RGBP *X*)

[Function]

Returns *X* if *X* is an RGB triple; otherwise it returns NIL.

(HLSP *X*)

[Function]

Returns *X* if *X* is an HLS triple; otherwise it returns NIL.

## Color Maps

---

The screen color map holds the information about what color is displayed on the color screen for each pixel value in the color screen bitmap. If you change the values in the current screen color map, this change is reflected in the colors displayed at the next vertical retrace (approximately 1/30 of a second). The color map can be changed to obtain dramatic effects.

(SCREENCOLORMAP *NEWCOLORMAP*) [Function]

Reads and sets the color map used by the color display. If *NEWCOLORMAP* is non-NIL, it should be a color map, and SCREENCOLORMAP sets the system color map to be that color map. The value returned is the value of the screen color map before SCREENCOLORMAP was called. If *NEWCOLORMAP* is NIL, the current screen color map is returned without change.

(CMYCOLORMAP *CYANBITS MAGENTABITS YELLOWBITS*  
*BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *CYANBITS* bits in the cyan plane, *MAGENTABITS* bits in the magenta plane, and *YELLOWBITS* bits in the yellow plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 0 and black is 255.

(RGBCOLORMAP *REDBITS GREENBITS BLUEBITS*  
*BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *REDBITS* bits in the red plane, *GREENBITS* bits in the green plane, and *BLUEBITS* bits in the blue plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 255 and black is 0.

(GRAYCOLORMAP *BITSPERPIXEL*) [Function]

Returns a color map containing shades of gray. White is 0 and black is 255.

(COLORMAPCREATE *INTENSITIES BITSPERPIXEL*) [Function]

Creates a color map for a screen that has *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is NIL, the number of bits per pixel is taken from the current color display setting. *INTENSITIES* specifies the initial colors that should be in the map. If *INTENSITIES* is not NIL, it should be a list of color specifications other than color numbers, e.g., the list of RGB triples returned by the function INTENSITIESFROMCOLOR MAP.

(INTENSITIESFROM COLORMAP *COLORMAP*) [Function]

Returns a list of the intensity levels of *COLORMAP* in a form accepted by COLORMAPCREATE. The default is SCREENCOLORMAP. This list can be written on file, providing a way of saving color map specifications.

(COLORMAPCOPY *COLORMAP BITSPERPIXEL*) [Function]

Returns a color map that contains the same color intensities as *COLORMAP* if *COLORMAP* is a color map. Otherwise, it returns a color map with default color values.

(MAPOFACOLOR *PRIMARIES*)

[Function]

Returns a color map that is different shades of one or more of the primary colors. For example, (MAPOFACOLOR ' (RED GREEN BLUE) ) gives a color map of different shades of gray; (MAPOFACOLOR ' RED) gives different shades of red.

## Changing Color Maps

The following functions are provided to access and change the intensity levels in a color map.

(SETCOLORINTENSITY *COLORMAP COLORNUMBER COLORSPEC*)

[Function]

Sets the primary intensities of color number *COLORNUMBER* in the color map *COLORMAP* to the ones specified by *COLORSPEC*. *COLORSPEC* can be either an RGB triple, an HLS triple, or a color name. The value returned is NIL.

(COLORLEVEL *COLORMAP COLORNUMBER PRIMARYNEWLEVEL*)

[Function]

Sets and reads the intensity level of the primary color *PRIMARY* (RED, GREEN, or BLUE) for the color number *COLORNUMBER* in the color map *COLORMAP*. If *NEWLEVEL* is a number between 0 and 255, it is set. The previous value of the intensity of *PRIMARY* is returned.

(ADJUSTCOLORMAP *PRIMARY DELTA COLORMAP*)

[Function]

Adds *DELTA* to the intensity of the *PRIMARY* color value (RED, GREEN, or BLUE) for every color number in *COLORMAP*.

(ROTATECOLORMAP *STARTCOLOR THRUCOLOR*)

[Function]

Rotates a sequence of colors in the SCREENCOLORMAP. The rotation moves the intensity values of color number *STARTCOLOR* into color number *STARTCOLOR*+1, the intensity values of color number *STARTCOLOR*+1 into color number *STARTCOLOR*+2, etc., and *THRUCOLOR*'s values into *STARTCOLOR*.

(EDITCOLORMAP *VAR NOQFLG*)

[Function]

Allows interactive editing of a color map. If *VAR* is an atom whose value is a color map, its value is edited. Otherwise a new color map is created and edited. The color map being edited becomes the screen color map while the editing takes place so that its effects can be observed. The edited color map is returned as the value. If *NOQFLG* is NIL and the color display is on, you are asked if you want a test pattern of colors. If you answer "yes," the function SHOWCOLORTESTPATTERN is called, which displays a test pattern showing blocks of each of the possible colors.

The system prompts for the location of a color control window to be placed on the black-and-white display. This window allows the value of any of the colors to be changed. The number of the color being edited is in the upper left part of the window. Six bars are displayed. The right three bars give the color intensities for the three primary colors of the current color number. The left three bars give the value of the color's Hue, Lightness, and Saturation parameters. To change these levels, position the mouse cursor in one of the bars and press the left mouse button. While the left button is down, the value of that parameter tracks the Y position of the cursor. When the left button is released, the color tracking stops. To change the color being edited, press the middle

mouse button while the cursor is inside the edit window. This brings up a menu of color numbers. Select one to set the current color to the selected color.

The color being edited can also be changed by selecting the menu item PickPt. This switches the cursor onto the color screen and allows you to select a point from the color screen. It then edits the color of the selected point.

To stop the editing, move the cursor into the title of the editing window and press the middle button. This brings up a menu. Select Stop to quit.

## Color Bitmaps

---

A color bitmap is actually a bitmap that has more than one bit per pixel. Use the function `BITSPERPIXEL` to test whether a bitmap is a color bitmap.

(`BITSPERPIXEL BITMAP`) [Function]

Returns the bits per pixel of *BITMAP*. If this does not equal one, *BITMAP* is a color bitmap.

In multiple-bit-per-pixel bitmaps, the bits that represent a pixel are stored contiguously. `BITMAPCREATE` is passed a *BITSPERPIXEL* argument to create multiple-bit-per-pixel bitmaps.

With CG4,CG6, `BITSPERPIXEL` is 8 for color bitmaps.

(`BITMAPCREATE WIDTH HEIGHT BITSPERPIXEL`) [Function]

Creates a color bitmap that is *WIDTH* pixels wide by *HEIGHT* pixels high allowing *BITSPERPIXEL* bits per pixel. Currently any value of *BITSPERPIXEL* except 1, 4, 8, or `NIL` (defaulting to 1) causes an error.

`BITMAPCREATE` may return two types of bitmap. An ordinary `BITMAP` type usually contains up to 131066 cells (32 bits of data) in it. A larger bitmap can be created as `BIGBM`.

(`DATATYPE BIGBM (BIGBMWIDTH BIGBMHEIGHT BIGBMLIST)`)

`BIGBM` has the list of pointers which point to `BITMAPS`.

An 8 bpp screen bitmap (1052 \* 900) uses approximately 1 Mbyte.

(`COLORSCREENBITMAP`) [Function]

Returns the bitmap that is being or will be displayed on the color display. This is `NIL` if the color display has never been turned on (see `COLORDISPLAY` below).

## Screens, Screenpositions, and Screenregions

---

In addition to positions and regions, you need to be aware of screens, screenpositions, and screenregions in the presence of multiple screens.

## Screens

SCREEN [Datatype]

There are generally two screen datatype instances in existence when working with color. This is because you are attached to two displays, a black and white display and a color display.

(MAINSCREEN) [Function]

Returns the screen datatype instance that represents the black and white screen. This will be something like {SCREEN}#74,24740.

(COLORSCREEN) [Function]

Returns the screen datatype instance that represents the color screen. Screens appear as part of screenpositions and screenregions, serving as the extra information needed to make clear whether a particular position or region should be viewed as lying on the black and white display or the color display.

(SCREENBITMAP SCREEN) [Function]

Returns the bitmap destination of SCREEN. If SCREEN is NIL, it returns the black and white screen bitmap.

## Screenpositions

SCREENPOSITION [Record]

Somewhat like a position, a screenposition denotes a point in an X,Y coordinate system on a particular screen. Screenpositions are defined according to the following record declaration:

```
(RECORD SCREENPOSITION (SCREEN . POSITION)
 (SUBRECORD POSITION))
```

A SCREENPOSITION is an instance of a record with fields XCOORD, YCOORD, and SCREEN and is manipulated with the standard record package facilities. For example, (create SCREENPOSITION XCOORD \_ 10 YCOORD \_ 20 SCREEN \_ (COLORSCREEN)) creates a screenposition representing the point (10,20) on the color display. You can extract the position of a screenposition by fetching its POSITION. For example, (fetch (SCREENPOSITION POSITION) of SP12).

## Screenregions

SCREENREGION [Record]

Similar to a region, a screenregion denotes a rectangular area in a coordinate system. Screenregions are defined according to the following record declaration:

```
(RECORD SCREENREGION (SCREEN . REGION) (SUBRECORD REGION))
```

A screenregion is characterized by the coordinates of its bottom left corner and its width and height. A SCREENREGION is a record with fields LEFT, BOTTOM, WIDTH, HEIGHT, and SCREEN. It can be manipulated with the standard record package facilities. There are access functions for the REGION record that return the TOP and RIGHT of the region. You can extract the region of a screenregion by fetching its REGION. For example, (fetch (SCREENREGION REGION) of SR8).

## Screenposition and Screenregion Prompting

The following functions can be used by programs to allow you to interactively specify screenpositions or screenregions on a display screen.

(GETSCREENPOSITION *WINDOW CURSOR*) [Function]

Similar to GETPOSITION. Returns a SCREENPOSITION you specify. GETSCREENPOSITION waits for you to press and release the left button of the mouse and returns the cursor screenposition at the time of release. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in screen coordinates.

(GETBOXSCREENPOSITION *BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Similar to GETBOXPOSITION. Returns a SCREENPOSITION you specified. This function allows you to position a "ghost" region of size *BOXWIDTH* by *BOXHEIGHT* on a screen, and returns the SCREENPOSITION of the lower left corner of the screenregion chosen. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. To change to another corner, press and hold the right button. With the right button down, the cursor can be moved across a screen or to other screens without effect on the ghost region frame. When the right button is released, the mouse snaps to the nearest corner, which then becomes locked to the cursor. (To change the held corner after the left or middle button is down, hold both the original button and the right button down while moving the cursor to the desired new corner, then release just the right button.) When the left or middle button is pressed and released, the lower left corner of the screenregion chosen at the time of release is returned. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in the screen.

(GETSCREENREGION *MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS*) [Function]

Similar to GETREGION. Returns a SCREENREGION you specified. This function allows you to specify a new screenregion and returns that screenregion. GETSCREENREGION prompts for a screenregion by displaying a four-pronged box next to the cursor arrow at one corner of a "ghost" region: . If you press the left button, the corner of a "ghost" screenregion opposite the cursor is locked where it is. Once one corner has been fixed, the ghost screenregion expands as the cursor moves.

To specify a screenregion:

1. Move the ghost box so that the corner opposite the cursor is at one corner of the intended screenregion.
2. Press the left button.
3. Move the cursor to the screenposition of the opposite corner of the intended screenregion while holding down the left button
4. Release the left button.

Before one corner has been fixed, you can switch the cursor to another corner of the ghost screenregion by holding down the right button. With the right button down, the cursor changes to a "forceps"  and the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner of the ghost screenregion.

After one corner has been fixed, you can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner, which becomes the moving corner. In this way, the screenregion may be moved all over a screen, and to other screens, before its size and screenposition are finalized.

The size of the initial ghost screenregion is controlled by the *MINWIDTH*, *MINHEIGHT*, *OLDREGION*, and *INITCORNERS* arguments.

(GETBOXSCREENREGION *WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Similar to GETBOXREGION. Returns a SCREENREGION you specified. This function performs the same prompting as GETBOXSCREENPOSITION and returns the SCREENREGION specified instead of the SCREENPOSITION of its lower left corner.

## Color Windows and Menus

The Medley window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion. Menus are a special type of window provided by the window system, used for displaying a set of items, on which you use the mouse cursor to make a selection. The menu facility also allows you to create and use menus in interactive programs.

(CREATEW *REGION TITLE BORDERSIZE NOOPENFLG*) [Function]

Creates a new window. *REGION* indicates where and how large the window should be by specifying the exterior screenregion of the window. In a user environment with multiple screens, such as a black and white screen and color screen both connected to the same machine, there is a new special problem in indicating which screen the *REGION* is supposed to be a region of. To resolve this problem, allow CREATEW to take screenregion arguments as *REGION*.

For example:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
 SCREEN _ (COLORSCREEN)
 LEFT _ 20 BOTTOM _ 210
 WIDTH _ 290 HEIGHT _ 170)
 "FOO WINDOW"))
```

creates a window titled "FOO WINDOW" on the color screen. To create a window on the black and white screen, use SCREEN \_ (MAINSCREEN) in the CREATE SCREENREGION expression.

Note: It is still perfectly legal to pass in a *REGION* that is a region, not a screenregion, to *CREATEW*, but it is better to pass screenregions. When *REGION* is a region, *REGION* is disambiguated in a somewhat arbitrary manner that may not always turn out to be what you want.

When *REGION* is a region, *REGION* is disambiguated by coercing *REGION* to be a screenregion on the screen which currently contains the cursor. Software calling *CREATEW* with regions instead of screenregions tends to do what you expect.

(WINDOWPROP *WINDOW PROP NEWVALUE*) [NoSpread Function]

If *PROP* is 'SCREEN, then WINDOWPROP returns the screen *WINDOW* is on. If *NEWVALUE* is given (even if given as NIL), with *PROP* is 'SCREEN, then WINDOWPROP generates an error. Any other *PROP* name is handled in the usual way.

(OPENWINDOWS *SCREEN*) [Function]

Returns a list of all open windows on *SCREEN* if *SCREEN* is a screen datatype such as (MAINSCREEN) or (COLORSCREEN). If *SCREEN* is NIL, then *SCREEN* defaults to the screen containing the cursor. If *SCREEN* is T, a list of all open windows on all screens is returned.

---

## Fonts in Color

There is no special function for creating color fonts. You use the same font on every screen.

You can use color characters by specifying the foreground color (the color of the characters of the font) and background color (the color of the background behind the characters that gets printed along with the characters) in the *DISPLAYSTREAM* data used for the current window (see below).

---

## Using Color

The current color implementation allows display streams to operate on color bitmaps. The two functions *DSPCOLOR* and *DSPBACKCOLOR* set the color in which a stream draws when the user defaults a color argument to a drawing function and printing characters.

(DSPCOLOR *COLOR STREAM*) [Function]

Sets the foreground color of a stream. It returns the previous foreground color. If *COLOR* is NIL, it returns the current foreground color without changing anything. The default foreground color is *MINIMUMCOLOR*=0, which is white in the default color maps.

(DSPBACKCOLOR *COLOR STREAM*) [Function]

Sets the background color of a stream. It returns the previous background color. If *COLOR* is NIL, it returns the current background color without changing anything. The default background color is (*MAXIMUMCOLOR* *BITSPERPIXEL*)=255, which is black in the default color maps.



The BITBLT, line-drawing and curve-drawing routines know how to operate on a color-capable stream. Following are some notes about them.

## BITBLTing in Color

If you are BITBLTing from a color bitmap onto another color bitmap of the same bpp, the operations PAINT, INVERT, and ERASE are done on a bit level, not on a pixel level. Thus, painting color 3 onto color 10 results in color 11.

If BITBLTing from a black-and-white bitmap onto a color bitmap, the 1 bits appear in the DSPCOLOR, and the 0 bits in DSPBACKCOLOR. BLTing from black-and-white to color is fairly expensive; if the same bitmap is going to be put up several times in the same color, it is faster to create a color copy and then to BLT the color copy.

If the source type is TEXTURE and the destination bitmap is a color bitmap, the *Texture* argument is taken to be a color. Therefore, to fill an area with the color BLUE assuming COLORSTR is a stream whose destination is the color screen, use (BITBLT NIL NIL NIL COLORSTR 50 75 100 200 'TEXTURE 'REPLACE 'BLUE) .

## Drawing Curves and Lines in Color

For the functions DRAWCIRCLE, DRAWELLIPSE, and DRAWCURVE, the notion of a brush has been extended to include a color. A BRUSH is now (BRUSHSHAPE BRUSHSIZE BRUSHCOLOR). Also, a brush can be a bitmap (which can be a color bitmap).

Line-drawing routines take a color argument which is the color of the line if the destination of the display stream is a color bitmap.

|                                                                        |            |
|------------------------------------------------------------------------|------------|
| (DRAWLINE <i>X1 Y1 X2 Y2 WIDTH OPERATION STREAM</i><br><i>COLOR</i> )  | [Function] |
| (DRAWTO <i>X Y WIDTH OPERATION STREAM COLOR</i> )                      | [Function] |
| (RELDRAWTO <i>X Y WIDTH OPERATION STREAM COLOR</i> )                   | [Function] |
| (DRAWBETWEEN <i>POS1 POS2 WIDTH OPERATION STREAM</i><br><i>COLOR</i> ) | [Function] |

If the *COLOR* argument is NIL, the DSPCOLOR of the stream is used.

## Printing in Color

Printing only works in REPLACE mode. The characters have a background color and a foreground color determined by the DSPCOLOR and DSPBACKCOLOR .

Example of printing to an 8 bpp color screen:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
 SCREEN _ (COLORSCREEN) LEFT _ 20
 BOTTOM _ 210 WIDTH _ 290
 HEIGHT _ 170) "FOO WINDOW"))
(DSPCOLOR FOO 'GREEN)
(DSPBACKCOLOR 'YELLOW)
```

(PRINT 'HELLO FOO) prints in green against a yellow background.

## Operating the Cursor on the Color Screen

---

To move the cursor to the color screen, slide the cursor off the left or right edge of the black and white screen onto the color screen, or call the CURSORPOSITION or CURSORSCREEN function.

(WORPCURSOR *ENABLE*) [Function]

If *ENABLE* is NIL, you cannot exit the current screen by sliding the cursor off.

(CURSORPOSITION *NEWPOSITION* - -) [Function]

*NEWPOSITION* can be a position or a screenposition.

(CURSORSCREEN *SCREEN XCOORD YCOORD*) [Function]

Moves the cursor to the screenposition determined by *SCREEN*, *XCOORD*, and *YCOORD*. *SCREEN* should be the value of either (COLORSCREEN) or (MAINSCREEN).

While on the color screen, the cursor is placed by doing BITBLTs in software rather than with microcode and hardware as with the black and white cursor. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. The speed of the color cursor compares well with that of the black-and-white cursor but there can be a noticeable flicker when there is much input/output to the color screen. While the cursor is on the color screen, the black-and-white cursor is cleared giving the appearance that there is never more than one cursor at a given time.

## Miscellaneous Color Functions

---

(COLORIZEBITMAP *BITMAP 0COLOR 1COLOR BITSPERPIXEL*) [Function]

Creates a color bitmap from a black-and-white bitmap. The returned bitmap has color number *1COLOR* in those pixels of *BITMAP* that were 1 and *0COLOR* in those pixels of *BITMAP* that were 0. This provides a way of producing a color bitmap from a black-and-white bitmap.

(UNCOLORIZEBITMAP *BITMAP COLORMAP*) [Function]

Creates a black-and-white bitmap from a color bitmap.

(SHOWCOLORTESTPATTERN *BARSIZE*) [Function]

Displays a pattern of colors on the color display. This is useful when editing a color map. The pattern has squares of the 16 possible colors laid out in two rows at the top of the screen. Colors 0 through 7 are in the top row, and colors 8 through 15 are in the next row. The bottom part of the screen is filled with bars of *BARSIZE* width with consecutive color numbers. The pattern is designed so that every color has a border with every other color (unless *BARSIZE* is too large to allow room for every color—about 20).

[This page intentionally left blank]

# COLOR

---

Color is the software for driving color displays. To run COLOR, you need either a Sun (3, 4 or SPARCstations) with CG4, CG6 color hardware and color display.

The machine independent color graphics code is stored in the library files `LLCOLOR.LCOM` and `COLOR.LCOM`. The Sun color driver resides in the file `MAIKOCOLOR.LCOM` which loads `LLCOLOR.LCOM`, `COLOR.LCOM`, and `TAKEBIGBM.LCOM`.

The CG4,CG6 device supports 8 bpp (bits per pixel) at a resolution of 1152 by 900.

Note: You cannot use COLOR if you are running Medley under X.

## Software Screens

---

To support the various hardware configurations and external display devices, the software has a special datatype, a "screen". A screen represents and controls a physical hardware display. It contains windows and icons, and tracks the mouse. There are two distinct types of screens: a black and white screen, and a color screen.

On workstations with a single hardware display, the display is shared by both the black and white screen and the color screen. It can be changed by moving the mouse cursor. The screen mode may be changed by moving the cursor out of the screen.

## Turning the Color Display Software On and Off

---

The color display software can be turned on and off. While the color display software is on, and a small amount of processing time is used to drive the color display.

(`COLORDISPLAYP`) [Function]

Returns T if the color display is on; otherwise it returns NIL.

(`COLORDISPLAYONOFF TYPE`) [Function]

Turns off the color display if *ONOFF* is set to OFF. If *ONOFF* is set to ON, it turns on the color-display-allocating memory for the color screen bitmap. *TYPE* should be `MAIKOCOLOR`, if you are using a Sun Workstation. The usual sequence of events is to `LOAD MAIKOCOLOR.LCOM` to drive GC4/GC6 color card and then to call `COLORDISPLAY` with the appropriate *TYPE* (`MAIKOCOLOR` for Sun Workstations) to turn the software on. For example,

```
(LOAD 'MAIKOCOLOR.LCOM)
(COLORDISPLAY 'ON 'MAIKOCOLOR)
```

Calling `COLORDISPLAY` allocates memory for the color screen bitmap in the `sysout`. Turning on the color display requires allocating the memory necessary to hold the color display screen bitmap. Turning off the color display does not free this memory; it still exists in the `sysout`.

## Colors

---

The number of bits per pixel (bpp) determines the number of different colors that can be displayed at one time. When there are 8 bpp, 256 colors can be displayed at once. A table, called a *color map*, determines what color actually appears for each pixel value. A color map gives the color in terms of how much of the three primary colors (red, green, and blue) is displayed on the screen for each possible pixel value.

A color can be represented as a number, an atom, or a triple of numbers. The color map provides the final interpretation of how much red, blue, and green a color represents. A color map maps a color number ( $[0 \dots 2^{n_{\text{bits}}}-1]$ ) into the intensities of the three color guns (primary colors red, green, and blue). Each entry in the color map has 8 bits for each of the primary colors, allowing 256 levels per primary or  $2^{24}$  possible colors (not all of which are distinct to the human eye). Within Xerox Lisp programs, colors can be manipulated as numbers, red-green-blue triples, names, or hue-lightness-saturation triples. Any function that takes a color accepts any of these different representations.

If a number is given, it is the color number used in the operation. It must be valid for the color bitmap used in the operation. (Since all of the routines that use a color need to determine the color's number, it is fastest to use numbers for colors. `COLORNUMBERP`, described below, provides a way to translate into numbers from the other representations.)

The following sections describe other ways to represent colors.

### Red-Green-Blue Triples

A red-green-blue (RGB) triple is a list of three numbers, each of which can be between 0 and 255. The nine digits are divided as follows:

- First 3 digits = red intensity
- Second 3 digits = green intensity
- Third 3 digits = blue intensity

When an RGB triple is used, the current color map is searched to find the color with the correct intensities. If none is found, an error is generated. (That is, no attempt is made by the system to assign color numbers to intensities automatically.) An example of an RGB triple is (255 255 255), which gives the color white.

RGB

[Record]

A record defined as (RED GREEN BLUE); it can be used to manipulate RGB triples.

COLORNAMES

[Association list]

Maps names into colors. The CDR of the color name's entry is used as the color corresponding to the color name. This can be any of the other representations.

Note: It can even be another color name. The system does not check for loops in the name space, such as would be caused by putting ' (RED . CRIMSON) and ' (CRIMSON . RED) on COLORNAMES.

Some color names are available in the initial system and allow color programs written by different users to coexist. These are:

| Name    | RGB           | Number in default color maps |     |
|---------|---------------|------------------------------|-----|
| BLACK   | (0 0 0)       | 15                           | 255 |
| BLUE    | (0 0 255)     | 14                           | 252 |
| GREEN   | (0 255 0)     | 13                           | 227 |
| CYAN    | (0 255 255)   | 12                           | 224 |
| RED     | (255 0 0)     | 3                            | 31  |
| MAGENTA | (255 0 255)   | 2                            | 28  |
| YELLOW  | (255 255 0)   | 1                            | 3   |
| WHITE   | (255 255 255) | 0                            | 0   |

## Hue–Lightness–Saturation Triples

A hue–lightness–saturation triple is a list of three numbers:

- The first number (HUE) is an integer between 0 and 355. It indicates a position in degrees on a color wheel (blue at 0, red at 120, and green at 240).
- The second (LIGHTNESS) is a real number between zero. It indicates how much total intensity is in the color.
- The third (SATURATION) is a real number between zero and one. It indicates how disparate the three primary levels are.

HLS

[Record]

A record defined as (HUE LIGHTNESS SATURATION); it is provided to help you manipulate HLS triples.

Example: the color blue is represented in HLS notation by (0 .5 1.0).

(COLORNUMBERP *COLOR* *BITSPERPIXEL* *NOERRFLG*)

[Function]

Returns the color number (offset into the screen color map) of *COLOR*. *COLOR* can be one of the following:

- A positive number less than the maximum number of colors
- A color name
- An RGB triple
- An HLS triple

If *COLOR* is one of the above and is found in the screen color map, its color number in the screen color map is returned. If not, an error is generated unless *NOERRFLG* is non-NIL, in which case NIL is returned.

(RGBP *X*)

[Function]

Returns *X* if *X* is an RGB triple; otherwise it returns NIL.

(HLSP *X*)

[Function]

Returns *X* if *X* is an HLS triple; otherwise it returns NIL.

## Color Maps

---

The screen color map holds the information about what color is displayed on the color screen for each pixel value in the color screen bitmap. If you change the values in the current screen color map, this change is reflected in the colors displayed at the next vertical retrace (approximately 1/30 of a second). The color map can be changed to obtain dramatic effects.

(SCREENCOLORMAP *NEWCOLORMAP*) [Function]

Reads and sets the color map used by the color display. If *NEWCOLORMAP* is non-NIL, it should be a color map, and SCREENCOLORMAP sets the system color map to be that color map. The value returned is the value of the screen color map before SCREENCOLORMAP was called. If *NEWCOLORMAP* is NIL, the current screen color map is returned without change.

(CMYCOLORMAP *CYANBITS MAGENTABITS YELLOWBITS*  
*BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *CYANBITS* bits in the cyan plane, *MAGENTABITS* bits in the magenta plane, and *YELLOWBITS* bits in the yellow plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 0 and black is 255.

(RGBCOLORMAP *REDBITS GREENBITS BLUEBITS*  
*BITSPERPIXEL*) [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes: *REDBITS* bits in the red plane, *GREENBITS* bits in the green plane, and *BLUEBITS* bits in the blue plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 255 and black is 0.

(GRAYCOLORMAP *BITSPERPIXEL*) [Function]

Returns a color map containing shades of gray. White is 0 and black is 255.

(COLORMAPCREATE *INTENSITIES BITSPERPIXEL*) [Function]

Creates a color map for a screen that has *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is NIL, the number of bits per pixel is taken from the current color display setting. *INTENSITIES* specifies the initial colors that should be in the map. If *INTENSITIES* is not NIL, it should be a list of color specifications other than color numbers, e.g., the list of RGB triples returned by the function INTENSITIESFROMCOLOR MAP.

(INTENSITIESFROM COLORMAP *COLORMAP*) [Function]

Returns a list of the intensity levels of *COLORMAP* in a form accepted by COLORMAPCREATE. The default is SCREENCOLORMAP. This list can be written on file, providing a way of saving color map specifications.

(COLORMAPCOPY *COLORMAP BITSPERPIXEL*) [Function]

Returns a color map that contains the same color intensities as *COLORMAP* if *COLORMAP* is a color map. Otherwise, it returns a color map with default color values.

(MAPOFACOLOR *PRIMARIES*)

[Function]

Returns a color map that is different shades of one or more of the primary colors. For example, (MAPOFACOLOR ' (RED GREEN BLUE) ) gives a color map of different shades of gray; (MAPOFACOLOR ' RED) gives different shades of red.

## Changing Color Maps

The following functions are provided to access and change the intensity levels in a color map.

(SETCOLORINTENSITY *COLORMAP COLORNUMBER COLORSPEC*)

[Function]

Sets the primary intensities of color number *COLORNUMBER* in the color map *COLORMAP* to the ones specified by *COLORSPEC*. *COLORSPEC* can be either an RGB triple, an HLS triple, or a color name. The value returned is NIL.

(COLORLEVEL *COLORMAP COLORNUMBER PRIMARYNEWLEVEL*)

[Function]

Sets and reads the intensity level of the primary color *PRIMARY* (RED, GREEN, or BLUE) for the color number *COLORNUMBER* in the color map *COLORMAP*. If *NEWLEVEL* is a number between 0 and 255, it is set. The previous value of the intensity of *PRIMARY* is returned.

(ADJUSTCOLORMAP *PRIMARY DELTA COLORMAP*)

[Function]

Adds *DELTA* to the intensity of the *PRIMARY* color value (RED, GREEN, or BLUE) for every color number in *COLORMAP*.

(ROTATECOLORMAP *STARTCOLOR THRUCOLOR*)

[Function]

Rotates a sequence of colors in the SCREENCOLORMAP. The rotation moves the intensity values of color number *STARTCOLOR* into color number *STARTCOLOR*+1, the intensity values of color number *STARTCOLOR*+1 into color number *STARTCOLOR*+2, etc., and *THRUCOLOR*'s values into *STARTCOLOR*.

(EDITCOLORMAP *VAR NOQFLG*)

[Function]

Allows interactive editing of a color map. If *VAR* is an atom whose value is a color map, its value is edited. Otherwise a new color map is created and edited. The color map being edited becomes the screen color map while the editing takes place so that its effects can be observed. The edited color map is returned as the value. If *NOQFLG* is NIL and the color display is on, you are asked if you want a test pattern of colors. If you answer "yes," the function SHOWCOLORTESTPATTERN is called, which displays a test pattern showing blocks of each of the possible colors.

The system prompts for the location of a color control window to be placed on the black-and-white display. This window allows the value of any of the colors to be changed. The number of the color being edited is in the upper left part of the window. Six bars are displayed. The right three bars give the color intensities for the three primary colors of the current color number. The left three bars give the value of the color's Hue, Lightness, and Saturation parameters. To change these levels, position the mouse cursor in one of the bars and press the left mouse button. While the left button is down, the value of that parameter tracks the Y position of the cursor. When the left button is released, the color tracking stops. To change the color being edited, press the middle



mouse button while the cursor is inside the edit window. This brings up a menu of color numbers. Select one to set the current color to the selected color.

The color being edited can also be changed by selecting the menu item PickPt. This switches the cursor onto the color screen and allows you to select a point from the color screen. It then edits the color of the selected point.

To stop the editing, move the cursor into the title of the editing window and press the middle button. This brings up a menu. Select Stop to quit.

## Color Bitmaps

---

A color bitmap is actually a bitmap that has more than one bit per pixel. Use the function `BITSPERPIXEL` to test whether a bitmap is a color bitmap.

(`BITSPERPIXEL BITMAP`) [Function]

Returns the bits per pixel of *BITMAP*. If this does not equal one, *BITMAP* is a color bitmap.

In multiple-bit-per-pixel bitmaps, the bits that represent a pixel are stored contiguously. `BITMAPCREATE` is passed a *BITSPERPIXEL* argument to create multiple-bit-per-pixel bitmaps.

With `CG4`, `CG6`, `BITSPERPIXEL` is 8 for color bitmaps.

(`BITMAPCREATE WIDTH HEIGHT BITSPERPIXEL`) [Function]

Creates a color bitmap that is *WIDTH* pixels wide by *HEIGHT* pixels high allowing *BITSPERPIXEL* bits per pixel. Currently any value of *BITSPERPIXEL* except 1, 4, 8, or `NIL` (defaulting to 1) causes an error.

`BITMAPCREATE` may return two types of bitmap. An ordinary `BITMAP` type usually contains up to 131066 cells (32 bits of data) in it. A larger bitmap can be created as `BIGBM`.

(`DATATYPE BIGBM (BIGBMWIDTH BIGBMHEIGHT BIGBMLIST)`)

`BIGBM` has the list of pointers which point to `BITMAPS`.

An 8 bpp screen bitmap (1052 \* 900) uses approximately 1 Mbyte.

(`COLORSCREENBITMAP`) [Function]

Returns the bitmap that is being or will be displayed on the color display. This is `NIL` if the color display has never been turned on (see `COLORDISPLAY` below).

## Screens, Screenpositions, and Screenregions

---

In addition to positions and regions, you need to be aware of screens, screenpositions, and screenregions in the presence of multiple screens.

## Screens

SCREEN [Datatype]

There are generally two screen datatype instances in existence when working with color. This is because you are attached to two displays, a black and white display and a color display.

(MAINSCREEN) [Function]

Returns the screen datatype instance that represents the black and white screen. This will be something like {SCREEN}#74,24740.

(COLORSCREEN) [Function]

Returns the screen datatype instance that represents the color screen. Screens appear as part of screenpositions and screenregions, serving as the extra information needed to make clear whether a particular position or region should be viewed as lying on the black and white display or the color display.

(SCREENBITMAP SCREEN) [Function]

Returns the bitmap destination of SCREEN. If SCREEN is NIL, it returns the black and white screen bitmap.

## Screenpositions

SCREENPOSITION [Record]

Somewhat like a position, a screenposition denotes a point in an X,Y coordinate system on a particular screen. Screenpositions are defined according to the following record declaration:

```
(RECORD SCREENPOSITION (SCREEN . POSITION)
 (SUBRECORD POSITION))
```

A SCREENPOSITION is an instance of a record with fields XCOORD, YCOORD, and SCREEN and is manipulated with the standard record package facilities. For example, (create SCREENPOSITION XCOORD \_ 10 YCOORD \_ 20 SCREEN \_ (COLORSCREEN)) creates a screenposition representing the point (10,20) on the color display. You can extract the position of a screenposition by fetching its POSITION. For example, (fetch (SCREENPOSITION POSITION) of SP12).

## Screenregions

SCREENREGION [Record]

Similar to a region, a screenregion denotes a rectangular area in a coordinate system. Screenregions are defined according to the following record declaration:

```
(RECORD SCREENREGION (SCREEN . REGION) (SUBRECORD REGION))
```

A screenregion is characterized by the coordinates of its bottom left corner and its width and height. A SCREENREGION is a record with fields LEFT, BOTTOM, WIDTH, HEIGHT, and SCREEN. It can be manipulated with the standard record package facilities. There are access functions for the REGION record that return the TOP and RIGHT of the region. You can extract the region of a screenregion by fetching its REGION. For example, (fetch (SCREENREGION REGION) of SR8).

## Screenposition and Screenregion Prompting

The following functions can be used by programs to allow you to interactively specify screenpositions or screenregions on a display screen.

(GETSCREENPOSITION *WINDOW CURSOR*) [Function]

Similar to GETPOSITION. Returns a SCREENPOSITION you specify. GETSCREENPOSITION waits for you to press and release the left button of the mouse and returns the cursor screenposition at the time of release. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in screen coordinates.

(GETBOXSCREENPOSITION *BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Similar to GETBOXPOSITION. Returns a SCREENPOSITION you specified. This function allows you to position a "ghost" region of size *BOXWIDTH* by *BOXHEIGHT* on a screen, and returns the SCREENPOSITION of the lower left corner of the screenregion chosen. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. To change to another corner, press and hold the right button. With the right button down, the cursor can be moved across a screen or to other screens without effect on the ghost region frame. When the right button is released, the mouse snaps to the nearest corner, which then becomes locked to the cursor. (To change the held corner after the left or middle button is down, hold both the original button and the right button down while moving the cursor to the desired new corner, then release just the right button.) When the left or middle button is pressed and released, the lower left corner of the screenregion chosen at the time of release is returned. If *WINDOW* is a WINDOW, the screenposition is on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position is in the screen.

(GETSCREENREGION *MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS*) [Function]

Similar to GETREGION. Returns a SCREENREGION you specified. This function allows you to specify a new screenregion and returns that screenregion. GETSCREENREGION prompts for a screenregion by displaying a four-pronged box next to the cursor arrow at one corner of a "ghost" region: . If you press the left button, the corner of a "ghost" screenregion opposite the cursor is locked where it is. Once one corner has been fixed, the ghost screenregion expands as the cursor moves.

To specify a screenregion:

1. Move the ghost box so that the corner opposite the cursor is at one corner of the intended screenregion.
2. Press the left button.
3. Move the cursor to the screenposition of the opposite corner of the intended screenregion while holding down the left button
4. Release the left button.

Before one corner has been fixed, you can switch the cursor to another corner of the ghost screenregion by holding down the right button. With the right button down, the cursor changes to a "forceps"  and the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner of the ghost screenregion.

After one corner has been fixed, you can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor snaps to the nearest corner, which becomes the moving corner. In this way, the screenregion may be moved all over a screen, and to other screens, before its size and screenposition are finalized.

The size of the initial ghost screenregion is controlled by the *MINWIDTH*, *MINHEIGHT*, *OLDREGION*, and *INITCORNERS* arguments.

(GETBOXSCREENREGION *WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

Similar to GETBOXREGION. Returns a SCREENREGION you specified. This function performs the same prompting as GETBOXSCREENPOSITION and returns the SCREENREGION specified instead of the SCREENPOSITION of its lower left corner.

## Color Windows and Menus

The Medley window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion. Menus are a special type of window provided by the window system, used for displaying a set of items, on which you use the mouse cursor to make a selection. The menu facility also allows you to create and use menus in interactive programs.

(CREATEW *REGION TITLE BORDERSIZE NOOPENFLG*) [Function]

Creates a new window. *REGION* indicates where and how large the window should be by specifying the exterior screenregion of the window. In a user environment with multiple screens, such as a black and white screen and color screen both connected to the same machine, there is a new special problem in indicating which screen the *REGION* is supposed to be a region of. To resolve this problem, allow CREATEW to take screenregion arguments as *REGION*.

For example:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
 SCREEN _ (COLORSCREEN)
 LEFT _ 20 BOTTOM _ 210
 WIDTH _ 290 HEIGHT _ 170)
 "FOO WINDOW"))
```

creates a window titled "FOO WINDOW" on the color screen. To create a window on the black and white screen, use SCREEN \_ (MAINSCREEN) in the CREATE SCREENREGION expression.

Note: It is still perfectly legal to pass in a *REGION* that is a region, not a screenregion, to *CREATEW*, but it is better to pass screenregions. When *REGION* is a region, *REGION* is disambiguated in a somewhat arbitrary manner that may not always turn out to be what you want.

When *REGION* is a region, *REGION* is disambiguated by coercing *REGION* to be a screenregion on the screen which currently contains the cursor. Software calling *CREATEW* with regions instead of screenregions tends to do what you expect.

(WINDOWPROP *WINDOW PROP NEWVALUE*) [NoSpread Function]

If *PROP* is 'SCREEN, then WINDOWPROP returns the screen *WINDOW* is on. If *NEWVALUE* is given (even if given as NIL), with *PROP* is 'SCREEN, then WINDOWPROP generates an error. Any other *PROP* name is handled in the usual way.

(OPENWINDOWS *SCREEN*) [Function]

Returns a list of all open windows on *SCREEN* if *SCREEN* is a screen datatype such as (MAINSCREEN) or (COLORSCREEN). If *SCREEN* is NIL, then *SCREEN* defaults to the screen containing the cursor. If *SCREEN* is T, a list of all open windows on all screens is returned.

---

## Fonts in Color

There is no special function for creating color fonts. You use the same font on every screen.

You can use color characters by specifying the foreground color (the color of the characters of the font) and background color (the color of the background behind the characters that gets printed along with the characters) in the *DISPLAYSTREAM* data used for the current window (see below).

---

## Using Color

The current color implementation allows display streams to operate on color bitmaps. The two functions *DSPCOLOR* and *DSPBACKCOLOR* set the color in which a stream draws when the user defaults a color argument to a drawing function and printing characters.

(DSPCOLOR *COLOR STREAM*) [Function]

Sets the foreground color of a stream. It returns the previous foreground color. If *COLOR* is NIL, it returns the current foreground color without changing anything. The default foreground color is *MINIMUMCOLOR*=0, which is white in the default color maps.

(DSPBACKCOLOR *COLOR STREAM*) [Function]

Sets the background color of a stream. It returns the previous background color. If *COLOR* is NIL, it returns the current background color without changing anything. The default background color is (*MAXIMUMCOLOR* *BITSPERPIXEL*)=255, which is black in the default color maps.

The BITBLT, line-drawing and curve-drawing routines know how to operate on a color-capable stream. Following are some notes about them.

## BITBLTing in Color

If you are BITBLTing from a color bitmap onto another color bitmap of the same bpp, the operations PAINT, INVERT, and ERASE are done on a bit level, not on a pixel level. Thus, painting color 3 onto color 10 results in color 11.

If BITBLTing from a black-and-white bitmap onto a color bitmap, the 1 bits appear in the DSPCOLOR, and the 0 bits in DSPBACKCOLOR. BLTing from black-and-white to color is fairly expensive; if the same bitmap is going to be put up several times in the same color, it is faster to create a color copy and then to BLT the color copy.

If the source type is TEXTURE and the destination bitmap is a color bitmap, the *Texture* argument is taken to be a color. Therefore, to fill an area with the color BLUE assuming COLORSTR is a stream whose destination is the color screen, use (BITBLT NIL NIL NIL COLORSTR 50 75 100 200 'TEXTURE 'REPLACE 'BLUE) .

## Drawing Curves and Lines in Color

For the functions DRAWCIRCLE, DRAWELLIPSE, and DRAWCURVE, the notion of a brush has been extended to include a color. A BRUSH is now (BRUSHSHAPE BRUSHSIZE BRUSHCOLOR). Also, a brush can be a bitmap (which can be a color bitmap).

Line-drawing routines take a color argument which is the color of the line if the destination of the display stream is a color bitmap.

|                                                                        |            |
|------------------------------------------------------------------------|------------|
| (DRAWLINE <i>X1 Y1 X2 Y2 WIDTH OPERATION STREAM</i><br><i>COLOR</i> )  | [Function] |
| (DRAWTO <i>X Y WIDTH OPERATION STREAM COLOR</i> )                      | [Function] |
| (RELDRAWTO <i>X Y WIDTH OPERATION STREAM COLOR</i> )                   | [Function] |
| (DRAWBETWEEN <i>POS1 POS2 WIDTH OPERATION STREAM</i><br><i>COLOR</i> ) | [Function] |

If the *COLOR* argument is NIL, the DSPCOLOR of the stream is used.

## Printing in Color

Printing only works in REPLACE mode. The characters have a background color and a foreground color determined by the DSPCOLOR and DSPBACKCOLOR .

Example of printing to an 8 bpp color screen:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
 SCREEN _ (COLORSCREEN) LEFT _ 20
 BOTTOM _ 210 WIDTH _ 290
 HEIGHT _ 170) "FOO WINDOW"))
(DSPCOLOR FOO 'GREEN)
(DSPBACKCOLOR 'YELLOW)
```

(PRINT 'HELLO FOO) prints in green against a yellow background.

## Operating the Cursor on the Color Screen

---

To move the cursor to the color screen, slide the cursor off the left or right edge of the black and white screen onto the color screen, or call the CURSORPOSITION or CURSORSCREEN function.

(WORPCURSOR *ENABLE*) [Function]

If *ENABLE* is NIL, you cannot exit the current screen by sliding the cursor off.

(CURSORPOSITION *NEWPOSITION* - -) [Function]

*NEWPOSITION* can be a position or a screenposition.

(CURSORSCREEN *SCREEN XCOORD YCOORD*) [Function]

Moves the cursor to the screenposition determined by *SCREEN*, *XCOORD*, and *YCOORD*. *SCREEN* should be the value of either (COLORSCREEN) or (MAINScreens).

While on the color screen, the cursor is placed by doing BITBLTs in software rather than with microcode and hardware as with the black and white cursor. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. The speed of the color cursor compares well with that of the black-and-white cursor but there can be a noticeable flicker when there is much input/output to the color screen. While the cursor is on the color screen, the black-and-white cursor is cleared giving the appearance that there is never more than one cursor at a given time.

## Miscellaneous Color Functions

---

(COLORIZEBITMAP *BITMAP 0COLOR 1COLOR BITSPERPIXEL*) [Function]

Creates a color bitmap from a black-and-white bitmap. The returned bitmap has color number *1COLOR* in those pixels of *BITMAP* that were 1 and *0COLOR* in those pixels of *BITMAP* that were 0. This provides a way of producing a color bitmap from a black-and-white bitmap.

(UNCOLORIZEBITMAP *BITMAP COLORMAP*) [Function]

Creates a black-and-white bitmap from a color bitmap.

(SHOWCOLORTESTPATTERN *BARSIZE*) [Function]

Displays a pattern of colors on the color display. This is useful when editing a color map. The pattern has squares of the 16 possible colors laid out in two rows at the top of the screen. Colors 0 through 7 are in the top row, and colors 8 through 15 are in the next row. The bottom part of the screen is filled with bars of *BARSIZE* width with consecutive color numbers. The pattern is designed so that every color has a border with every other color (unless *BARSIZE* is too large to allow room for every color—about 20).

[This page intentionally left blank]



# COLOR

---

---

## Introduction

---

This document describes software for driving color displays. In order to run COLOR, you need either a Sun (3 or 4) with CG4 color hardware and display, a Dorado (Xerox 1132) with attached color display, or a Dandelion (Xerox 1108) with attached BusMaster and color display.

The color software which is distributed among a number of files can be divided into a machine independent group of files that all users can usefully load and a machine dependent group containing files that work for particular combinations of hardware.

The machine independent color graphics code is stored in the library files LLCOLOR.LCOM and COLOR.LCOM. LOADING COLOR.LCOM causes LLCOLOR.LCOM to be LOADED.

The machine dependent portions of Xerox Lisp color software is stored in files such as MAIKOCOLOR.LCOM, DORADOCOLOR.LCOM, or COLORNNCC.LCOM. The user LOADs one of these files according to what kind of machine and color card the user is using.

The Sun color driver resides in the file MAIKOCOLOR.LCOM which loads LLCOLOR.LCOM and COLOR.LCOM. The CG4 device supports 8 bpp at 1152 by 900 resolution. The user must be running ldecolor, the special color capable emulator. The physical display monitor is shared by both the monochrome and color screens (described below).

The Dorado color driver resides in the file DORADOCOLOR.LCOM which loads LLCOLOR.LCOM and COLOR.LCOM. The Dorado color board supports four or eight bits per pixel (bpp) at 640 by 480 resolution. (The board supports 24 bpp also, but Xerox Lisp doesn't yet.)

The Dandelion color drivers reside in the files DANDELIONUFO.LCOM, DANDELIONUFO4096.LCOM, and COLORNNCC.LCOM, one package for each of three different kinds of boards. The user should load one of these packages on a Dandelion attached to a BusMaster and color display. The DANDELIONUFO and DANDELIONUFO4096 packages drive 4 bpp at 640 by 400 resolution color boards used inside Xerox which have been made obsolete by COLORNNCC. The COLORNNCC package drives an 8 bpp color at 512 by 480 resolution board, the Revolution 512 x 8, made by Number Nine Computer Corporation. The Revolution 512 x 8 is available both inside and outside Xerox through Number Nine.

---

## Hardware Displays and Software Screens

---

On some workstations (such as the Dorado and Dandelion), there may be physically two separate displays. On most Suns, there is a single physical display, which additionally may be shared by two Unix devices. One device is monochrome (b/w), and the other is color.

To support the various hardware configurations and external display devices, the software has a special datatype, a "screen". There are two distinct instances of screens, a b/w screen, and a color screen. A screen represents and controls a physical hardware display, and contains windows, icons, and tracks the mouse.

On workstations with physically two separate hardware displays, each display is represented by a corresponding screen data structure. On workstations with a single hardware display, the display is shared by both the b/w screen and the color screen.

In all cases, before initialization only the b/w screen (and thus display) is visible and active. After initialization both screens are active (can contain screen images), although on single displays, only one screen is visible at a time. Since each screen logically controls a display, we will henceforth use the terms "screen" and "display" interchangeably. Screens are discussed in greater detail below.

---

## Turning the Color Display Software On and Off

---

The color display software can be turned on and off. While the color display software is on, the memory used for the color display screen bitmap is locked down, and a small amount of processing time is used to drive the color display.

**(COLORDISPLAYP)** [Function]

returns T if the color display is on; otherwise it returns NIL.

**(COLORDISPLAYONOFF TYPE)** [Function]

turns off the color display if *ONOFF* is 'OFF. If *ONOFF* is 'ON, it turns on the color display allocating memory for the color screen bitmap. *TYPE* should be one of 'MAIKOCOLOR, 'DORADOCOLOR, 'DANDELIONUFO, 'DANDELIONUFO4096, or 'COLORNCC. The usual sequence of events for the user is to LOAD the software needed to drive a particular color card and then to call COLORDISPLAY with the appropriate *TYPE* to turn the software on. For example,

```
(LOAD 'COLOR.LCOM)
```

```
(LOAD 'COLORNCC.LCOM)
```

```
(COLORDISPLAY 'ON 'REV512X8)
```

will turn on the software needed to drive the Number Nine Computer Corporation's Revolution 512 x 8 card with 1108 and BusMaster.

Besides initializing or reinitializing a color card that has been powered off, COLORDISPLAY allocates memory for the color screen bitmap. Turning on the color display requires allocating and locking down the memory necessary to hold the color display screen bitmap. Turning off the color display frees this memory.

## Colors

---

The number of bits per pixel determines the number of different colors that can be displayed at one time. When there are 4 bpp, 16 colors can be displayed at once. When there are 8 bpp, 256 colors can be displayed at once. A table called a *color map* determines what color actually appears for each pixel value. A color map gives the color in terms of how much of the three primary colors (red, green, and blue) is displayed on the screen for each possible pixel value.

A color can be represented as a number, an atom, or a triple of numbers. Colors are ultimately given their final interpretation into how much red, blue, and green they represent through a color map. A color map maps a color number ( $[0 \dots 2^{n_{\text{bits}}}-1]$ ) into the intensities of the three color guns (primary colors red, green, and blue). Each entry in the color map has eight bits for each of the primary colors, allowing 256 levels per primary or  $2^{24}$  possible colors (not all of which are distinct to the human eye). Within Xerox Lisp programs, colors can be manipulated as numbers, red-green-blue triples, names, or hue-lightness-saturation triples. Any function that takes a color accepts any of the different representations.

If a number is given, it is the color number used in the operation. It must be valid for the color bitmap used in the operation. (Since all of the routines that use a color need to determine its number, it is fastest to use numbers for colors. COLORNUMBERP, described below, provides a way to translate into numbers from the other representations.)

## Red Green Blue Triples

---

A red green blue (RGB) triple is a list of three numbers between 0 and 255. The first element gives the intensity for red, the second for green, and the third for blue. When an RGB triple is used, the current color map is searched to find the color with the correct intensities. If none is found, an error is generated. (That is, no attempt is made by the system to assign color numbers to intensities automatically.) An example of an RGB triple is (255 255 255), which gives the color white.

### RGB

[Record]

is a record that is defined as (RED GREEN BLUE); it can be used to manipulate RGB triples.

### COLORNAMES

[Association list]

maps names into colors. The CDR of the color name's entry is used as the color corresponding to the color name. This can be any of the other representations. (Note: It can even be another color name. Loops in the name space such as would be caused by putting '(RED . CRIMSON) and '(CRIMSON . RED) on COLORNAMES are *not* checked for by the system.) Some color names are available in the initial system and are intended to allow color programs written by different users to coexist. These are:

| Name    | RGB           | Number in default color maps |     |
|---------|---------------|------------------------------|-----|
| BLACK   | (0 0 0)       | 15                           | 255 |
| BLUE    | (0 0 255)     | 14                           | 252 |
| GREEN   | (0 255 0)     | 13                           | 227 |
| CYAN    | (0 255 255)   | 12                           | 224 |
| RED     | (255 0 0)     | 3                            | 31  |
| MAGENTA | (255 0 255)   | 2                            | 28  |
| YELLOW  | (255 255 0)   | 1                            | 3   |
| WHITE   | (255 255 255) | 0                            | 0   |

## Hue Lightness Saturation Triples

A hue lightness saturation triple is a list of three numbers. The first number (HUE) is an integer between 0 and 355 and indicates a position in degrees on a color wheel (blue at 0, red at 120, and green at 240). The second (LIGHTNESS) is a real number between zero and one that indicates how much total intensity is in the color. The third (SATURATION) is a real number between zero and one that indicates how disparate the three primary levels are.

**HLS** [Record]

is a record defined as (HUE LIGHTNESS SATURATION); it is provided to manipulate HLS triples. Example: the color blue is represented in HLS notation by (0 .5 1.0).

**(COLORNUMBERP COLOR BITSPERPIXEL NOERRFLG)**[Function]

returns the color number (offset into the screen color map) of *COLOR*. *COLOR* is one of the following:

- A positive number less than the maximum number of colors,
- A color name,
- AN RGB triple, or
- An HLS triple.

If *COLOR* is one of the above and is found in the screen color map, its color number in the screen color map is returned. If not, an error is generated unless NOERRFLG is non-NIL, in which case NIL is returned.

**(RGBP X)** [Function]

returns *X* if *X* is an RGB triple; NIL otherwise.

**(HLSP X)** [Function]

returns *X* if *X* is an HLS triple; NIL otherwise.

## Color Maps

The screen color map holds the information about what color is displayed on the color screen for each pixel value in the color screen bitmap. The values in the current screen color map may be changed, and this change is reflected in the colors displayed at the

next vertical retrace (approximately 1/30 of a second). The color map can be changed to obtain dramatic effects.

**(SCREENCOLORMAP NEWCOLORMAP)** [Function]

reads and sets the color map that is used by the color display. If *NEWCOLORMAP* is non-NIL, it should be a color map, and **SCREENCOLORMAP** sets the system color map to be that color map. The value returned is the value of the screen color map before **SCREENCOLORMAP** was called. If *NEWCOLORMAP* is NIL, the current screen color map is returned without change.

**(CMYCOLORMAP CYANBITS MAGENTABITS YELLOWBITS BITSPERPIXEL)** [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes with *CYANBITS* bits being in the cyan plane, *MAGENTABITS* bits being in the magenta plane, and *YELLOWBITS* bits being in the yellow plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 0 and black is 255.

**(RGBCOLORMAP REDBITS GREENBITS BLUEBITS BITSPERPIXEL)** [Function]

Returns a color map that assumes the *BITSPERPIXEL* bits are to be treated as three separate color planes with *REDBITS* bits being in the red plane, *GREENBITS* bits being in the green plane, and *BLUEBITS* bits being in the blue plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 255 and black is 0.

**(GRAYCOLORMAP BITSPERPIXEL)** [Function]

Returns a color map containing shades of gray. White is 0 and black is 255.

**(COLORMAPCREATE INTENSITIES BITSPERPIXEL)** [Function]

creates a color map for a screen that has *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is NIL, the number of bits per pixel is taken from the current color display setting. *INTENSITIES* specifies the initial colors that should be in the map. If *INTENSITIES* is not NIL, it should be a list of color specifications other than color numbers, e.g., the list of RGB triples returned by the function *INTENSITIESFROMCOLORMAP*.

**(INTENSITIESFROMCOLORMAP COLORMAP)** [Function]

returns a list of the intensity levels of *COLORMAP* (default is *SCREENCOLORMAP*) in a form accepted by *COLORMAPCREATE*. This list can be written on file and thus provides a way of saving color map specifications.

**(COLORMAPCOPY COLORMAP BITSPERPIXEL)** [Function]

returns a color map that contains the same color intensities as *COLORMAP* if *COLORMAP* is a color map. Otherwise, it returns a color map with default color values.

**(MAPOFACOLOR PRIMARIES)** [Function]

returns a color map that is different shades of one or more of the primary colors. For example, *(MAPOFACOLOR '(RED GREEN BLUE))* gives a color map of different shades of gray; *(MAPOFACOLOR 'RED)* gives different shades of red.

## Changing Color Maps

The following functions are provided to access and change the intensity levels in a color map.

**(SETCOLORINTENSITY *COLORMAP* *COLORNUMBER* *COLORSPEC*)** [Function]

sets the primary intensities of color number *COLORNUMBER* in the color map *COLORMAP* to the ones specified by *COLORSPEC*. *COLORSPEC* can be either an RGB triple, an HLS triple, or a color name. The value returned is NIL.

**(COLORLEVEL *COLORMAP* *COLORNUMBER* *PRIMARY* *NEWLEVEL*)** [Function]

sets and reads the intensity level of the primary color *PRIMARY* (RED, GREEN, or BLUE) for the color number *COLORNUMBER* in the color map *COLORMAP*. If *NEWLEVEL* is a number between 0 and 255, it is set. The previous value of the intensity of *PRIMARY* is returned.

**(ADJUSTCOLORMAP *PRIMARY* *DELTA* *COLORMAP*)** [Function]

adds *DELTA* to the intensity of the *PRIMARY* color value (RED, GREEN, or BLUE) for every color number in *COLORMAP*.

**(ROTATECOLORMAP *STARTCOLOR* *THRUCOLOR*)** [Function]

rotates a sequence of colors in the SCREENCOLORMAP. The rotation moves the intensity values of color number *STARTCOLOR* into color number *STARTCOLOR*+1, the intensity values of color number *STARTCOLOR*+1 into color number *STARTCOLOR*+2, etc., and *THRUCOLOR*'s values into *STARTCOLOR*.

**(EDITCOLORMAP *VAR* *NOQFLG*)** [Function]

allows interactive editing of a color map. If *VAR* is an atom whose value is a color map, its value is edited. Otherwise a new color map is created and edited. The color map being edited is made the screen color map while the editing takes place so that its effects can be observed. The edited color map is returned as the value. If *NOQFLG* is NIL and the color display is on, you are asked if you want a test pattern of colors. A yes response causes the function SHOWCOLORTESTPATTERN to be called, which displays a test pattern with blocks of each of the possible colors.

You are prompted for the location of a color control window to be placed on the black-and-white display. This window allows the value of any of the colors to be changed. The number of the color being edited is in the upper left part of the window. Six bars are displayed. The right three bars give the color intensities for the three primary colors of the current color number. The left three bars give the value of the color's Hue, Lightness, and Saturation parameters. These levels can be changed by positioning the mouse cursor in one of the bars and pressing the left mouse button. While the left button is down, the value of that parameter tracks the Y position of the cursor. When the left button is released, the color tracking stops. The color being edited is changed by pressing the middle mouse button while the cursor is in the interior of the edit window. This brings up a menu of color numbers. Selecting one sets the current color to the selected color.

The color being edited can also be changed by selecting the menu item "PickPt." This switches the cursor onto the color screen and allows you to select a point from the color screen. It then edits the color of the selected point.

To stop the editing, move the cursor into the title of the editing window and press the middle button. This brings up a menu. Select Stop to quit.

---

## Color Bitmaps

---

A color bitmap is actually a bitmap that has more than one bit per pixel. To test whether a bitmap is a color bitmap, the function `BITSPERPIXEL` can be used.

**(BITSPERPIXEL *BITMAP*)** [Function]

returns the bits per pixel of *BITMAP*; if this does not equal one, *BITMAP* is a color bitmap.

In multiple-bit-per-pixel bitmaps, the bits that represent a pixel are stored contiguously. `BITMAPCREATE` is passed a *BITSPERPIXEL* argument to create multiple-bit-per-pixel bitmaps.

**(BITMAPCREATE *WIDTH HEIGHT BITSPERPIXEL*)** [Function]

creates a color bitmap that is *WIDTH* pixels wide by *HEIGHT* pixels high allowing *BITSPERPIXEL* bits per pixel. Currently any value of *BITSPERPIXEL* except one, four, eight, or NIL (defaults to one) causes an error.

A four-bit-per-pixel color screen bitmap uses approximately 76K words of storage, and an eight-bit-per-pixel one uses approximately 153K words. There is only one such bitmap. The following function provides access to it.

**(COLORSCREENBITMAP)** [Function]

returns the bitmap that is being or will be displayed on the color display. This is NIL if the color display has never been turned on (see `COLORDISPLAY` below).

---

## Screens, Screenpositions, and Screenregions

---

In addition to positions and regions, the user needs to be aware of screens, screenpositions, and screenregions in the presence of multiple screens.

---

### Screens

---

**SCREEN** [Datatype]

There are generally two screen datatype instances in existence when working with color. This is because the user is attached to two displays, a black and white display and a color display.

**(MAINSCREEN)** [Function]

returns the screen datatype instance that represents the black and white screen. This will be something like `{SCREEN}#74,24740`.

**(COLORSCREEN)** [Function]

returns the screen datatype instance that represents the color screen. Screens appear as part of screenpositions and screenregions, serving as the extra information needed to make

clear whether a particular position or region should be viewed as lying on the black and white display or the color display.

**(SCREENBITMAP SCREEN)** [Function]

returns the bitmap destination of *SCREEN*. If *SCREEN*=NIL, returns the black and white screen bitmap.

## Screenpositions

**SCREENPOSITION** [Record]

Somewhat like a position, a screenposition denotes a point in an X,Y coordinate system on a particular screen. Screenpositions have been defined according to the following record declaration:

(RECORD SCREENPOSITION (SCREEN . POSITION)

(SUBRECORD POSITION))

A SCREENPOSITION is an instance of a record with fields XCOORD, YCOORD, and SCREEN and is manipulated with the standard record package facilities. For example, (create SCREENPOSITION XCOORD \_ 10 YCOORD \_ 20 SCREEN \_ (COLORSCREEN)) creates a screenposition representing the point (10,20) on the color display. The user can extract the position of a screenposition by fetching its POSITION. For example, (fetch (SCREENPOSITION POSITION) of SP12).

## Screenregions

**SCREENREGION** [Record]

Somewhat like a region, a screenregion denotes a rectangular area in a coordinate system. Screenregions have been defined according to the following record declaration:

(RECORD SCREENREGION (SCREEN . REGION)

(SUBRECORD REGION))

Screenregions are characterized by the coordinates of their bottom left corner and their width and height. A SCREENREGION is a record with fields LEFT, BOTTOM, WIDTH, HEIGHT, and SCREEN. It can be manipulated with the standard record package facilities. There are access functions for the REGION record that return the TOP and RIGHT of the region. The user can extract the region of a screenregion by fetching its REGION. For example, (fetch (SCREENREGION REGION) of SR8).

## Screenposition and Screenregion Prompting

The following functions can be used by programs to allow the user to interactively specify screenpositions or screenregions on a display screen.

**(GETSCREENPOSITION WINDOW CURSOR)** [Function]

Similar to GETPOSITION. Returns a SCREENPOSITION that is specified by the user. GETSCREENPOSITION waits for the user to press and release the left button of the mouse and returns the cursor screenposition at the time of release. If *WINDOW* is a WINDOW, the screenposition will be on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position will be in screen coordinates.



---

**(GETBOXSCREENPOSITION BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG)** [Function]

---

Similar to GETBOXPOSITION. Returns a SCREENPOSITION that is specified by the user. Allows the user to position a "ghost" region of size *BOXWIDTH* by *BOXHEIGHT* on a screen, and returns the SCREENPOSITION of the lower left corner of the screenregion chosen. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. The user can change to another corner by holding down the right button. With the right button down, the cursor can be moved across a screen or to other screens without effect on the ghost region frame. When the right button is released, the mouse will snap to the nearest corner, which will then become locked to the cursor. (The held corner can be changed after the left or middle button is down by holding both the original button and the right button down while the cursor is moved to the desired new corner, then letting up just the right button.) When the left or middle button is pressed and released, the lower left corner of the screenregion chosen at the time of release is returned. If *WINDOW* is a WINDOW, the screenposition will be on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position will be in screen coordinates.its lower left corner in screen coordinates.

---

**(GETSCREENREGION MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS)** [Function]

---

Similar to GETREGION. Returns a SCREENREGION that is specified by the user. Lets the user specify a new screenregion and returns that screenregion. GETSCREENREGION prompts for a screenregion by displaying a four-pronged box next to the cursor

arrow at one corner of a "ghost" region:  If the user presses the left button, the corner of a "ghost" screenregion opposite the cursor is locked where it is. Once one corner has been fixed, the ghost screenregion expands as the cursor moves.

To specify a screenregion: (1) Move the ghost box so that the corner opposite the cursor is at one corner of the intended screenregion. (2) Press the left button. (3) Move the cursor to the screenposition of the opposite corner of the intended screenregion while holding down the left button. (4) Release the left button.

Before one corner has been fixed, one can switch the cursor to another corner of the ghost screenregion by holding down the right button. With the right button down, the cursor changes to a

 "forceps"  and the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor will snap to the nearest corner of the ghost screenregion.

After one corner has been fixed, one can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor will snap to the nearest corner, which will become the moving corner. In this way, the screenregion may be moved all over a screen and to other screens, before its size and screenposition is finalized.

The size of the initial ghost screenregion is controlled by the *MINWIDTH*, *MINHEIGHT*, *OLDREGION*, and *INITCORNERS* arguments.

---

---

**(GETBOXSCREENREGION** *WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG*) [Function]

---

Similar to GETBOXREGION. Returns a SCREENREGION that is specified by the user. Performs the same prompting as GETBOXSCREENPOSITION and returns the SCREENREGION specified by the user instead of the SCREENPOSITION of its lower left corner.

---



---

## Color Windows and Menus

---

The Xerox Lisp window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion (see page X.XX). Menus are a special type of window provided by the window system, used for displaying a set of items to the user, and having the user select one using the mouse and cursor. The menu facility also allows users to create and use menus in interactive programs (see page X.XX). As of the LUTE release of Xerox Lisp, it is possible for the user to create and use windows and menus on the color display.

**(CREATEW** *REGION TITLE BORDERSIZE NOOPENFLG*) [Function]

---

Creates a new window. *REGION* indicates where and how large the window should be by specifying the exterior screenregion of the window. In a user environment with multiple screens, such as a black and white screen and color screen both connected to the same machine, there is a new special problem in indicating which screen the *REGION* is supposed to be a region of. This problem is solved by allowing CREATEW to take screenregion arguments as *REGION*. For example,

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
 SCREEN _
 (COLORSCREEN)
 LEFT _ 20
 BOTTOM _ 210
 WIDTH _ 290
 HEIGHT _ 170)
 "FOO WINDOW"))
```

creates a window titled "FOO WINDOW" on the color screen. To create a window on the black and white screen, the user should use SCREEN \_ (MAINSCEEN) in the CREATE SCREENREGION expression. Note that it is still perfectly legal to pass in a *REGION* that is a region, not a screenregion, to CREATEW, but it is preferable to be passing screenregions rather than regions to CREATEW. This is because when *REGION* is a region, *REGION* is disambiguated in a somewhat arbitrary manner that may not always turn out to be what the user was hoping for.

When *REGION* is a region, *REGION* is disambiguated by coercing *REGION* to be a screenregion on the screen which currently contains the cursor. This is so that software calling CREATEW with regions instead of screenregions tends to do the right thing in a user environment with multiple screens.

---

**(WINDOWPROP WINDOW PROP NEWVALUE)**

[NoSpread Function]

If *PROP*='SCREEN', then WINDOWPROP returns the screen *WINDOW* is on. If *NEWVALUE* is given, (even if given as NIL), with *PROP*='SCREEN', then WINDOWPROP will generate an error. Any other *PROP* name is handled in the usual way.

---

**(OPENWINDOWS SCREEN)**

[Function]

Returns a list of all open windows on *SCREEN* if *SCREEN* is a screen datatype such as (MAINSCREEN) or (COLORSCREEN). If *SCREEN*=NIL, then *SCREEN* will default to the screen containing the cursor. If *SCREEN*=T, then a list of all open windows on all screens is returned.

---

---

## Color Fonts

---

The user can create color fonts and specify in the font profile that certain color fonts be used when printing in color.

---

### Color Font Creation

---

The user can create and manipulate color fonts through the same functions that are used to create and manipulate black and white fonts. This is made possible in some cases by there being new ways to call familiar font functions.

**(FONTCREATE FAMILY SIZE FACE ROTATION DEVICE NOERRORFLG CHARSET)** [Function]

In addition to creating black and white fonts, FONTCREATE can be used to create color fonts. For example,

```
(FONTCREATE 'GACHA 10
 '(BOLD REGULAR REGULAR YELLOW
 BLUE)
 0 '8DISPLAY)
```

will create an 8 bit per pixel font with blue letters on a yellow background. The user indicates the color and bits per pixel of the font by the *FACE* and *DEVICE* arguments passed to FONTCREATE. *DEVICE*='8DISPLAY means to create an 8bpp font and *DEVICE*='4DISPLAY means to create a 4bpp font. A color font face is a 5 tuple,

(WEIGHT SLOPE EXPANSION BACKCOLOR FORECOLOR)

whereas a black and white font face is just a 3 tuple,

(WEIGHT SLOPE EXPANSION)

The FORECOLOR is the color of the characters of the font and the BACKCOLOR is the color of the background behind the characters that gets printed along with the characters. Both BACKCOLOR and FORECOLOR are allowed to a color name, color number, or any other legal color representation. A color font face can also be represented as a LITATOM. A three character atom such as MRR or any of the special atoms STANDARD, ITALIC, BOLD, BOLDITALIC can optionally be continued by hyphenating on BACKCOLOR and FORECOLOR suffixes. For example,

MRR-YELLOW-BLUE

BOLD-YELLOW-RED

ITALIC-90-200

BRR-100-53

are acceptable color font faces. Hence,

```
(FONTCREATE 'GACHA 10 'BOLD-YELLOW-BLUE 0
'8DISPLAY)
```

will create a color font. LITATOM FACE arguments fall into one of the following patterns:

|            |                                |
|------------|--------------------------------|
| wse        | wse-backcolor-forecolor        |
| STANDARD   | STANDARD-backcolor-forecolor   |
| ITALIC     | ITALIC-backcolor-forecolor     |
| BOLD       | BOLD-backcolor-forecolor       |
| BOLDITALIC | BOLDITALIC-backcolor-forecolor |

where w=B, M, or L; s=I or R; e=R, C, or E; backcolor=a color name or color number; and forecolor=a color name or color number.

| <b>(FONTPROP FONT PROP)</b> | [Function]                                                                                                                                                                                                                                                                         |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                             | Returns the value of the <i>PROP</i> property of font <i>FONT</i> . Besides black and white font properties, the following font properties are recognized:                                                                                                                         |
| <b>FORECOLOR</b>            | The color of the characters of the font, represented as a color number. This is the color in which the characters of the font will print.                                                                                                                                          |
| <b>BACKCOLOR</b>            | The color of the background of the characters of the font, represented as a color number. This is the color in which the the background of characters of the font will print. A font with red characters on a yellow background would have a red FORECOLOR and a yellow BACKCOLOR. |

## Color Font Profiles

Font profiles are the facility PRETTYPRINT uses to print different elements (user functions, system functions, clisp words, comments, etc.) in different fonts to emphasize (or deemphasize) their importance, and in general to provide for a more pleasing appearance. The user can specify that different colors of fonts be used for different kinds of elements when printing in color. A well chosen font profile will allows user to DEDIT functions, PP functions, and SEE source files in color, for example. A **FONTPROFILE** such as

```
((DEFAULTFONT 1 (GACHA 10)
 (GACHA 8)
 (TERMINAL 8)
 (4DISPLAY (GACHA 10 MRR-WHITE-RED))
 (8DISPLAY (GACHA 10 MRR-WHITE-RED)))
 (BOLDFONT 2 (HELVETICA 10 BRR)
 (HELVETICA 8 BRR)
 (MODERN 8 BRR)
 (4DISPLAY (HELVETICA 10 BRR-WHITE-MAGENTA)))
```

```
(8DISPLAY (HELVETICA 10 BRR-WHITE-MAGENTA)))
(LITTLEFONT 3 (HELVETICA 8)
 (HELVETICA 6 MIR)
 (MODERN 8 MIR)
 (4DISPLAY (HELVETICA 8 MRR-WHITE-GREEN))
 (8DISPLAY (HELVETICA 8 MRR-WHITE-GREEN)))
(BIGFONT 4 (HELVETICA 12 BRR)
 (HELVETICA 10 BRR)
 (MODERN 10 BRR)
 (4DISPLAY (HELVETICA 12 BRR-WHITE-BLUE))
 (8DISPLAY (HELVETICA 12 BRR-WHITE-BLUE)))
(USERFONT BOLDFONT)
(COMMENTFONT LITTLEFONT)
(LAMBDAFONT BIGFONT)
(SYSTEMFONT)
(CLISPFONT BOLDFONT)
...)
```

would have comments print in green and clisp words print in blue while ordinary atoms would print in red.

Not all combinations of fonts will be aesthetically pleasing and the user may have to experiment to find a compatible set.

The user should indicate what font is to be used for each font class by calling the function FONTPROFILE:

| <b>(FONTPROFILE PROFILE)</b> | [Function]                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                              | Sets up the font classes as determined by <i>PROFILE</i> , a list of elements which defines the correspondence between font classes and specific fonts. Each element of <i>PROFILE</i> is a list of the form:<br><b>(FONTCLASS FONT# DISPLAYFONT PRESSFONT INTERPRESSFONT (OTHERDEVICE1 OTHERFONT1) ... (OTHERDEVICE<sub>n</sub> OTHERFONT<sub>n</sub>))</b><br><i>FONTCLASS</i> is the font class name and <i>FONT#</i> is the font number for that class. <i>DISPLAYFONT</i> , <i>PRESSFONT</i> , and <i>INTERPRESSFONT</i> are font specifications (of the form accepted by FONTCREATE) for the fonts to use when printing to the black and white display and to Press and Interpress printers respectively. The appearance of color fonts can be affected by including an <b>(OTHERDEVICE<sub>i</sub> OTHERFONT<sub>i</sub>)</b> entry where <i>OTHERDEVICE<sub>i</sub></i> is either 4DISPLAY or 8DISPLAY for a 4 bits per pixel or 8 bits per pixel color font and <i>OTHERFONT<sub>i</sub></i> is a color font specification such as (GACHA 10 MRR-WHITE-RED). |

| <b>FONTPROFILE</b> | [Variable]                                                                                                                                                                                                                                                                                                                |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    | This is the <i>variable</i> used to store the current font profile, in the form accepted by the <i>function</i> FONTPROFILE. Note that simply editing this value will not change the fonts used for the various font classes; it is necessary to execute (FONTPROFILE FONTPROFILE) to install the value of this variable. |

---

## Using Color

---

The current color implementation allows display streams to operate on color bitmaps. The two functions `DSPCOLOR` and `DSPBACKCOLOR` set the color in which a stream draws when the user defaults a color argument to a drawing function.

**(`DSPCOLOR` *COLOR* *STREAM*)** [Function]

sets the foreground color of a stream. It returns the previous foreground color. If *COLOR* is `NIL`, it returns the current foreground color without changing anything. The default foreground color is `MINIMUMCOLOR=0`, which is white in the default color maps.

**(`DSPBACKCOLOR` *COLOR* *STREAM*)** [Function]

sets the background color of a stream. It returns the previous background color. If *COLOR* is `NIL`, it returns the current background color without changing anything. The default background color is `(MAXIMUMCOLOR BITS PER PIXEL)=15` or 255, which is black in the default color maps.

The `BITBLT`, line-drawing routines, and curve-drawing routines know how to operate on a color-capable stream. Following are some notes about them.

---

## BITBLTing in Color

---

If `BITBLT`ing from a color bitmap onto another color bitmap of the same bpp, the operations `PAINT`, `INVERT`, and `ERASE` are done on a bit level, not on a pixel level. Thus painting color 3 onto color 10 results in color 11.

If `BITBLT`ing from a black-and-white bitmap onto a color bitmap, the one bits appear in the `DSPCOLOR`, and the zero bits in `DSPBACKCOLOR`. `BLT`ing from black-and-white to color is fairly expensive; if the same bitmap is going to be put up several times in the same color, it is faster to create a color copy and then `BLT` the color copy.

If the source type is `TEXTURE` and the destination bitmap is a color bitmap, the *Texture* argument is taken to be a color. Thus to fill an area with the color `BLUE` assuming `COLORSTR` is a stream whose destination is the color screen, use `(BITBLT NIL NIL NIL COLORSTR 50 75 100 200 'TEXTURE 'REPLACE 'BLUE)`.

---

## Drawing Curves and Lines in Color

---

For the functions `DRAWCIRCLE`, `DRAWELLIPSE`, and `DRAWCURVE`, the notion of a brush has been extended to include a color. A `BRUSH` is now `(BRUSHSHAPE BRUSHSIZE BRUSHCOLOR)`. Also, a brush can be a bitmap (which can be a color bitmap).

Line-drawing routines take a color argument which is the color the line is to appear in if the destination of the display stream is a color bitmap.

**(DRAWLINE** *X1 Y1 X2 Y2 WIDTH OPERATION*  
*STREAM COLOR*) [Function]

**(DRAWTO** *X Y WIDTH OPERATION STREAM COLOR*) [Function]

**(RELDRAWTO** *X Y WIDTH OPERATION*  
*STREAM COLOR*) [Function]

**(DRAWBETWEEN** *POS1 POS2 WIDTH OPERATION*  
*STREAM COLOR*) [Function]

If the *COLOR* argument is NIL, the DSPCOLOR of the stream is used.

---

## Printing in Color

---

Printing only works in REPLACE mode. The characters have a background color and a foreground color determined by the font face of the font the characters are being printed with.

Example of printing to an 8bpp color screen:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
 SCREEN _
 (COLORSCREEN)
 LEFT _ 20
 BOTTOM _ 210
 WIDTH _ 290
 HEIGHT _ 170)
 "FOO WINDOW"))
(DSPFONT (FONTCREATE 'GACHA
 10
 'MRR-YELLOW-GREEN
 0
 '8DISPLAY)
 FOO)
(PRINT 'HELLO FOO) ; will print in green against a yellow
background.
```

---

## Operating the Cursor on the Color Screen

---

The cursor can be moved to the color screen. The cursor can be moved to the color screen by sliding the cursor off the left or right edge of the black and white screen on to the color screen or by calling function `CURSORPOSITION` or `CURSORSCREEN`.

**(CURSORPOSITION NEWPOSITION - -)** [Function]

---

*NEWPOSITION* can be a position or a screenposition.

**(CURSORSCREEN SCREEN XCOORD YCOORD)** [Function]

---

Moves the cursor to the screenposition determined by *SCREEN*, *XCOORD*, and *YCOORD*. *SCREEN* should be the value of either (COLORSCREEN) or (MAINSCEEN).

While on the color screen, the cursor is placed by doing BITBLTs in software rather than with microcode and hardware as with the black and white cursor. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. The speed of the color cursor compares well with that of the black and white cursor but there can be a noticeable flicker when there is much input/output to the color screen. While the cursor is on the color screen, the black-and-white cursor is cleared giving the appearance that there is never more than one cursor at a given time.

---

## Miscellaneous Color Functions

---

**(COLORIZEBITMAP BITMAP 0COLOR 1COLOR BITSPEPIXEL)**[Function]

creates a color bitmap from a black and white bitmap. The returned bitmap has color number *1COLOR* in those pixels of *BITMAP* that were one and *0COLOR* in those pixels of *BITMAP* that were zero. This provides a way of producing a color bitmap from a black and white bitmap.

**(UNCOLORIZEBITMAP BITMAP COLORMAP)** [Function]

creates a black and white bitmap from a color bitmap.

**(SHOWCOLORTESTPATTERN BARSIZE)** [Function]

displays a pattern of colors on the color display. This is useful when editing a color map. The pattern has squares of the 16 possible colors laid out in two rows at the top of the screen. Colors 0 through 7 are in the top row, and colors 8 through 15 are in the next row. The bottom part of the screen is filled with bars of *BARSIZE* width with consecutive color numbers. The pattern is designed so that every color has a border with every other color (unless *BARSIZE* is too large to allow room for every color—about 20).



# COPYFILES

---

CopyFiles makes it easy to copy or move groups of files from one place to another.

## Installation

---

Load COPYFILES.LCOM from the library.

## Function

---

(COPYFILES *SOURCE DESTINATION OPTIONS*) [Function]

Copies the files designated by *SOURCE* to the place designated by *DESTINATION*.

*SOURCE* is a pattern such as given to DIRECTORY or DIR; it can also be a list of file names.

*DESTINATION* is either a directory name or a file-name pattern, with a one-to-one match of the wild card characters (\*s) in *DESTINATION* to \*s in *SOURCE*. The number of \*s in each source pattern needs to match the number of \*s in each destination pattern. (See examples below.)

Note: You must use \*s if you want to name files. If you do not, the COPYFILES code assumes that any *DESTINATION* specified without \*s in it is a destination directory.

*OPTIONS* is a list (if you have only one and it is a symbol, you can supply it as a symbol) that may include one or more of the options specified below.

Note: If the destination is a non-existent NS subdirectory, COPYFILES asks whether it should create it. If you answer YES, then it creates the subdirectory. If you answer NO, it aborts without processing any files.

### OPTION: Conversation Mode

You can specify how verbose CopyFiles is about what it is doing:

QUIET Don't print anything while working.

(OUTPUT *LISTFILE*) Print the name of each file that gets copied on *LISTFILE*.  
(OUTPUT T) is the default.

TERSE Only print a period (.) for each file moved/copied.

### OPTION: Query Mode

You can specify whether CopyFiles should ask for confirmation before each transfer.

ASK Ask each time before moving/copying a file (default is to not ask).

(ASK N) Ask, with default to No after DWIMWAIT seconds.

(ASK Y) Ask, with default to Yes after DWIMWAIT seconds.

### **OPTION: Version Control**

CopyFiles normally uses the Lisp function COPYFILE to create a new file. It also usually copies only the highest version, and creates a new version at the destination. Alternatively, you can specify any of the following:

RENAME or MOVE    Use RENAMEFILE instead of COPYFILE; i.e., the source is deleted afterwards.

ALLVERSIONS    Copy all versions and preserve version numbers.

REPLACE    If a file by the same name exists on the destination, overwrite it (don't create a new version).

Note: When \* is used as the source version number, be sure to specify ALLVERSIONS. This is important because some devices list files by version number from highest to lowest, while by default the version numbers at the destination are assigned in ascending order. Hence, if ALLVERSIONS is not specified, the versions may be reversed, as can be verified by looking at the creation dates.

### **OPTION: When To Copy**

CopyFiles normally compares the creation dates of the file on the source and any matching file on the destination to determine whether it is necessary to copy. The following options are mutually exclusive:

ALWAYS    Always copy the file.

>    Copy only when a file by the same name but an earlier creation date exists on the destination.

>A    Similar to >, but also copy if the file doesn't exist on the destination; i.e., > ALWAYS.

#    Copy only when a file by the same name but a different creation date exists on the destination.

#A    Similar to #, but also copy if the file doesn't exist on the destination, i.e., # ALWAYS.

=A    Copy only if there isn't a file of the same name on the destination.

Not all combinations of options make sense; for example, ALLVERSIONS probably doesn't work right with any date comparison algorithms.

The default setting is (>A); that is, copy the highest version if it doesn't exist on the destination or if an older creation date exists, and print out messages about all files considered.

### **OPTION: Clean-Up After Copying Files**

CopyFiles can be instructed to delete some files after it has finished copying.

- PURGE** This involves a separate pass (afterwards): any file on the destination which doesn't have a counterpart on the source is deleted.
- PURGESOURCE** Converse of **PURGE** (and used by it): if the file is on the source and not on the destination, delete it.

## Limitations

The creation date comparison does not work when either the source or the destination does not support creation dates. For example, the TCP-IP protocol doesn't support any way to find out the creation date of a remote file. For this reason, COPYFILES can only be used in **ALWAYS** mode when using a TCP-IP protocol.

## Examples

```
(COPYFILES ' {ERIS}<USER>*.MAIL ' {PHYLUM}<USER>OLD-*.MAIL)
```

Copies any mail file on {ERIS}<USER> to {PHYLUM}<USER>, copying FOO.MAIL to OLD-FOO.MAIL.

```
(COPYFILES ' {ERIS}<USER>*.MAIL ' {PHYLUM}<USER>OLD-*.MAIL ' RENAME)
```

Uses **RENAMEFILE** instead.

```
(COPYFILES ' ({DSK}TEST {DSK}WEST) ' {PHYLUM}<MYDIR>)
```

Copies the files **TEST** and **WEST** from {DSK} to {PHYLUM}<MYDIR>.

```
(COPYFILES ' {PHYLUM}<USER>*.AR ' {PHYLEX:}<USER> '=A)
```

Copies all ARs on {PHYLUM}<USER> to the **PHYLEX NS** file server; if any are already there, it won't bother copying them.

```
(COPYFILES ' {PHYLUM}<USER>AR.INDEX ' {DSK}AR.INDEX ' (>A REPLACE))
```

Copies the AR index to {DSK}, replacing any older version that is already there.

```
COPYFILES ({DSK}*.; {FLOPPY})
```

Copies all files on {DSK} that have no file name extensions to {FLOPPY}.

```
(COPYFILES ' {ERIS}<USER> ' {PHYLUM}<USER> ' (#A PURGE))
```

Makes {PHYLUM}<USER> look like {ERIS}<USER>, bringing over any file that isn't already on {PHYLUM} and then deleting the ones that were on {PHYLUM} and aren't on {ERIS} any more.

[This page intentionally left blank]

# DATABASEFNS

---

DataBaseFns makes the construction and maintenance of MasterScope database essentially an automatic process.

DataBaseFns modifies the behavior of the Lisp functions MAKEFILE, LOAD and LOADFROM, such that writing out a file also updates and saves a MasterScope database, and loading the file also loads the database for you to use.

For example,

```
(LOAD 'FILE1)
```

loads FILE1, then looks for the corresponding database file FILE1.DATABASE. If the database exists, it is also loaded. The result is the same as if you had typed:

```
(LOAD 'FILE1)
.ANALYZE ALL ON FILE1
```

The database is maintained automatically for any file (containing functions) whose file name has the property DATABASE with value YES. Whenever such a file is dumped via MAKEFILE, MasterScope analyzes any new or changed functions on the file, and a database for all of the functions on the file is written on a separate file whose name is of the form FILE.DATABASE. Whenever a file that has a database property with value YES is loaded via LOAD or LOADFROM, then the corresponding database file, if any, is also loaded. The database is not dumped or loaded if the value of the DATABASE property for the file is NO. The DATABASE property is considered to be NO if the file is loaded with LDFLG=SYSLOAD.

If you change some of the functions defined in FILE1, and perhaps add new ones, then do:

```
(MAKEFILE 'FILE1)
```

Then FILE1 is written out. MasterScope analyzes all changed or new functions and writes FILE1.DATABASE, which contains MasterScope data for all functions on FILE1.

## Requirements

---

MASTERSCOPE

## Installation

---

Load MASTERSCOPE.DFASL from the library, then load DATABASEFNS.LCOM.

## User Interface

---

If DataBaseFns is loaded, the first LOAD of a file will ask if you want to load the corresponding database:

```
(LOAD 'FILE1)
Do you want to load the data base for FILE1?
```

And MAKEFILEing a new file will ask if you want to save the database:

```
(MAKEFILE 'FILE1)
Save the data base for FILE1?
```

Once you tell the system YES or NO for a particular file, it will remember and will not ask you again (until you load a new sysout).

## Functions

---

You can set up default answers to these questions by means of the following variables:

LOADDBFLG [Variable]

This controls whether you are asked before the database file is loaded:

Yes Always try to load the database when a file is loaded.  
No Never try to load the database when a file is loaded.  
Ask (default) Ask whether to load the database when a file is loaded.

SAVEDBFLG [Variable]

This controls whether you are asked before the database file is saved:

Yes Always try to save the database when a file is saved.  
No Never try to save the database when a file is saved.  
Ask (default) Ask whether to save the database when a file is saved.

(DUMPDB *FILE* *PROPFLG*) [Function]

Dumps a database for *FILE* then sets the *DATABASE* property to YES, so that database maintenance for *FILE* will subsequently be automatic.

(LOADDB *FILE* *ASKFLG*) [Function]

Loads the file *FILE*.*DATABASE* if one exists. After the database is loaded, the *DATABASE* property for *FILE* is set to YES, so that maintenance is automatic thereafter.

Database files include the date and full file name of the file to which they correspond. LOADDB prints out a warning message if it loads a database that does not correspond to the in-core version of the file, and asks you if you approve.

Note: LOADDB is the only approved way of loading a database. Attempting to LOAD a database file will cause an error.

[This page intentionally left blank]

# DEDIT

---

Many important objects such as function definitions, property lists, and variable values are represented as list structures. There are two list structure editors (SEdit in the system, and DEdit in the library, for backward compatibility) to allow users to modify list structures rapidly and conveniently.

## Description

---

The list structure editor is most often used to edit function definitions. Editing function definitions in memory is a facility not offered by many Lisp systems, where typically the user edits external text files containing function definitions, then loads them into the environment. In Lisp, function definitions are edited in the environment, and written to an external file using the file manager (see *IRM*), which provides tools for managing the contents of a file.

## History

Early implementations of Interlisp using primitive terminals offered a teletype-oriented editor, which included a large set of cryptic commands for printing different parts of a list structure, searching a list, replacing elements, etc. The library includes an extended, display-oriented version of the teletype list structure editor, called DEdit.

The teletype editor is still available (see *IRM*), as it offers a facility for doing complex modifications of program structure under program control. DEdit also provides facilities for using the teletype editor commands from within DEdit.

## DEdit

DEdit is a structure oriented, modeless, display based editor for objects represented as list structures, such as functions, property lists, data values, etc.

DEdit incorporates the interfaces of the teletype-oriented Interlisp editor, so the two can be used interchangeably. In addition, the full power of the teletype editor, and indeed the full Interlisp system is easily accessible from within DEdit.

DEdit is structure-oriented rather than character-oriented, to facilitate selecting and operating on pieces of structure as objects in their own right, rather than as collections of characters. However, for the occasional situation when character-oriented editing is appropriate, DEdit provides access to the text editing facilities. DEdit is modeless, in that all commands operate on previously selected arguments, rather than causing the behavior of the interface to change during argument specification.

## Requirements

---

DEDITPP

## Installation

---

Load DEDIT.LCOM from the library.



Loading DEDit makes it the default Lisp structure editor.

If another Lisp structure editor is already the default and you want to make DEDit the default, call (EDITMODE 'DEDIT) after loading DEDit.

## User Interface

---

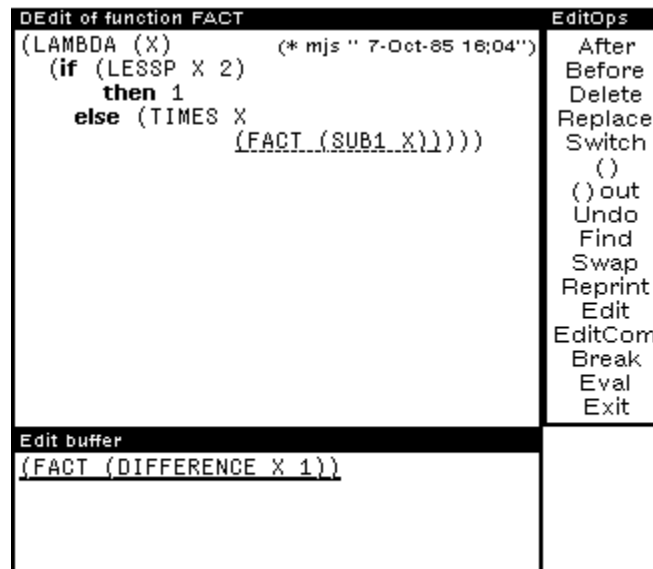
DEDit is normally called using one of the DEDit functions. See also "Advanced Features" below.

### DEDit Window

When DEDit is called for the first time, it prompts for an edit window which is preserved and reused for later DEDits, and it pretty-prints the expression to be edited therein.

Note: The DEDit pretty printer ignores user PRETTYPRINTMACROS because they do not provide enough structural information during printing to enable selection.

The expression being edited can be scrolled by using the standard scroll bar on the left edge of the window. DEDit adds a command menu, which remains active throughout the edit, on the right edge of the edit window. If you type anything, an edit buffer window is positioned below the edit window. This is illustrated in the figure below, which shows the definition of a function called FACT. While DEDit is running, it yields control so that background activities, such as mouse commands in other windows, continue to be performed.



### Selecting Objects and Lists

Selection in a DEDit window is as follows:

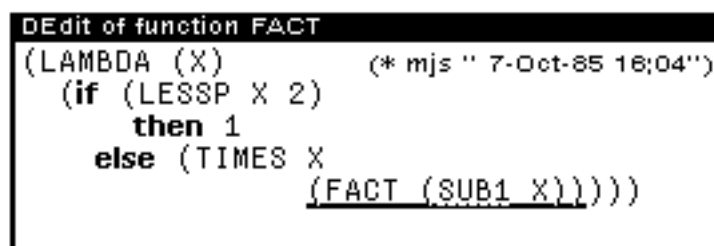
- The left button selects the object being directly pointed at.
- The middle button selects the containing list.

- The right button extends the current selection to the lowest common ancestor of that selection and the current position.

The only things that may be pointed at are atomic objects (symbols, numbers, etc.) and parentheses, which are considered to represent the list they delimit. White space cannot be selected or edited.

When a selection is made, it is pushed on a selection stack, which will be the source of operands for DEdit commands. As each new selection pushes down the selections made before it, this stack can grow arbitrarily deep, so only the top two selections on the stack are highlighted on the screen. This highlighting is done by underscoring the topmost (most recent) selection with a solid black line and the second topmost selection with a dashed line. The patterns used were chosen so that their overlappings would be both visible and distinct, since selecting a subpart of another selection is quite common.

For example, in the next figure, the last selection is the list (FACT (SUB1 X)), and the previous selection is the single symbol SUB1:



```

DEdit of function FACT
(LAMBDA (X) (* mjs " 7-Oct-85 16:04")
 (if (LESSP X 2)
 then 1
 else (TIMES X (FACT (SUB1 X)))))

```

Because you can invoke DEdit recursively, there may be several DEdit windows active on the screen at once. This is often useful when transferring material from one object to another (as when reallocating functionality within a set of programs). Selections may be made in any active DEdit window, in any order. When there is more than one DEdit window, the edit command menu (and the type-in buffer) attaches itself to the most recently opened (or current) DEdit window.

## Typing Characters to DEdit

Characters may be typed at the keyboard at any time. This creates a type-in buffer window which positions itself under the current DEdit window and does a LISPXREAD (which must be terminated by a right parenthesis or a return) from the keyboard. During the read, any character-editing subsystem (such as TTYIN) that is loaded can be used to do character-level editing on the type-in. When the read is complete, the type-in becomes the current selection (top of stack) and is available as an operand for the next command. Once the read is complete, objects displayed in the type-in buffer can be selected from, scrolled, or even edited, just like those in the main window.

You can also enter editing commands directly into the type-in buffer. Typing Control-Z interprets the rest of the line as a teletype editor command that is interpreted when the line is closed. Likewise, Control-S OLD NEW substitutes NEW for OLD, and Control-F X finds the next occurrence of X.

## Copy-Selection

Often, significant pieces of what you wish to type can be found in an active DEdit window. To aid in transferring the keystrokes that these objects represent into the type-in buffer, DEdit supports copy-selection. Whenever a selection is made in the

DEdit window with either shift key or the COPY key down, the selection made is not pushed on the selection stack, but is instead unread into the keyboard input (and hence shows up in the type-in buffer). A characteristically different highlighting is used to indicate when copy selection (as opposed to normal selection) is taking place.

Note: Copy-selection remains active even when DEdit is not. Thus you can unread particularly choice pieces of text from DEdit windows into an Exec window.

## Entering DEdit Commands

Invoke a DEdit command by selecting an item from the DEdit command menu. This can be done either directly, using the left mouse button in the usual way, or by selecting a subcommand. Subcommands are less frequently used commands than those on the main edit command menu and are grouped together in submenus under the main menu to which they are most closely related.

For example, the teletype editor defines six commands for adding and removing parentheses (defined in terms of transformations on the underlying list structure). Of these six commands, only two (inserting and removing parentheses as a pair) are commonly used, so DEdit provides the other four as subcommands of the common two.

The subcommands of a command are accessed by selecting the command from the commands menu with the middle button. This brings up a menu of the subcommand options from which a choice can be made. Subcommands are flagged in the list below with the name of the top level command of which they are options.

If you have a large DEdit window, or several DEdit windows active at once, the edit command window may be far away from the area of the screen in which you are operating. To solve this problem, the DEdit command menu is in an attached window. Whenever the TAB key is pressed, the command window moves over to the current cursor position and stay there as long as either the TAB key remains down or the cursor is in the command window. Thus, you can pull the command window over, slide the cursor into it and then release the TAB key (or not) while you make a command selection in the normal way. This eliminates a great deal of mouse movement.

Whenever a change is made, the pretty-printer reprints until the printing stabilizes. As the standard pretty print algorithm is used, and as it leaves no information behind on how it makes its choices, this is a somewhat heuristic process. The REPRINT command can be used to tidy the result up if it is not, in fact, "pretty."

## DEdit Functions

---

The functions used to start an editor are documented in the Edit Interface section of the *Lisp Release Notes*.

(RESETDEDIT)

[Function]

Completely reinitializes DEdit. Closes all DEdit windows, so that you must specify the window the next time DEdit is invoked. RESETDEDIT is also used to make DEdit recognize the new values of variables such as DEDITYPEINCOMS (see the DEdit Parameters section below), when you change them.

## DEdit Commands

All commands take their operands from the selection stack, and may push a result back on the stack. In general, the rule is to select target selections first and source selections second. Thus, a REPLACE command is done by selecting the thing to be replaced, selecting (or typing) the new material, and then selecting the REPLACE command in the command menu.

Using *TOP* to denote the topmost (most recent) element of the stack and *NXT* the second element, the DEdit commands are:

AFTER [DEdit Command]

Inserts a copy of *TOP* after *NXT*.

BEFORE [DEdit Command]

Inserts a copy of *TOP* before *NXT*.

DELETE [DEdit Command]

Deletes *TOP* from the structure being edited. (A copy of) *TOP* remains on the stack and appears, selected, in the edit buffer.

REPLACE [DEdit Command]

Replaces *NXT* with a copy of *TOP* obtained by substituting a copy of *NXT* wherever the value of the atom EDITEMBEDTOKEN (initially, the & character) appears in *TOP*. This provides a facility like the MBD edit command in Lisp; see EXTRACT, EMBED and IDIOMS below.

SWITCH [DEdit Command]

Exchanges *TOP* and *NXT* in the structure being edited.

() [DEdit Command]

( IN [DEdit Command]

Subcommands of (). Inserts ( before *TOP* (like the LI EDIT command; see the Commands That Edit Parentheses section below).

) IN [DEdit Command]

Subcommand of (). Inserts ) after *TOP* (like the RI EDIT command; see the Commands That Edit Parentheses section below).

() OUT [DEdit Command]

Removes parentheses from *TOP*.

( OUT [DEdit Command]

Subcommand of () OUT. Removes ( from before *TOP* (like the LO EDIT command; see the Commands That Edit Parentheses section below).

) OUT [DEdit Command]

Subcommand of () OUT. Removes ) from after *TOP* (like the RO EDIT command; see the Commands That Edit Parentheses section below).

|                                                                                                                                                                                                                                                                                                                                                        |                 |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| UNDO                                                                                                                                                                                                                                                                                                                                                   | [DEdit Command] |
| Undoes last command.                                                                                                                                                                                                                                                                                                                                   |                 |
| !UNDO                                                                                                                                                                                                                                                                                                                                                  | [DEdit Command] |
| Subcommand of UNDO. Undoes all changes since the start of this call on DEdit. This command can be undone.                                                                                                                                                                                                                                              |                 |
| ?UNDO                                                                                                                                                                                                                                                                                                                                                  | [DEdit Command] |
| &UNDO                                                                                                                                                                                                                                                                                                                                                  | [DEdit Command] |
| Subcommands of UNDO that allow selective undoing of other than the last command. Both of these commands bring up a menu of all the commands issued during this call on DEdit. When you select an item from this menu, the corresponding command (and if &UNDO, all commands since that point) will be undone.                                          |                 |
| FIND                                                                                                                                                                                                                                                                                                                                                   | [DEdit Command] |
| Selects, in place of <i>TOP</i> , the first place after <i>TOP</i> that matches <i>NXT</i> . Uses the edit subsystem's search routine, so supports the full wildcarding conventions of EDIT.                                                                                                                                                           |                 |
| SWAP                                                                                                                                                                                                                                                                                                                                                   | [DEdit Command] |
| Exchanges <i>TOP</i> and <i>NXT</i> on the stack, i.e. the stack is changed, the structure being edited isn't.                                                                                                                                                                                                                                         |                 |
| SWAP and its subcommands affect the stack and the selections, rather than the structure being edited.                                                                                                                                                                                                                                                  |                 |
| CENTER                                                                                                                                                                                                                                                                                                                                                 | [DEdit Command] |
| Subcommand of SWAP. Scrolls until <i>TOP</i> is visible in its window.                                                                                                                                                                                                                                                                                 |                 |
| CLEAR                                                                                                                                                                                                                                                                                                                                                  | [DEdit Command] |
| Subcommand of SWAP. Discards all selections (i.e., clears the stack).                                                                                                                                                                                                                                                                                  |                 |
| COPY                                                                                                                                                                                                                                                                                                                                                   | [DEdit Command] |
| Subcommand of SWAP. Puts a copy of <i>TOP</i> into the edit buffer and makes it the new <i>TOP</i> .                                                                                                                                                                                                                                                   |                 |
| POP                                                                                                                                                                                                                                                                                                                                                    | [DEdit Command] |
| Subcommand of SWAP. Pops <i>TOP</i> off the selection stack.                                                                                                                                                                                                                                                                                           |                 |
| REPRINT                                                                                                                                                                                                                                                                                                                                                | [DEdit Command] |
| Reprints <i>TOP</i> .                                                                                                                                                                                                                                                                                                                                  |                 |
| EDIT                                                                                                                                                                                                                                                                                                                                                   | [DEdit Command] |
| Runs DEdit on the definition of the atom <i>TOP</i> (or <i>CAR</i> of list <i>TOP</i> ). Uses TYPESOF to determine what definitions exist for <i>TOP</i> and, if there is more than one, asks you, via a menu, which one to use. If <i>TOP</i> is defined and is a non-list, calls INSPECT on that value. Edit also has a variety of subcommands which |                 |

allow choice of editor (DEdit, TTYEdit, etc.) and whether to invoke that editor on the definition of *TOP* or the form itself.

Note: DEdit caches each subordinate edit window in the window from which it was entered for as long as the higher window is active. Thus, multiple DEdit commands do not incur the cost of repeatedly allocating a new window.

EDITCOM [DEdit Command]

Allows you to run arbitrary EDIT commands on the structure being DEdited (there are far too many of these for them all to appear on the main menu). *TOP* should be an EDIT command, which will be applied to *NXT* as the current edit expression. On return to DEdit, the (possibly changed) current EDIT expression will be selected as the new *TOP*. Thus, selecting some expression, typing (R FOO BAZ), and selecting EDITCOM will cause FOO to be replaced with BAZ in the expression selected.

In addition, a variety of common EDIT commands are available as subcommands of EDITCOM. Currently, these include ?=, GETD, CL, DW, REPACK, CAP, LOWER, and RAISE.

BREAK [DEdit Command]

Does a BREAKIN AROUND the current expression *TOP*. (See BREAKIN function in *IRM*).

EVAL [DEdit Command]

Evaluates *TOP*, whose value is pushed onto the stack in place of *TOP*, and which will therefore appear, selected, in the edit buffer.

EXIT [DEdit Command]

Exits from DEdit (equivalent to Edit OK; see "Commands For Leaving The Editor," below).

OK [DEdit Command]

STOP [DEdit Command]

Subcommands of EXIT. OK exits without an error; STOP exits with an error. Equivalent to the EDIT commands with the same names.

## DEdit Parameters

There are several global variables that can be used to affect various aspects of DEdit's operation.

EDITEMBEDTOKEN [Variable]

Initially &. Used in both DEdit and the teletype editor to indicate the special atom used as the embed token.

DEDITLINGER [Variable]

Initially T. The default behavior of the topmost DEdit window is to remain active on the screen when exited. This is occasionally inconvenient for

programs that call DEdit directly, so it can be made to close automatically when exited by setting this variable to NIL.

DEDITTYPEINCOMS

[Variable]

Defines the control characters recognized as commands during DEdit type-in. The elements of this list are of the form

(*LETTER COMMANDNAME FN*)

*LETTER* is the alphabetic character corresponding to the control character desired (e.g., A for control-A),

*COMMANDNAME* is a symbol used both as a prompt and internal tag,

*FN* is a function applied to the expressions typed as arguments to the command.

See the current value of DEDITTYPEINCOMS for examples. DEDITTYPEINCOMS is only accessed when DEdit is initialized, so DEdit should be reinitialized with RESETDEDIT (see the Calling DEdit section above) if it is changed.

DT.EDITMACROS

[Variable]

Defines the behavior of the EDIT command when invoked on a form that is not a list or symbol, thus telling DEdit how to edit instances of certain datatypes. DT.EDITMACROS is an association list keyed by datatype name; entries are of the form

(*DATATYPE MAKESOURCEFN INSTALLEDITFN*)

When told to edit an object of type *DATATYPE*, DEdit calls *MAKESOURCEFN* with the object as its argument.

*MAKESOURCEFN* can either do the editing itself, in which case it returns NIL, or else it destructures the object into an editable list and returns that list.

In the latter case, DEdit is then invoked recursively on the list; when that edit is finished, DEdit calls *INSTALLEDITFN* with two arguments, the original object and the edited list. If *INSTALLEDITFN* causes an error, the recursive DEdit is invoked again, and the process repeats until the you either exit the lower editor with STOP, or exit with an expression that *INSTALLEDITFN* accepts.

For example, suppose the you have a datatype declared by (DATATYPE FOO (NAME AGE SEX)). To make sure that instances of FOO can be edited, an entry (FOO DESTRUCTUREFOO INSTALLFOO) is added to DT.EDITMACROS, where the functions are defined by the following:

```
(DESTRUCTUREFOO (OBJECT)
 (LIST (fetch NAME of OBJECT)
 (fetch AGE of OBJECT)
 (fetch SEX of OBJECT)))
(INSTALLFOO (OBJECT CONTENTS)
 (if (EQLENGTH CONTENTS 3)
 then (replace NAME of OBJECT with (CAR CONTENTS))
 (replace AGE of OBJECT with (CADR CONTENTS))
 (replace SEX of OBJECT with (CADDR CONTENTS))
 else (ERROR "Wrong number of fields for FOO" CONTENTS)))
```

## User Interface — Advanced Features

### Multiple DEdit Commands

It is occasionally useful to be able to give several commands at once — either because you think of them as a unit or because the intervening re-pretty-printing is distracting. The stack architecture of DEdit makes such multiple commands easy to construct. You just push whatever arguments are required for the complete suite of commands you have in mind. Multiple commands are specified by holding down the CONTROL key during command selection. As long as the control key is down, commands selected will not be executed, but merely saved on a list. Finally, when a command is selected without the control key down, the command sequence is terminated with that command being the last one in the sequence.

You would rarely construct long sequences of commands in this fashion, because the feedback of being able to inspect the intermediate results is usually worthwhile. Typically, just two or three step idioms are composed in this fashion.

### DEdit Idioms

As with any interactive system, there are certain common idioms on which experienced users depend heavily. In the case of DEdit, many of these idioms concern easy ways to achieve the effects of specific commands from the Edit system, with which many users are already familiar. The DEdit idioms described below are the result of the experience of the early users of the system and are by no means exhaustive. In addition to those that each user will develop to fit his own particular style, there are many more to be discovered and you are encouraged to share your discoveries.

Because of the novel argument specification technique (postfix; target first) many of the DEdit idioms are very simple, but opaque until you have absorbed the "target-source-command" way of looking at the world. Thus, you select where type-in is to go before touching the keyboard. After typing, the target will be selected second and the type-in selected on top, so that an AFTER, BEFORE or REPLACE will have the desired effect. If the order is switched, the command will try to change the type-in (which may or may not succeed), or will require tiresome swapping or reselection. Although this discipline seems strange at first, it comes easily with practice.

Segment selection and manipulation are handled in DEdit by first making them into a sublist, so they can be handled in the usual way. Thus, if you want to remove the three elements between A and E in the list (A B C D E), you select B, then D (either order), then make them into a sublist with the ( ) command. This will leave the sublist (B C D) selected, so a subsequent DELETE will remove it. This can be issued as a single " ( ) ; DELETE" command using multiple command selection as described above, in which case the intermediate state of (A (B C D) E) will not show on the screen.

Inserting a segment proceeds in a similar fashion. Once the location of the insertion is selected, the segment to be inserted is typed as a list (if it is a list of atoms, they can be typed without parentheses and the READ will make them into a list, as you would expect). Then, the command sequence "AFTER (or BEFORE or REPLACE) ; ( ) OUT" (given either as a multiple command or as two separate commands) will insert the type-in and splice it in by removing its parentheses.

Moving an expression to another place in the structure being edited is easily accomplished by a DELETE followed by an INSERT. Select the location where the moved expression is to go to; select the expression to be moved; then give the command sequence "DELETE; AFTER (or BEFORE or REPLACE)". The expression will first be



deleted into the edit buffer where it will remain selected. The subsequent insertion will insert it back into the structure at the selected location.

Embedding and extracting are done with the `REPLACE` command. Extraction is simply a special case of replacing something with a subpiece of itself:

- Select the thing to be replaced
- Select the subpart that is to replace it
- `REPLACE`

Embedding also uses `Replace`, in conjunction with the `embed` token (the value of `EDITEMBEDTOKEN`, initially the single character atom `&`). Thus, to embed some expression in a `PROG`:

- Select the expression.
- Type: (`PROG VARSLST &`)
- `REPLACE`

`SWITCH` can also be used to generate a whole variety of complex moves and embeds.

For example, switching an expression with type-in not only replaces that expression with the type-in, but provides a copy of the expression in the buffer, from where it can be edited or moved to somewhere else.

Finally, you can exploit the stack structure on selections to queue multiple arguments for a sequence of commands. Thus, to replace several expressions by one common replacement, select each of the expressions to be replaced (any number), then the replacing expression. Now hit the `REPLACE` command as many times as there are replacements to be done. Each `REPLACE` will pop one selection off the stack, leaving the most recently replaced expression selected. As the latter is now a copy of the original source, the next `REPLACE` will have the desired effect, and so on.

---

## Limitations

DEdit is not error-protected. If you select the up-arrow to close a break window which resulted from using the `EVAL` command, the DEdit window is also closed.

[This page intentionally left blank]

# EDITBITMAP

---

EditBitMap provides an interface (`EDIT.BITMAP`) for creating and editing bitmaps, which may exist as named files or as part of another type of a file (for example, a document written in TEdit).

EditBitMap puts up a menu of bitmap-manipulation commands, one of which is `HAND.EDIT`, which accesses `EDITBM`, the Interlisp-D bitmap editor.

EditBitMap also works on cursors (produces new cursor) and symbols (works on the value and resets the value with the result).

## Requirements

---

`READNUMBER`

`SCALEBITMAP`

## Installation

---

Load `EDITBITMAP.LCOM` and the required `.LCOM` modules from the library.

## User Interface

---

The user interface consists of a function (`EDIT.BITMAP`), a main operation menu, and a three-part window for low-level pixel editing.

There are two principal ways of entering the bitmap editor. If the bitmap is an object in a document being edited, you can enter the bitmap editor by pressing the left button over the bitmap. If the bitmap is an object you are manipulating as part of a program, you can call the function `EDIT.BITMAP` from the Executive, passing it the bitmap (typically the value of some variable).

In either case, EditBitMap presents its main menu, from which you select the operation you desire. If the operation is "Hand Edit", EditBitMap brings up a three-part window to show, create, or edit a bitmap. The individual EditBitMap operations can also be performed programmatically or from the Executive (see "Functions").

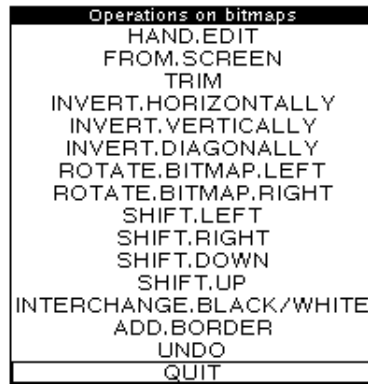
### EDIT.BITMAP

The function `EDIT.BITMAP` is the principal way to create, view or edit bitmaps stored as the values of variables (or other easily accessible Lisp values):

`(EDIT.BITMAP BITMAP)` [Function]

*BITMAP* may be a bitmap, a cursor, or a symbol. If *BITMAP* is a bit map, then `EDIT.BITMAP` returns a new bitmap as the result of the edit. If *BITMAP* is a cursor, then `EDIT.BITMAP` operates on its bitmap and returns a new cursor. If *BITMAP* is a symbol, then `EDIT.BITMAP` operates on the symbol's value (a bitmap or cursor), and resets the symbol's value to the result of the edit.

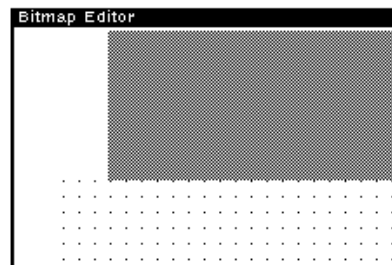
EditBitMap brings up a main menu containing the following items:



EditBitMap performs each command you select, until you select QUIT, at which point it returns the final result of all the edits. You can select UNDO to undo the most recent operation (selecting it several times undoes several operations). If you select HAND.EDIT, you enter the pixel editor EDITBM (see the *IRM*). If you select FROM.SCREEN, EditBitMap prompts you for a screen region from which to initialize a new bit map. The remaining menu items are described under the corresponding "Function" below.

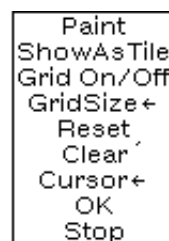
## Window

The EditBitMap window consists of three parts. The main, lower part is the region where the bitmap is displayed on a grid by means of fat pixels. The smaller top part is a gray background against which the middle-button submenu is displayed. The small portion in the upper left corner displays a miniature picture of the entire bitmap.



## Submenu

The EditBitMap submenu is displayed when you press the middle button in the upper gray region of the window. It contains the following items:

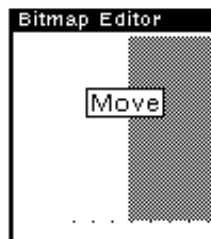


These menu items are described in the system prompt window when an item is selected.

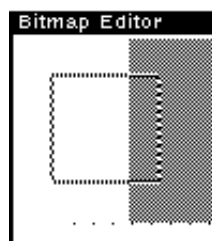
## Mouse Buttons

Pressing the right button anywhere in the EditBitMap window causes the usual window menu to be displayed.

Pressing the left or middle button in the upper left portion of the window presents a MOVE icon.



Pressing on the MOVE icon presents a rectangle which can be moved about in the subwindow. This rectangle indicates the portion of the entire bitmap which will be displayed in the large, bottom subwindow, as soon as the button is released.



Pressing the middle button in the upper, gray subwindow causes the EditBitMap submenu to be displayed (see above).

Pressing the left button in the upper, gray subwindow highlights a rectangle in the upper left subwindow, showing which portion of the entire bitmap is displayed in the large lower subwindow.

Pressing the left button (or dragging the mouse with the left button held down) in the large, lower portion of the window changes the pixels; you are editing at the pixel level.

## Editing an Existing Bitmap

If you have a variable *OLDNAME* whose value is a bitmap, you can make a modified version assigned to *NEWNAME* by typing to the Executive window:

```
(SETQ NEWNAME (EDIT.BITMAP OLDNAME))
```

Or if you want to modify *OLDNAME* in place, pass the quoted name itself:

```
(EDIT.BITMAP 'OLDNAME)
```

In either case, the main menu pops up on the screen. Select the operations you wish to perform, including HAND.EDIT to edit at the pixel level.

Edit the bitmap as needed.

Move the cursor into the gray upper region. Press the middle button to get the submenu. Select OK.

In the main menu, select QUIT. The Executive window displays the new bitmap address.

## Viewing an Existing Bitmap

You can use the hand editor simply to view a bitmap. In this case you don't need to include a SETQ to save the value. For example, type

```
(EDIT.BITMAP BITMAPNAME)
```

Note: Any edits you might be tempted to make while viewing the bitmap in this way will not be saved.

## Creating a New Bitmap

You can use any of the standard graphics interfaces documented in the *IRM* to create a new bitmap. EditBitMap does provide one convenient way to create a bitmap from a region of the screen. In the Executive window, type

```
(SETQ NEWBITMAPNAME (EDIT.BITMAP))
```

Again the main menu pops up on the screen. Select FROM.SCREEN. The cursor changes into the standard region prompt, allowing you to select a region of the screen. Hold down the left mouse button to mark one corner of the region, and drag the mouse to the opposite corner. When you let go of the mouse button, the contents of the screen region you selected are used to initialize a new bitmap. You can then select any other operations you wish to transform this initial image, including HAND.EDIT if appropriate. When finished with all the operations, select QUIT from the main menu. The variable *NEWBITMAPNAME* is now set to the bitmap you have created.

If you want to create a bitmap completely from scratch using the pixel editor, it is simplest to call it directly:

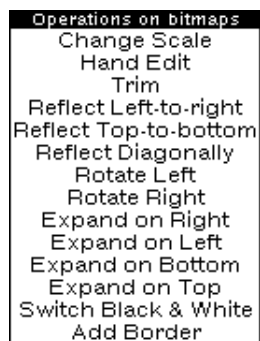
```
(SETQ NEWBITMAPNAME (EDITBM))
```

You will be prompted to supply the width and height of the new bitmap in pixels. When you are finished editing, move the cursor into the gray upper region, press the middle button to get the submenu, and select OK.

If you want to perform further transformations on the new bitmap, edit *NEWBITMAPNAME* as described above for existing bitmaps.

## Editing a Bitmap in a Document

To edit a bitmap that exists inside another file, such as a document being edited in TEdit, press the left or middle button anywhere inside the image of the bitmap. A modified version of EditBitMap's main menu pops up containing the following items:



```
Operations on bitmaps
Change Scale
Hand Edit
Trim
Reflect Left-to-right
Reflect Top-to-bottom
Reflect Diagonally
Rotate Left
Rotate Right
Expand on Right
Expand on Left
Expand on Bottom
Expand on Top
Switch Black & White
Add Border
```

These menu items correspond exactly to the similarly-named items in EditBitMap's regular menu, except that only one operation is performed at a time (to repeat an

operation, just select it again with the mouse; to Undo, use TEdit's Undo command). The menu also contains an additional item, `CHANGE SCALE`, which allows you to change the scale at which the bitmap's image appears in the document (on the screen and on the printer). Scaling a bitmap changes only the size of its image; it has no effect on its contents (even though on the screen you may not always see it that way).

## Functions

(`EDIT.BITMAP BITMAP`) [Function]

(See above)

(`ADD.BORDER.TO.BITMAP BITMAP NBITS TEXTURE`) [Function]

Returns a new bitmap that is *BITMAP* extended by *NBITS* in all four directions, the border being filled in with *TEXTURE*.

(`BIT.IN.COLUMN BITMAP COLUMN`) [Function]

Returns T if any bit in column numbered *COLUMN* (left = 0) is not 0, NIL otherwise.

(`BIT.IN.ROW BITMAP ROW`) [Function]

Returns T if any bit in row numbered *ROW* (bottom = 0) is not zero, NIL otherwise.

(`INVERT.BITMAP.B/W BITMAP`) [Function]

Swaps black and white bits in a bitmap. On the EditBitMap menu, this is called by selecting `INTERCHANGE.BLACK/WHITE`. On the TEdit bitmap editor menu, it is called by selecting "Switch Black & White."

(`INVERT.BITMAP.DIAGONALLY BITMAP`) [Function]

Returns a new bitmap, which is *BITMAP* flipped about the X=Y diagonal. (The resulting bitmap's width will be *BITMAP*'s height.)

(`INVERT.BITMAP.HORIZONTALALLY BITMAP`) [Function]

Returns a new bitmap, which is *BITMAP* flipped about its vertical center line.

(`INVERT.BITMAP.VERTICALLY BITMAP`) [Function]

Returns a new bitmap, which is *BITMAP* flipped about its horizontal center line.

(`ROTATE.BITMAP.LEFT BITMAP`) [Function]

Returns a new bitmap, which is *BITMAP* rotated 90 degrees counterclockwise. (The resulting bitmap's width will be *BITMAP*'s height.)

(`ROTATE.BITMAP.RIGHT BITMAP`) [Function]

Returns a new bitmap, which is *BITMAP* rotated 90 degrees clockwise. (The resulting bitmap's width will be *BITMAP*'s height.)

(`SHIFT.BITMAP.DOWN BITMAP NBITS`) [Function]

Returns a new bitmap, which is *BITMAP* extended by *NBITS* in the upward direction, the new space being filled in with white.

(SHIFT.BITMAP.UP *BITMAP* *NBITS*) [Function]

Returns a new bitmap, which is *BITMAP* extended by *NBITS* in the downwards direction, the new space being filled in with white.

(SHIFT.BITMAP.LEFT *BITMAP* *NBITS*) [Function]

Returns a new bitmap, which is *BITMAP* extended by *NBITS* to the right, the new space being filled in with white.

(SHIFT.BITMAP.RIGHT *BITMAP* *NBITS*) [Function]

Returns a new bitmap, which is *BITMAP* extended by *NBITS* to the left, the new space being filled in with white.

(TRIM.BITMAP *BITMAP*) [Function]

Returns a new bitmap, which is *BITMAP* trimmed at all four edges of all completely white (0) columns and rows.

(FROM.SCREEN.BITMAP NIL) [Function]

Prompts for a region on the screen and returns a copy of the bitmap.

(INTERACT&SHIFT.BITMAP.LEFT *BITMAP*) [Function]

Prompts for number of bits to shift the *BITMAP* left and returns the new bitmap.

(INTERACT&SHIFT.BITMAP.RIGHT *BITMAP*) [Function]

Prompts for number of bits to shift the *BITMAP* right and returns the new bitmap.

(INTERACT&SHIFT.BITMAP.DOWN *BITMAP*) [Function]

Prompts for number of bits to shift the *BITMAP* down and returns the new bitmap.

(INTERACT&SHIFT.BITMAP.UP *BITMAP*) [Function]

Prompts for number of bits to shift the *BITMAP* up and returns the new bitmap.

(INTERACT&ADD.BORDER.TO.BITMAP *BITMAP*) [Function]

Prompts for number of bits in the border and calls EDITSHADE to interactively fill in the texture. Returns a new bitmap, which is a bitmap extended in all four directions by the border being filled in with the texture.

Note: If the interactive functions are called from the menus, the prompt for the number of bits is in the form of a ReadNumber window:



|                                                |                                                                                                                                                                                                                                     |    |   |     |  |   |   |   |   |   |   |   |   |   |    |   |    |
|------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|---|-----|--|---|---|---|---|---|---|---|---|---|----|---|----|
| Number of bits to<br>shift the bitmap<br>left: |                                                                                                                                                                                                                                     |    |   |     |  |   |   |   |   |   |   |   |   |   |    |   |    |
| <input type="text" value="0"/>                 | <table border="1"><tr><td>-</td><td colspan="2">clr</td></tr><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr><tr><td>bs</td><td>0</td><td>ok</td></tr></table> |    | - | clr |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | bs | 0 | ok |
| -                                              | clr                                                                                                                                                                                                                                 |    |   |     |  |   |   |   |   |   |   |   |   |   |    |   |    |
| 1                                              | 2                                                                                                                                                                                                                                   | 3  |   |     |  |   |   |   |   |   |   |   |   |   |    |   |    |
| 4                                              | 5                                                                                                                                                                                                                                   | 6  |   |     |  |   |   |   |   |   |   |   |   |   |    |   |    |
| 7                                              | 8                                                                                                                                                                                                                                   | 9  |   |     |  |   |   |   |   |   |   |   |   |   |    |   |    |
| bs                                             | 0                                                                                                                                                                                                                                   | ok |   |     |  |   |   |   |   |   |   |   |   |   |    |   |    |

---

## Limitations

Selecting OK in the submenu does NOT save the edits made in the bitmap. Edits are saved only if you specify a new bitmap name before you begin editing an old one, or if you pass a quoted name to `EDIT.BITMAP`.

[This page intentionally left blank]

# ETHERRECORDS

---

EtherRecords contains a collection of record definitions needed for low-level Ethernet programming in Lisp.

## Installation

---

Load ETHERRECORDS from the library.

## General Purpose Records

---

ETHERPACKET [Data type]

A data type describing a level-zero Ethernet packet. Use a BLOCKRECORD overlaying this record to define various level-one packets (see PUP and XIP below for examples).

SYSQUEUE [Data type]

A data type implementing a low-level queue for Ethernet use.

QABLEITEM [Record]

A record that overlays any data type whose first field is a pointer used for linking items on a SYSQUEUE.

## NS Records

---

XIP [Record]

A record overlaying ETHERPACKET describing the layout of a standard Xerox Internet Packet.

ERRORXIP [Record]

A record overlaying ETHERPACKET describing the layout of a standard XNS error packet. The value of the ERRORXIPCODE field of this record is the most interesting one for programmatic handling of XIP errors. The variable XIPERRORCODES contains constants defining most of the standard error codes.

\XIPOVLEN [Constant]

A constant representing the number of bytes in a XIP exclusive of the data portion; i.e., the LENGTH field of a XIP is the byte length of its data portion plus \XIPOVLEN.

\MAX.XIPDATALENGTH [Constant]

A constant, the maximum number of bytes permitted in a standard XIP (546).

NSHOSTNUMBER

[Record]

A record describing a 48-bit XNS host number.

|                                                                                            |             |
|--------------------------------------------------------------------------------------------|-------------|
| NSADDRESS                                                                                  | [Data type] |
| A data type describing a complete XNS address: 32-bit network, 48-bit host, 16-bit socket. |             |
| NSNAME                                                                                     | [Data type] |
| A data type describing a standard three-part Clearinghouse name.                           |             |

## PUP Records

|                                                                                                                                                                                                                                                                                                                                      |            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| PUP                                                                                                                                                                                                                                                                                                                                  | [Record]   |
| A record overlaying ETHERPACKET describing the layout of a standard PUP (PARC Universal Packet).                                                                                                                                                                                                                                     |            |
| ERRORPUP                                                                                                                                                                                                                                                                                                                             | [Record]   |
| A record overlaying ETHERPACKET describing the layout of a standard PUP error packet. The value of the ERRORPUPCODE field of this record is the most interesting one for programmatic handling of PUP errors. The variable PUPERRORCODES contains constants defining most of the standard error codes.                               |            |
| PUPADDRESS                                                                                                                                                                                                                                                                                                                           | [Record]   |
| A record describing how to take a 16-bit PUP address apart into 8-bit network and host numbers.                                                                                                                                                                                                                                      |            |
| \PUPOVLEN                                                                                                                                                                                                                                                                                                                            | [Constant] |
| A constant representing the number of bytes in a PUP exclusive of the data portion; i.e., the LENGTH field of a PUP is the byte length of its data portion plus \PUPOVLEN.                                                                                                                                                           |            |
| \MAX.PUPDATALENGTH                                                                                                                                                                                                                                                                                                                   | [Constant] |
| A constant, the maximum number of bytes permitted in a standard PUP (532).                                                                                                                                                                                                                                                           |            |
| \LOCALPUPADDRESS                                                                                                                                                                                                                                                                                                                     | [Macro]    |
| \LOCALPUPHOSTNUMBER                                                                                                                                                                                                                                                                                                                  | [Macro]    |
| \LOCALPUPNETNUMBER                                                                                                                                                                                                                                                                                                                   | [Macro]    |
| These three macros return components of the PUP address of the machine on which the code is running. \LOCALPUPHOSTNUMBER and \LOCALPUPNETNUMBER return the machine's 8-bit host and 8-bit net numbers, respectively; \LOCALPUPADDRESS returns both as a 16-bit number, suitable as a value of the PUPSOURCE field of the PUP record. |            |

[This page intentionally left blank]



# FILEBROWSER

---

FileBrowser provides a convenient user interface for manipulating files stored on a workstation or file server. It enables you to see, edit, delete, print, load, copy, move, rename, compile, sort, and get several types of information about groups of files. You can also customize FileBrowser by adding your own commands.

## Requirements

---

TABLEBROWSER

In addition, the `HARDCOPY` commands require printer drivers and fonts, and the `EDIT` command requires one or more editors (`TEdit`, `SEdit`, `DEdit`).

## Installation

---

Load `FILEBROWSER.LCOM` and the required `.LCOM` modules from the library.

## User Interface

---

### Starting FileBrowser

Once you have loaded FileBrowser, there are two ways to open a browser on a set of files:

1. Select the FileBrowser command from the background menu, in which case you are prompted for a file name pattern, or
2. Type the command `FB FILEPATTERN` in your Executive window.

In either case, FileBrowser will prompt you to create a window by presenting you with a dashed rectangle with the mouse cursor and a small geometric design at the lower right corner.

1. Move your mouse until the upper left corner of the rectangle is where you want it on the screen.
2. Hold down the left mouse button and move your mouse down and to the right, thus expanding the window diagonally, until the window is the right size.
3. Release the mouse button. This creates a window group on your screen in the outlined area.

Next, if you did not specify a pattern by using the `FB` command, FileBrowser prompts you for a file pattern. Type a pattern, as described in the *Specifying What Files to Browse* section below.

FileBrowser enumerates the set of files matching the pattern you requested to see. While the enumeration is in progress, the `RECOMPUTE` command is grayed out. When the enumeration is finished, you may select files and issue commands. You can scroll the window at any time, even while the browser is busy.



If FileBrowser cannot find any files matching the pattern you specified, or you decide you specified the wrong pattern and want to try again, you can specify a new file name pattern from within the browser using the `NEW PATTERN` command; see `RECOMPUTE` in the FileBrowser Commands section below.

You can have as many active FileBrowsers open at once as you like.

## Specifying What Files to Browse

A full file name in Lisp consists of a device or host (such as your local disk, a file server, or a floppy disk), a principal directory and zero or more subdirectories, a file name (possibly including an extension), and a version number. These fields are put together in the form

```
{HOST}<DIRECTORY>SUBDIRECTORY>FILENAME.EXTENSION;VERSION
```

A file name pattern, as specified to FileBrowser, consists of a file name with one or more pieces omitted or filled with wild cards (\*). All the files matching the pattern are listed by FileBrowser. Thus, you can browse all the files in a particular directory, all the files in a subdirectory of that directory, all the files in a directory with a particular extension, and so forth. The wild card \* can be used to stand for zero or more consecutive characters in the file name. You can use as many wild cards in a pattern as you wish.

If you leave out some of the fields in a file name pattern, the missing fields are defaulted by the system. Omitted fields in the front of the pattern, i.e., the host, device, or directory fields, are filled in by consulting your connected directory. Other omitted fields are filled in with wild cards unless they are explicitly omitted; i.e., the field is empty, but the preceding punctuation is still present. In more detail, some of the cases are as follows:

If you leave out the name of the host/device, specifying `<DIRECTORY>FILENAME`, FileBrowser will use the name of the host/device for the directory to which you are currently connected.

If you leave out both the device and directory names, specifying `FILENAME`, FileBrowser will use the device and directory to which you are currently connected.

If you do not specify a file name, FileBrowser lists all the files in the specified directory (or the connected directory if you also omitted the host and directory).

If you leave out the extension of a file name, FileBrowser lists all the files with the specified file name and any extension. If you omit the extension but include the period that usually precedes the extension, FileBrowser lists only the files with the specified name and *no* extension.

If you omit the version number of the file name, FileBrowser lists all versions of the matching files. If you omit the version number but include the semicolon that usually precedes the version, FileBrowser lists only the highest version of the matching files.

Thus, the minimal pattern you can type is \* (asterisk—enumerate all files in the connected directory) or ; (semicolon—enumerate just the highest version of all files). If you press the RETURN key without giving a pattern, FileBrowser aborts the prompt for a pattern, leaving you with an empty browser in which the only things you can do are change some FileBrowser parameters (see the subcommands of `RECOMPUTE` in the FileBrowser Commands section below) and then use the `RECOMPUTE` command to be prompted for a pattern again.

## Example

The pattern `*` specifies all files in the connected directory. It is equivalent to `*.*` or `*.*;*`.

The pattern `<FOO>BAR` specifies all files in directory `FOO` with name `BAR` and any extension. It is equivalent to `<FOO>BAR.*;*`.

The pattern `<FOO>BAR.` specifies all files in directory `FOO` with name `BAR` and *no* extension. It is equivalent to `<FOO>BAR.;*`.

The pattern `*.TEdit` specifies all files in the connected directory with the extension `TEdit`. It is equivalent to `*.TEdit;*`.

The pattern `*.TEdit;` specifies only the newest version of all files in the connected directory with the extension `.TEdit`.

The pattern `<FOO>A*E` specifies all files in directory `FOO` whose names begin with `A` and end with `E` and have any extension.

The pattern `{TOAST}<FOO>*MY*` specifies all files in directory `{TOAST}<FOO>` whose names contain the substring `MY` and any extension.

## Using the FileBrowser Window

The FileBrowser window has six major subwindows, which from top to bottom are as follows:

### PROMPT window

This topmost subwindow is where FileBrowser prints messages about what it is doing and receives input from you. Its contents are cleared before every command.

### TALLY window

This subwindow immediately below the prompt window keeps a running tally of the total number of files listed in the window and the number of files that you have marked for deletion. In addition, if one of the attributes you are displaying is a size attribute (Pages or Length, as in the INFO menu, described below), this window maintains a tally of the total number of pages in the files listed and the files marked for deletion.

This window also has a title bar across the top identifying the pattern you specified and the time at which the directory enumeration was performed.

The window is blank while the files are being enumerated.

| File group description: {erinyes}<Lisp>Lyric>Library>*.lcom           |       |                        |           |   |
|-----------------------------------------------------------------------|-------|------------------------|-----------|---|
| Enumerating {erinyes}<Lisp>Lyric>Library>*.lcom; * ...done            |       |                        |           |   |
| {ERINYES}<Lisp>Lyric>Library>*.lcom;* at 11:13 Wed 3-Feb- FB Commands |       |                        |           |   |
| Total: 99 / 4803 pages                                                |       | Deleted: 0 / 0 pages   |           |   |
| Name                                                                  | Pages | Created                |           |   |
| 4045XLPSTREAM.LCOM;1                                                  | 101   | 5-Mar-87 17:52:40 PST  | Delete    | » |
| BROWSER.LCOM;1                                                        | 19    | 24-Apr-87 10:59:38 PDT | Undelete  | » |
| CENTRONICS.LCOM;1                                                     | 8     | 29-Nov-86 17:22:04 PST | Copy      | » |
| CHARCODETABLES.LCOM;1                                                 | 8     | 29-Nov-86 17:22:28 PST | Rename    | » |
| CHAT.LCOM;1                                                           | 59    | 26-Jan-87 22:28:05 PST | Hardcopy  | » |
| CHATTERMINAL.LCOM;1                                                   | 20    | 27-Nov-86 13:27:14 PST | See       | » |
| CMLFLOATARRAY.LCOM;1                                                  | 23    | 9-Apr-87 16:32:31 PDT  | Edit      | » |
| COPYFILES.LCOM;1                                                      | 16    | 27-Nov-86 15:21:53 PST | Load      | » |
| DATABASEFNS.LCOM;1                                                    | 14    | 3-Dec-86 11:50:59 PST  | Compile   | » |
| DEDIT.LCOM;1                                                          | 96    | 29-Nov-86 17:31:58 PST | Expunge   | » |
| DEDITPP.LCOM;1                                                        | 30    | 16-Dec-86 17:56:57 PST | Recompute | » |
| DES.LCOM;1                                                            | 29    | 22-Dec-86 11:41:51 PST | Sort      | » |
| DLRS232C.LCOM;1                                                       | 109   | 3-Jun-87 10:18:19 PDT  |           |   |

Figure 1. Tally Window

### BROWSER window

This is the principal subwindow, in which the files matching the specified pattern are listed. Each file's name appears at the left, and various attributes of the file are displayed in columns to the right. A title bar across the top of the browser window identifies the contents of each column (e.g., Name, Pages, Created). The files are listed in alphabetical order, with multiple versions of the same file listed in decreasing version order; i.e., the newest version appears first. The width of the column listing the file names is initially chosen to be appropriate for average-sized file names. If the files you asked to browse have particularly long names, then when FileBrowser has finished listing all the files it may choose to redraw the browser window with the attribute columns moved farther to the right to accommodate the longer file names.

### COMMAND menu

This menu appears vertically along the right side of FileBrowser window (under a title bar "FB Commands") and lists the commands that you may select to perform operations on the files in the browser, or to change the appearance of the browser. Most of the commands operate on the set of currently selected files (see the Selecting Files section below). Some commands have subcommands, as indicated by the small triangle alongside them, which can be selected by holding down the left mouse button and sliding the mouse to the right over the triangle.

### INFO menu

An additional subwindow, the Info menu, is not normally displayed. It is used to change the set of file information (attributes) displayed in the browser (see the Getting Information About Files section below).

### SCROLL bar

If there are more files in the listing than fit at one time in the browser window, you can scroll the browser window to view more files. Slide the mouse cursor out the left side of the browser window to get the scroll bar and press the left mouse button to scroll the region up and the right mouse button to scroll it down. Pressing a mouse button when the cursor is near the bottom of the scroll bar scrolls the region by larger increments than when the cursor is at the top.

You can also press the middle mouse button in the scroll bar to move the listing to the place that corresponds to that position in the scroll bar. For example,

pressing the middle mouse button when the cursor is at the bottom of the scroll bar displays the end of the listing. This quick-scrolling technique is called thumbing. The gray box in the scroll region indicates where the currently displayed contents are, relative to the entire contents of the browser.

Similarly, if there is more attribute information than fits in the browser window, you can scroll the browser window horizontally to view the rest of the attribute information. To do this, slide the mouse out the bottom of the browser window to get the horizontal scroll bar. The left button scrolls to the left, the right button to the right.

## Selecting Files

Most FileBrowser operations are performed by selecting a single file or set of files, then giving a command that specifies what you want to do with the selected files. The current selection is indicated by a small triangle in the left margin of the browser next to each selected file.

|                         |    |           |          |     |           |    |
|-------------------------|----|-----------|----------|-----|-----------|----|
| ▶ CENTRONICS.LCOM;1     | 8  | 29-Nov-86 | 17:22:04 | PST | See       | ▶▶ |
| ▶ CHARCODETABLES.LCOM;1 | 8  | 29-Nov-86 | 17:22:28 | PST | Edit      | ▶▶ |
| ▶ CHAT.LCOM;1           | 59 | 26-Jan-87 | 22:28:05 | PST | Load      | ▶▶ |
| ▶ CHATTERMINAL.LCOM;1   | 20 | 27-Nov-86 | 13:27:14 | PST | Compile   | ▶▶ |
| CMLFLOATARRAY.LCOM;1    | 23 | 9-Apr-87  | 16:32:31 | PDT | Expunge   | ▶▶ |
| COPYFILES.LCOM;1        | 16 | 27-Nov-86 | 15:21:53 | PST | Recompute | ▶▶ |
| DATABASEFNS.LCOM;1      | 14 | 3-Dec-86  | 11:50:59 | PST | Sort      | ▶▶ |
| ▶ DEDIT.LCOM;1          | 96 | 29-Nov-86 | 17:31:58 | PST |           |    |
| ▶ DEDITPP.LCOM;1        | 30 | 16-Dec-86 | 17:56:57 | PST |           |    |
| DES.LCOM;1              | 29 | 22-Dec-86 | 11:41:51 | PST |           |    |

Figure 2. FileBrowser Window with Files Selected

To select one file, point to any part of the line (which lists the file name and its attributes) and press the left mouse button. If other files are already selected, this unselects them; thus, a file selected with the left mouse button is always the only selection.

To add a single file to the current selection, press the middle mouse button at any place in the line. The file is selected without unselecting any other file.

To remove a single file from the current selection, hold down the control key and press the middle mouse button at any place in the line. The file is unselected without affecting any other file.

To extend the selection to include a group of contiguous files, that is, to select all the files between a file and the nearest already selected file, press the right mouse button on any part of the line. You can only extend the selection from the first selected file upward, or the last selected file downward. In addition, files marked for deletion are not normally selected when you extend.

If you want to include all files, both deleted and undeleted, hold down the control key while extending the selection.

Some lines in a FileBrowser display are directory-only lines. These lines are slightly indented and name the directory and subdirectory to which the files listed below that line belong. You cannot select in these lines, though you can copy-select them (see the Copy-Selecting Files section below).

## Commands that Require Input

Some FileBrowser commands require input from you. For example, the COPY command requires that you supply a destination file name. When a command requires

input, FileBrowser prints a prompt message in its prompt window. This is usually followed by a default answer. If you want the default answer, you can just press the carriage return to finish the input. If you want to specify a different answer, simply start typing it; the default answer is erased and your answer replaces it.

Alternatively, you can modify the default answer by backspacing over individual letters, or typing control-W to back up over complete words. Typing control-Q erases the entire answer. You can also use the mouse to edit your answer, using the same rules as followed by the Executive (see the documentation of TTYIN). Briefly, the left mouse button positions the caret at a character boundary; the middle mouse button positions the caret at the nearest word boundary; and the right mouse button deletes the characters between the caret and the mouse.

When you have finished, position the caret at the end of your answer, if it isn't there already, and press the carriage return. You can also type control-X to finish your answer even if the caret isn't at the end.

If you change your mind and want to abort the command, supply an empty input; i.e., if there is an answer in progress, backspace over it or type control-Q to erase it, then press the carriage return. FileBrowser prints "aborted" and aborts the command. In most situations, the control-E interrupt can also be used to abort your answer.

While you are typing an answer, you can copy-select file names out of the browser (or any other browser), as described below in the Copy-Selecting Files section. This can be useful, for example, if you wish to rename a file to a similar name in the same directory, or move a file into a subdirectory listed in the browser.

## Aborting Commands

During commands of indefinite duration, such as RECOMPUTE or COPY, FileBrowser adds another command to the browser, "Abort".

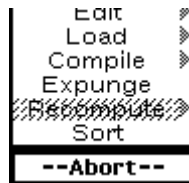


Figure 3. Command Menu with "Abort" Added at Bottom

Clicking on the Abort command will immediately abort the current operation. Aborting some commands can take a little while, as FileBrowser may need to do some cleaning up, so the Abort command is greyed out during this time to show you that it is doing something.

## Quitting the FileBrowser

To quit a FileBrowser, simply close the browser window. If any files have been deleted but not expunged, a small menu will pop up listing two options: "Expunge Deleted Files" and "Don't Expunge." If you choose "Expunge Deleted Files", the files will be expunged before the window closes. If you choose the "Don't Expunge" command, your deletions are ignored. If you click outside the menu, no action is taken, and the Close command is aborted.

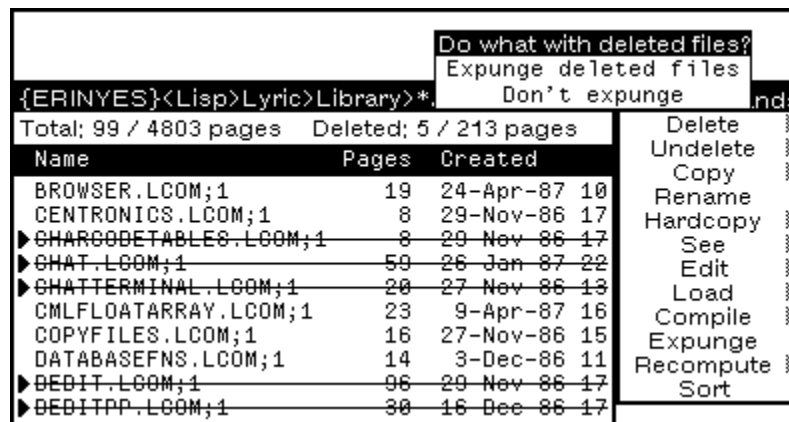


Figure 4. FileBrowser Window with Files Selected for Deletion

If you have finished with FileBrowser only temporarily and want to put it aside to work on later, you can shrink the browser (by selecting the SHRINK command from the right-button background menu). The browser shrinks to an icon which displays the file pattern inside the browser. If any files are marked for deletion, you will be prompted with the same menu of EXPUNGE options as when you close a browser.



Figure 5. Shrink Icon with File Pattern

## Copy-Selecting Files

You can copy-select file names from a FileBrowser into other windows, such as Executive and TEdit windows, by holding down the Copy (or Shift) key while selecting a name in the window. The full name of the file is inserted as if you had typed it where the input caret is flashing. You can also copy-select in a directory-only line, in which case the full directory name is inserted in your type-in.

**Note:** Most file names contain characters, in particular colon and semi-colon, that have special meaning to the Lisp reader. Thus, if your type-in point is in an Executive window, you probably want to type a double-quote character before and after the file name, so that the file name is presented to Lisp as a string.

## Getting Hardcopy Directory Listings

You can get a hardcopy listing of the directory displayed in a FileBrowser by using the regular window Hardcopy command. Press the right button in FileBrowser's prompt window or tally window and select HARDCOPY from the menu. FileBrowser will produce a hardcopy listing of the files and the attributes displayed in the browser.

If the browser displays a large number of attributes, or your default printer font is too large, the listing may not accommodate all the attributes on one line, making the listing less readable. You may want to make the listing with fewer attributes, or use a smaller font for the listing (see description of FB.HARDCOPY.FONT in the Customizing FileBrowser and Using the Programmer Interface section).

## FileBrowser Commands

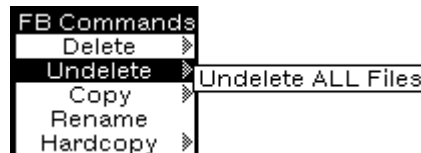
### DELETE, UNDELETE

Removing a file from the file system using FileBrowser is a two-step process. First, mark the file or files for deletion. Then issue the Expunge command. Any time between the deletion and the expunge you can change your mind and undelete any of the files.

To mark a file or files for deletion, select them, then choose the DELETE command. FileBrowser draws a line through the deleted files. It also adjusts the numbers in the tally window to show how many files are marked deleted and how many pages they contain. It is thus easy to see how much file space you will regain when you issue the EXPUNGE command.

|                        |    |           |    |           |
|------------------------|----|-----------|----|-----------|
| CENTRONICS.LCOM;1      | 8  | 29-Nov-86 | 17 | Hardcopy  |
| ► CHARGOBTABLES.LCOM;1 | 8  | 29-Nov-86 | 17 | See       |
| ► CHAT.LCOM;1          | 50 | 26-Jan-87 | 22 | Edit      |
| ► CHATTERMINAL.LCOM;1  | 20 | 27-Nov-86 | 13 | Load      |
| CMLFLOATARRAY.LCOM;1   | 23 | 9-Apr-87  | 16 | Compile   |
| COPYFILES.LCOM;1       | 16 | 27-Nov-86 | 15 | Expunge   |
| DATABASEFNS.LCOM;1     | 14 | 3-Dec-86  | 11 | Recompute |
| ► DEBIT.LCOM;1         | 96 | 29-Nov-86 | 17 | Sort      |
| ► DEBITPP.LCOM;1       | 30 | 16-Dec-86 | 17 |           |
| DES.LCOM;1             | 29 | 22-Dec-86 | 11 |           |

To undelete a file or files (i.e., to remove the deletion mark), select them, then choose the UNDELETE command. The lines through the files are removed, and the tally of deleted files is updated. The UNDELETE command has a single subcommand, UNDELETE ALL FILES, which undeletes all the files in the browser, independently of whether they are selected. This is useful if you completely change your mind about deleting any files.



The DELETE command has a useful subcommand, DELETE OLD VERSIONS. When you have been editing a file in the text editor and performing repeated PUT commands, or you the programmer have done many MAKEFILES of the same file, multiple versions of the file accumulate, each more recent version denoted by a higher version number. The DELETE OLD VERSIONS command is used to delete excess versions of the files displayed in the browser.



To use this command, press the mouse down on the DELETE command and slide the cursor out to the right, choosing the DELETE OLD VERSIONS command. Unlike the DELETE command (or DELETE SELECTED FILES, the equivalent subcommand), the DELETE OLD VERSIONS command operates on all the files in the browser. FileBrowser prompts you for the number of versions of each file that you wish to retain. It offers the default of one version. You can accept the default or you can type a different number of your choosing, followed by a carriage return. FileBrowser then marks for deletion all but the most recent N versions of all the files in the browser, where N is the number you specified. Before issuing the EXPUNGE command, you can, if you wish, scroll

through the browser, undeleting any particular files for which you wish to retain more versions than you specified.

The `DELETE OLD VERSIONS` command is sometimes useful even when you are not planning to actually expunge the files. This is because of the way extending the selection avoids deleted files (see the section "Selecting Files," above).

For example, if you wanted to copy only the most recent version of all the files in the browser to another location, you could do the following:

1. Use the `DELETE OLD VERSIONS` command, retaining just one version. This marks deleted all files but the newest version of each.
2. Go to the start of the browser and select the first file, then scroll to the end of the browser and press the right mouse button to extend the selection to the end of the browser. You have selected exactly the newest version of each file.
3. Use the `COPY` command to copy those files.
4. Finally, use the `UNDELETE ALL FILES` command to undelete all the old versions.

## COPY

The `COPY` command is used to copy an entire file or set of files to another file system location; for example, from your disk to a file server. Select the file(s) you wish to copy, then select the `COPY` command. FileBrowser prompts you to supply a destination.

If you selected just one file, FileBrowser prints the old name and offers a default, which consists of the same file name and either the same directory that was last used in a `COPY` or `RENAME` in this FileBrowser, or the connected directory if this is the first use of `COPY` in this FileBrowser. You can accept the default or supply your own destination file name. If you supply just a directory specification, e.g., `{SERVER}<DIRECTORY>`, the file is copied to that directory under its current name. If you supply a complete name, the file is copied to that exact name.

| Copy file {ERINYES}<LISP>LYRIC>LIBRARY>CHAT.LCOM;1 to new file<br>name: {DSK}CHAT.LCOM |       |              |
|----------------------------------------------------------------------------------------|-------|--------------|
| {ERINYES}<Lisp>Lyric>Library>*.lcom;* at 11:13 FB Commands                             |       |              |
| Total: 99 / 4803 pages Deleted: 0 / 0 pages                                            |       |              |
| Name                                                                                   | Pages | Created      |
| CHARCODETABLES.LCOM;1                                                                  | 8     | 29-Nov-86 17 |
| ▶ CHAT.LCOM;1                                                                          | 59    | 26-Jan-87 22 |
| CHATTERMINAL.LCOM;1                                                                    | 20    | 27-Nov-86 13 |
| CMLFLOATARRAY.LCOM;1                                                                   | 23    | 9-Apr-87 16  |
| COPYFILES.LCOM;1                                                                       | 16    | 27-Nov-86 15 |

|          |    |
|----------|----|
| Delete   | ▶▶ |
| Undelete | ▶▶ |
| Copy     | ▶▶ |
| Rename   | ▶▶ |
| Hardcopy | ▶▶ |
| See      | ▶▶ |
| Edit     | ▶▶ |
| Load     | ▶▶ |

Note: Unless you specify a version number in the destination file name, the version number of the new file will be 1, or one higher than the highest existing version of the file in the destination directory, independent of the version number of the old name.

Even files marked for deletion can be copied.

If you selected several files, FileBrowser notes how many files you wish to copy and offers as a default destination the connected directory. You can accept the default or supply a different directory. All the files are copied to that directory under the names they currently have.

You must supply a directory specification, e.g., `{SERVER}<YOURDIRECTORY>`, rather than a complete file name, since you can't copy multiple files to the same name. If you



mistakenly type a file name, rather than a directory specification, FileBrowser will complain and abort the command.

If you want to copy files from different subdirectories, FileBrowser will ask, via a message in its prompt window, if you want to preserve the subdirectory structure at the destination. If you answer YES, then the names at the destination will include not just the root name of each source file, but also all the subdirectory names below the greatest subdirectory prefix common to all the selected files (this common prefix is displayed as part of the question). If you answer NO, then the names at the destination are formed solely from the root name of each file (the name displayed in the browser), ignoring any directory information each name might have. This can cause multiple files with the same root name to be copied into the same destination name (but with different version numbers, of course).

| Copy 15 files to which directory? {ERINYES}<Medley>Library>A |       |              |
|--------------------------------------------------------------|-------|--------------|
| {ERINYES}<Lisp>Lyric>Library>*.lcom;* at 11:13 FB Commands   |       |              |
| Total: 99 / 4803 pages Deleted: 0 / 0 pages                  |       |              |
| Name                                                         | Pages | Created      |
| ▶ KERMIT.LCOM;1                                              | 83    | 8-Jan-87 19  |
| ▶ KERMITMENU.LCOM;1                                          | 17    | 8-Jan-87 19  |
| ▶ KEYBOARDEDITOR.LCOM;1                                      | 56    | 8-Jan-87 19  |
| ▶ MASTERSCOPE.LCOM;1                                         | 124   | 11-Feb-87 15 |
| ▶ MATCH.LCOM;1                                               | 71    | 8-Jan-87 18  |
| ▶ MATMULT.LCOM;1                                             | 35    | 22-Apr-87 10 |
| MINISERVE.LCOM;1                                             | 10    | 17-Apr-87 11 |
| ▶ MSANALYZE.LCOM;1                                           | 40    | 10-Dec-86 16 |
| ▶ MSPARSE.LCOM;1                                             | 41    | 12-Jan-87 17 |
| NSCHAT.LCOM;1                                                | 32    | 27-Nov-86 13 |
| NSMAINTAIN.LCOM;1                                            | 24    | 28-Jan-87 11 |
| PRESS.LCOM;1                                                 | 80    | 25-Mar-87 11 |

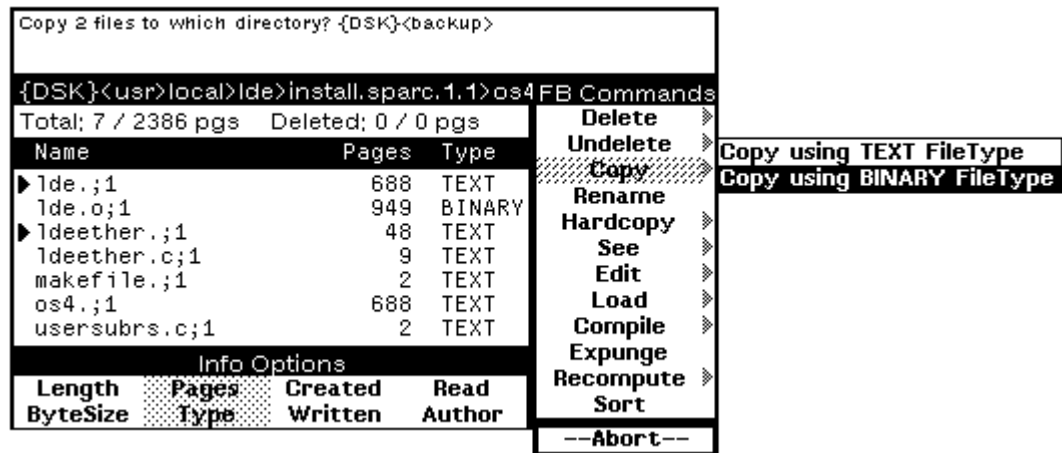
Delete  
 Undelete  
 Copy  
 Rename  
 Hardcopy  
 See  
 Edit  
 Load  
 Compile  
 Expunge  
 Recompute  
 Sort

--Abort--

When copying (or renaming) multiple versions of the same file, FileBrowser does the copying in order of increasing version number, so that the versions at the destination are in the same relative order as at the source.

As each file is copied, FileBrowser prints a message giving the full name of the new file. If a file with the chosen name already exists, the new file's version number will be one higher; otherwise it will be version 1 (one). The new file will have the same creation date as the original file. If the destination file happens to be one that matches the pattern of the files in the browser, the new file is inserted in the appropriate place in the browser display. However, if it matches the pattern of some other FileBrowser, it is not inserted in that other browser's display (in other words, FileBrowsers do not know about each other). You would have to RECOMPUTE the destination FileBrowser to see that the file was copied into it.

Copying files from filesystems that do not normally distinguish between **TEXT** and **BINARY** file types, like UNIX, can sometimes cause problems. On these systems the type of a file is inferred from its extension, or by using the Lisp variables **DEFAULTFILETYPE** and **DEFAULTFILETYPELIST**. At times, there will be files where the system cannot properly infer the type. By selecting the submenu of COPY you can copy files with the right attributes.



In this example, since the chosen files had no extension, the system incorrectly inferred their type from the value of **DEFAULTFILETYPE**. By selecting the **BINARY** copy option, the correct type of the file will be preserved after the operation.

## RENAME

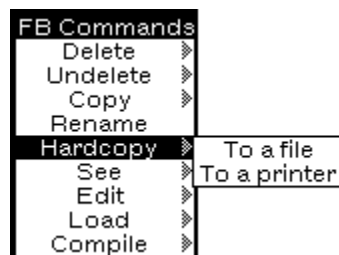
The RENAME command is used for changing the name of a file or group of files, or for moving a file or group of files to a different directory.

The RENAME command is used in exactly the same way as the COPY command. If you rename a single file, you can supply a complete new name or just a directory; if you rename several files, you must specify a directory. As each file is renamed, FileBrowser prints a message giving the file's new name and removes the file from the browser display. If the new name belongs in the same browser, it is inserted in the appropriate place. If for some reason a file could not be renamed, this is noted in the FileBrowser prompt window. The reasons for the failure of a renaming operation are roughly the same as for the failure of an EXPUNGE; the file is open, or you do not have the access rights needed to rename the file.

**Note:** If the destination of the rename is on a different file system than the original file, changing its name is equivalent to copying the file to its new name and then deleting the original file.

## HARDCOPY

You can print text files, TEdit files, Interpress or Press files, and Lisp files from FileBrowser. Select the appropriate file or files, then select the HARDCOPY command. The HARDCOPY command will determine what type of file you are printing and call the appropriate function for printing that file. Then the files will be printed one at a time on your default printer. The prompt window will display status messages telling you when files are being printed and when they are done (if your printer is one that provides this status service).



You may specify printing to a file or to a printer other than the default printer by means of a submenu from the HARDCOPY command. This menu is the same as the one on the HARDCOPY command in the background menu. Selecting TO A PRINTER presents you with a choice of printers from a menu. Selecting TO A FILE prompts you to supply a file name. If you selected a single file, you must specify a single hardcopy file name (or accept the offered default). If you selected multiple files, then you must specify a pattern with a single asterisk somewhere in the "name" field, for example, \*.INTERPRESS or Hardcopy-\*.IP. The output file names are constructed by merging the pattern with each selected file name. If the name includes an extension that implies the type of print format (e.g., .IP or .INTERPRESS implies the InterPress print format), then a file of the specified type is made automatically. Otherwise, you are prompted to supply a print format type.

**Note:** For files stored on servers not supporting random access, FileBrowser is currently unable to determine that a file is in TEdit format unless the file has the extension .TEDIT. Therefore you should use TEdit to hardcopy TEdit files with other extensions. Use FileBrowser's EDIT command (to call TEdit), then the HARDCOPY command either from the TEdit Expanded Menu or from the right-button menu. As of the Lyric release, TEdit files written directly to an NS

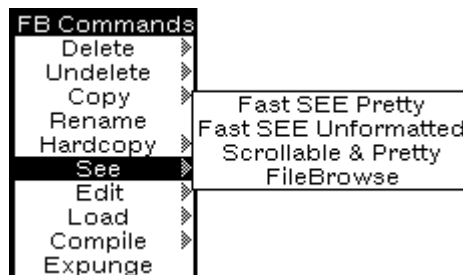
file server are identifiable as TEdit files, so this restriction does not apply to them.

Note: To obtain a hardcopy of the directory itself, use the Hardcopy command from the right-button window menu. See the Getting Hardcopy Directory Listings section.

## SEE

When you browse a directory you sometimes want to see a file before printing or performing some other operation on it. To do this, select the file, then select the SEE command from the command menu. FileBrowser will prompt you to open a window by presenting you with a dashed rectangle and printing a message in the system prompt window. The window will be blank until FileBrowser starts printing the contents of the file in it.

There are actually four different SEE commands, as shown in the submenu for the SEE command. The two FAST SEE commands are provided to let you quickly see the contents of a file, but not do anything fancy, such as scroll around at random in the file. The slower SCROLLABLE & PRETTY command does let you scroll, and if the file contains formatting information of a kind that FileBrowser knows about (via the editors you have loaded), you will see the file formatted. However, this command does much more work, and may take a bit longer to show you even the first line of the file. The FILEBROWSE command is for use on "files" that are actually directories; it is described in the next section.



The two FAST SEE commands display the selected file in the display window one windowfull at a time. When the file fills the window, a small menu appears at the bottom-left corner of the window (or top-left if your display window is at the bottom of the screen) giving you the option of seeing more of the file or aborting the SEE command. If you issued the SEE command with more than one file selected, you also have the choice of aborting just the display of this file or the entire SEE command.

```

Viewing {ERIS}<Lisp>Koto>Lispusers>ACTIVEREGIONS.1
(FILECREATED " 5-JUL-84 17:48:34" {SDHVX1}DISKD:<DOLPHIN.LISPUS
ERS>ACTIVEREG.;8 7633
changes to: (FNS ACTIVEREGIONS/MULTIPLEREGIONS? FINDACTIVEREGIO
N
 ACTIVEREGIONS/DEFAULTHIGHLIGHTFN)
 (VARS ACTIVEREGIONSFNS ACTIVEREGIONSHIDDENFNS)
previous date: "15-MAR-84 08:43:50" {SDRVX1}DISKD:<DOLPHIN.LISP
USERS>ACTIVEREG.;5) **COMMENT**
(PRETTYCOMPRINT ACTIVEREGIONSCOMS)
(RPAQQ ACTIVEREGIONSCOMS (**COMMENT**
 (RECORDS ACTIVEREGION)
 (FNS * ACTIVEREGIONSFNS) **COMMENT**
 (FNS * ACTIVEREGIONSHIDDENFNS)))
COMMENT
[DECLARE: EVAL@COMPILE
(RECORD ACTIVEREGION (REGION HELPSTRING DOWNFN UPFN HIGHLIGHTFN LO
WLIGHTFN DATA))
]
More Next File Abort

```

If you select More, the SEE command displays another windowful of the file. If you select Next File, the SEE command closes this file and goes on to display the next file in the current selection. If you select Abort, the entire SEE command is aborted. You can also abort the SEE command by closing the display window.

The next time you give a FAST SEE command, the same window will be reused.

The only difference between the FAST SEE PRETTY and the FAST SEE UNFORMATTED commands is the manner in which the characters of the file are processed as they are displayed.

The pretty (formatted) version interprets certain control characters found in Lisp source files to be font change commands, and interprets certain multibyte sequences as representing characters in the Xerox extended character set (see *XSIS Character Code Standard*, version *XC1-2-2-0*). It also squeezes out blank lines and shrinks the indentation of indented lines in order to better fit the text in a window that is generally much narrower than the standard file width. The formatted version is thus most appropriate for viewing source files and files containing plain text.

The unformatted version of the SEE command does no special processing on the characters whatsoever. It simply displays each eight-bit byte as a single character, uninterpreted. This means that bytes that do not represent normal printing characters may be displayed as black boxes, in the form ^x or #x, or as a flashing of the window (for the byte that represents the ASCII "bell" character). The Unformatted version is thus most appropriate for viewing binary files that also contain text portions that might be worth seeing; e.g., compiled files (those with extension .MCOM) or Interpress masters (extension .IP or .INTERPRESS).

The SEE SCROLLABLE & PRETTY command views a file in a different way. This command brings up a new read-only TEdit window for viewing a file (only if TEdit is loaded in your system; otherwise, SCROLLABLE & PRETTY reverts to FAST). You can scroll and copy-select the file's contents at will, as with any TEdit window. If the file is a Lisp source file, its contents are first formatted into a TEdit document, so that all the font information is retained. This formatting, however, can take a long time for a large file. For other kinds of files, the SEE SCROLLABLE & PRETTY command is exactly like viewing the file in a regular TEdit window, except that you can't edit it. If you want to edit a file, use the Edit command instead of the See command.

You can keep the display window used by the SEE SCROLLABLE & PRETTY command open as long as you like. The command uses a different window for each file

you select. Simply close the window with the standard right-button window menu when you are finished with it.

## FILEBROWSE

The FILEBROWSE command is a subcommand of SEE used to view a subdirectory in its own FileBrowser window. The selected file must be a (sub)directory. Subdirectory files appear in browsers on XNS file servers when the depth is finite (see the SET DEPTH command), and their names always end in ">".

| Name (depth 1) | Pages | Created      | Undo | Copy | Rename | Hardcopy | See | Edit | Load |
|----------------|-------|--------------|------|------|--------|----------|-----|------|------|
| Examples>      | 139   | 4-Feb-88 12  |      |      |        |          |     |      |      |
| Lisp>          | 1     | 15-Sep-84 15 |      |      |        |          |     |      |      |
| Mail>          | 685   | 26-May-87 11 |      |      |        |          |     |      |      |
| Misc>          | 1     | 17-Jul-87 17 |      |      |        |          |     |      |      |
| Star>          | 273   | 1-Jul-87 17  |      |      |        |          |     |      |      |

Fast SEE Pretty  
Fast SEE Unformatted  
Scrollable & Pretty  
FileBrowse

On UNIX servers, subdirectories are not syntactically distinguishable from ordinary files, nor is Lisp able to distinguish them internally; you simply have to know. The FILEBROWSE command prompts you for a region for a new FileBrowser window group, in which it proceeds to enumerate the contents of the selected subdirectory, to the same depth as the main browser used, if any.

## EDIT

The EDIT command invokes an editor on the selected file. To specify an editor explicitly, use one of the commands on the submenu.

|          |           |
|----------|-----------|
| Hardcopy |           |
| See      |           |
| Edit     | TEdit     |
| Load     | Lisp Edit |
| Compile  |           |

To start up a TEdit editor on a selected text file, select EDIT with the left mouse button. If you have recently closed a TEdit window, then TEdit will probably reuse that window; otherwise, you will be prompted to create an editor window. TEdit only remembers the most recently abandoned window, however, so if you issue the EDIT command when you have several files selected, you will be prompted to create windows for all but the first file.

The subcommand LISP EDIT is appropriate for Lisp source files produced by the file manager. It calls the Lisp structure editor on the file's coms. If the file is not yet known to the file manager, you will be asked whether you want to load it first (using LOAD PROP). If not, the operation is aborted. The editor used is the default structure editor (SEdit or DEdit, depending on your setting of \*EDITMODE\*).

If you select the main Edit command, without sliding off to the submenu, then FileBrowser's default editor is called. This editor is initially TEdit, but you can change the default behavior by setting the variable FB.DEFAULT.EDITOR (see the section "Customizing FileBrowser and Using the Programmer Interface," below).

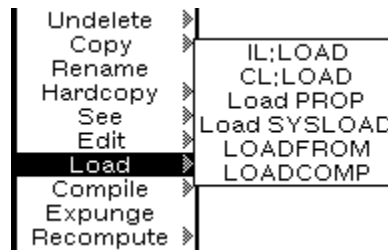
## LOAD

FileBrowser's LOAD command can be used to load both source (interpreted) and compiled files into your workstation's virtual memory. First select the file or files you want to load, then select Load with the left mouse button.

A special display window is opened to give information about the files as they are loaded. When the load is complete, FileBrowser closes the load window.

```
TTY window for LOAD
{ERINYES}<LYRIC>LIBRARY>KEYBOARDEDITOR.LCOM;1
compiled on 8-Jan-87 19:03:57
File created 21-Sep-85 08:03:04
KEYBOARDEDITORCOMS
{ERINYES}<LYRIC>LIBRARY>VIRTUALKEYBOARDS.LCOM;1
compiled on 6-Jan-87 22:41:02
File created 5-Nov-86 16:55:40
VIRTUALKEYBOARDSCOMS
```

The LOAD command also has subcommands that enable you to load files in different ways. These commands are described briefly here; see the *IRM* for further details. Only the LOAD and LOAD SYSLOAD commands are of interest to nonprogrammers. All load commands are placed on the history list. With the exception of LOAD SYSLOAD, all are undoable using Lisp history list commands.



LOAD (same as the Lisp LOAD command) loads the file with LDFLG = NIL. If any functions or variables in the file redefine ones that are already in memory, messages such as "(FOO redefined)" are printed. This command is used for loading source files that you as a programmer plan to make modifications to, or for loading any file that you have doubts about, in which case you might want to be able to undo the load.

CL:LOAD differs from IL:LOAD in that it calls the Common Lisp LOAD function, rather than that of Interlisp. In the present implementation, these two commands are essentially identical.

LOAD PROP loads the file(s) with LDFLG = PROP. Interlisp functions are loaded onto property lists, rather than redefining the functions already in memory. Common Lisp functions, macros, etc. are loaded into a definitions table, again without changing the definitions currently in effect. This command is used for loading Lisp source files for which the compiled version already exists in memory, but which you plan to edit.

LOAD SYSLOAD loads the file(s) with LDFLG=SYSLOAD. Function and variable redefinitions occur quietly (i.e., without printing (FOO redefined), (BAR reset), etc.). The file manager is not informed of this file. This is the fastest loading command and consumes the fewest resources, but it is not undoable. It is the best way to load compiled (extensions LCOM or DFASL) files that you are certain you want to load into your environment and are not planning to edit.

LOADFROM calls the file manager's function LOADFROM. This loads variables and other expressions but not Interlisp functions, and does so in a way that informs the file manager, so that the editor knows where to find the functions.

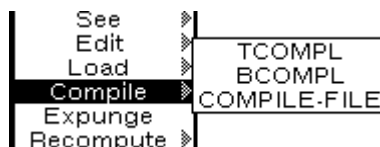
Note: The LOADFROM command is not appropriate for Common Lisp files—it is better to use LOAD PROP for them.

The LOADCOMP subcommand calls the file manager's function LOADCOMP. This loads from the file all the expressions that the compiler would evaluate if compiling the

file—macros, records, and any other expressions enclosed in an EVAL@COMPILE declaration.

## COMPILE

The COMPILE command is used to compile a selected Lisp source file or files. The files do not have to be loaded. The COMPILE command uses the same compiler as CLEANUP does (the value of \*DEFAULT-CLEANUP-COMPILER\*), unless you select a different compiler from the COMPILE submenu.



Note that this command is placed on the history list, so that it and its subcommands are undoable.

A special Executive window is opened for each file to display information about the code being compiled. When the compilation is finished, the window is closed. In the case of TCOMPL and BCOMPL, which invoke the Interlisp compiler, each compiled file is saved on your connected directory with the original file name and the extension MCOM. In the case of COMPILE-FILE, which invokes the Xerox Common Lisp compiler, the compiled file is saved on the same directory as the source file with the original file name and the extension MFASL.

Note that this command compiles files found on a storage device, not the functions defined in the Lisp image. If you have made changes to any of the functions on a loaded file, you must perform a MAKEFILE to write an updated version before compiling it. For more information on making files and compiling, see the *IRM*.

## EXPUNGE

If you are sure you want to delete files permanently, choose the EXPUNGE command. The EXPUNGE command is grayed while FileBrowser expunges the files that were marked for deletion by the DELETE command. As each file is removed from the system, it is removed as well from the browser display, and the tally of total number of files and number of deleted files is updated, so you can see the progress of the command.

If for some reason a file can not be expunged, FileBrowser prints a message saying so in its prompt window, but continues to expunge the other files. The main reasons that prevent a file from being expunged are its being opened, either by you or some other user, or your not having the access rights required to delete it (if it is on a file server). See the section "Troubleshooting Problems with FileBrowser," below.

Note: The EXPUNGE command is not affected by the current selection; it operates only on files marked for deletion, whether currently selected or not.

## RECOMPUTE

FileBrowser's display shows those files that existed and matched the specified pattern at the time you created the browser. If you want the browser to reflect the latest state of the file system, use the RECOMPUTE command.

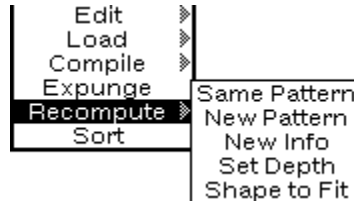
For example, if you open a FileBrowser on your directory, then save several versions of a TEdit file on that directory, the file listing will not display the new versions until you RECOMPUTE.



The RECOMPUTE command operates exactly as when you started up FileBrowser initially: it clears the display and tally windows, then enumerates the files matching the pattern. The RECOMPUTE command in the menu is grayed until the enumeration is finished. During this time you cannot scroll or perform any other operations on the browser. However, you can close the window if you want to abort the command and throw away the browser.

If any files are marked for deletion at the time you request a RECOMPUTE, FileBrowser will present the choice of expunging or undeleting the files, just as it does when you want to quit the browser (see the section "Quitting the FileBrowser," above).

The RECOMPUTE command also has a menu of subcommands that allow you to list different files or different information for the same set of files.



SAME PATTERN is the same as the main RECOMPUTE command, i.e., it enumerates the files matching the same pattern as before.

NEW PATTERN lets you change the pattern, i.e., browse a new set of files. FileBrowser prompts you to supply a new file name pattern and offers the old pattern as an initial default. You can either type an entirely new pattern, replacing the one offered, or delete the old pattern one character at a time by backspacing. Press the carriage return when you have finished specifying the pattern. FileBrowser then enumerates the files matching this pattern, just as with the RECOMPUTE command. You can abort the command with the Abort button, or by erasing the whole pattern (by backspacing or using control-Q) and then pressing the carriage return.

NEW INFO lets you change which attributes the browser displays. It is described in the next section.

SET DEPTH lets you change the depth to which FileBrowser enumerates a directory on an XNS file server. It is described in the section "SET DEPTH".

SHAPE TO FIT reshapes the FileBrowser window so that all the attributes in the display are visible at once, eliminating the need to horizontally scroll the window to get at all the information.

## NEW INFO

FileBrowser displays some file attributes, or information about the file, alongside each file in the browser display. Ordinarily, the attributes displayed are the size of the file in pages, its creation date, and its author. You can change which attributes are displayed for all new FileBrowsers by changing FB.DEFAULT.INFO (see the section "Customizing FileBrowser and Using the Programmer Interface," below). You can change the attributes displayed in a particular browser window by using the NEW INFO command.

To use the NEW INFO command, select it from the submenu of the RECOMPUTE command. FileBrowser opens up an additional subwindow, the Info Options window, below the display window. This subwindow contains a menu of attributes, with the current defaults shaded. Selecting a shaded item unshades it; selecting an unshaded item shades it. When you have selected all the attributes you wish to see displayed, issue either the RECOMPUTE or the NEW PATTERN command. The files will be

listed with the new information you requested. The Info Options window stays open—you can close it at any time.

|                                                                                       |       |             |             |
|---------------------------------------------------------------------------------------|-------|-------------|-------------|
| Select from the lower menu which attributes are to be displayed, then click Recompute |       |             |             |
| {DSK}*,*:* browser                                                                    |       | FB Commands |             |
| Total: 48 / 2838 pgs Deleted: 0 / 0 pgs                                               |       | Delete >    |             |
| Name                                                                                  | Pages | Create      | Undelete >  |
| EDITBITMAP.TEDIT;1                                                                    | 76    | 20-Ma       | Copy >      |
| ETHERRECORDS.TEDIT;1                                                                  | 15    | 20-Ma       | Rename >    |
| FILEBROWSER.TEDIT;2                                                                   | 388   | 3-Ap        | Hardcopy >  |
| FONTSAMPLE.TEDIT;1                                                                    | 17    | 20-Ma       | See >       |
| FTPSEVER.TEDIT;1                                                                      | 13    | 20-Ma       | Edit >      |
| FX-80DRIVER.TEDIT;2                                                                   | 88    | 8-Ap        | Load >      |
| FX-80DRIVER.TEDIT;1                                                                   | 88    | 20-Ma       | Compile >   |
| GCHAX.TEDIT;1                                                                         | 40    | 20-Ma       | Expunge >   |
| GRAPHER.TEDIT;1                                                                       | 67    | 20-Ma       | Recompute > |
| Sort >                                                                                |       |             |             |
| Info Options                                                                          |       |             |             |
| Length                                                                                | Pages | Created     | Read        |
| ByteSize                                                                              | Type  | Written     | Author      |

The Info Options items have the following meanings:

- CREATED** The date and time that the content of the file was created. This date changes whenever the file is modified, but does not change when a file is copied or renamed.
- WRITTEN** The date and time the file was last written to the file system. This date is never older than the Created date, but it can be newer if the file is copied, unmodified, from one file system to another.
- READ** The date and time the file was last read. This attribute may be blank if the file has never been read.
- AUTHOR** The login name of the person who wrote the file, or last modified it.
- LENGTH** The length of the file in (usually eight-bit) bytes.
- PAGES** The number of 512-byte pages in the file. On some servers, this attribute is blank if the file is empty.
- BYTESIZE** The size (in bits) of the bytes in the file. In Xerox Lisp this is always eight, but some older computers and file servers allow other sizes.
- TYPE** A value indicating what kind of data the file contains. The usual values of this attribute are TEXT, meaning the file contains just characters, or BINARY, meaning the file contains arbitrary data. Some servers have additional types, such as INTERPRESS for files in Interpress format.

## SET DEPTH

XNS file servers support a feature that allows enumerating a directory to a user-specifiable depth. The "depth" of a file reflects the number of subdirectories between it and the root of the enumeration, i.e., the directory or subdirectory you gave in the pattern to FileBrowser, not counting any containing wildcards (asterisks). The immediate descendants of the root are at depth 1, files in subdirectories of depth 1 are at depth 2, and so on.

Ordinarily, FileBrowser enumerates a directory to the default depth, which is usually unlimited. To enumerate a directory to a different default, use the FB command with argument :DEPTH *n*, for some positive integer *n*, or T for unlimited depth. To change

the depth in an existing FileBrowser, use the SET DEPTH command, a subcommand under the RECOMPUTE command. The command offers you a menu of choices:

|                                               |
|-----------------------------------------------|
| Global default<br>Infinite<br>1<br>2<br>Other |
|-----------------------------------------------|

"Global default" means use the default depth, overriding the depth at which this browser was last enumerated. "Infinite" means use no depth limit (same as depth T). "1" and "2" are common depth choices; to choose some other numeric value, select "Other" and enter the value via the displayed keypad.

The SET DEPTH command does not affect the current display. It takes effect the next time you use the RECOMPUTE or FILEBROWSE commands from the same browser.

During a RECOMPUTE, if a subdirectory appears at the specified maximum depth, its descendants are not enumerated; rather, the subdirectory itself appears as an entry in the browser display. This entry can be selected, just like a file, but only a small number of commands can be used on it: you can RENAME it, you can DELETE it if it has no descendants, and you can FILEBROWSE it. It has attributes, just as ordinary files do. Its page size is the size of the entire subtree rooted at the subdirectory.

Note: Depth currently affects only XNS servers; all other devices ignore it and enumerate to their own default depths. In addition, due to a bug in XNS Services 10, depth is ignored for nontrivial patterns, i.e., anything but ".\*.\*".

## SORT

The SORT command allows you to sort the files in the browser by any attribute of the files displayed in the browser. Selecting SORT brings up a menu of attributes by which to sort. This menu includes all the attributes currently displayed in the browser (such as Creation Date, Author), plus the choice Name. For some attributes you can sort forward or backwards; the choice is on a submenu, and the default is generally in the order of numerically greatest (e.g., size) or most recent (e.g., creation date) first.

If the attribute you select is not Name, then the file names displayed in the browser will be reformatted to include their directory portion (if there are any subdirectories below the browser's main pattern), as the subdirectory information is no longer implicit in a file's position in the browser.

The sort order Name, Decreasing Version is the default order in which browsers initially are created.

---

## Customizing FileBrowser and Using the Programmer Interface

---

FileBrowsers are created programmatically by the function FILEBROWSER, or by type-in from an Executive window with the FB command.

Note: All functions, variables and literals in this section are in the Interlisp package. You'll have to use the prefix IL: if you are using another Executive.

### FileBrowser Functions

(FILEBROWSER *FILESPEC ATTRIBUTES OPTIONS*)

[Function]

Creates a FileBrowser on files matching the pattern *FILESPEC* and displaying the values of the specified *ATTRIBUTES* for each file. Returns the main window of the browser.

If *ATTRIBUTES* is missing or NIL, it defaults to the value of FB.DEFAULT.INFO (below). See FB.INFO.MENU.ITEMS for the complete set of allowable attributes.

*OPTIONS* is a list in property-list format. The currently implemented properties and their values are as follows:

- :REGION A screen region in which FILEBROWSER will open the browser; if this option is omitted, FileBrowser will prompt you for a region.
- :DEPTH The depth to which the enumeration should be performed. Affects only XNS servers (see SET DEPTH).
- :TITLE The title for the main browser. If this is omitted, the title derives from *FILESPEC*, and is updated whenever the RECOMPUTE command is used.
- :MENU-TITLE The title for the command menu. Defaults to "FB Commands".
- :MENU-ITEMS The set of ITEMS composing the command menu. The default is the value of FB.MENU.ITEMS.

FB *FILESPEC ATTR1 . . . ATTRN*

[Command]

This is an Executive command for creating FileBrowsers. FB creates a FileBrowser on files matching the pattern *FILESPEC* and displaying the attributes *ATTR1* through *ATTRN*, or the value of FB.DEFAULT.INFO if no attributes are specified. It prompts you for a window region. If a keyword appears in the command line, the remainder of the line from that point on is interpreted as the *OPTIONS* argument to FILEBROWSER.

For example, the Executive command

```
FB *.MCOM LENGTH CREATIONDATE
```

browses all files on the connected directory with extension MCOM, displaying the length in bytes and creation date for each. The command

```
FB * :DEPTH 1
```

browses the connected directory to depth 1, displaying the default attributes.

FB always returns NIL.

## FileBrowser Variables

There are several global variables that can be altered to affect FileBrowser's behavior. You can set them by typing

```
(SETQ VARIABLENAME NEWVALUE)
```

to your Executive window, and can save their values with a VARS command in your initialization file.

FB.DEFAULT.INFO

[Variable]

A list specifying which attributes should be displayed for each file. The elements of this list are the Lisp names for the attributes you want displayed.

The choices are CREATIONDATE, WRITEDATE, READDATE, LENGTH, SIZE, BYTESIZE, AUTHOR, and TYPE. The attribute SIZE corresponds to the info item "Pages".

For example,

```
(SETQ FB.DEFAULT.INFO ' (CREATIONDATE LENGTH))
```

would cause all new FileBrowsers to display exactly the attributes creation date and length in bytes for each file.

FB.INFO.MENU.ITEMS [Variable]

The list of items in the menu used by the NEW INFO command. Each element of the list is of the form (*LABEL ATTRIBUTE "DOCUMENTATION"*). If you add new attributes to this variable, you should also add corresponding entries to FB.INFO.FIELDS.

FB.INFO.FIELDS [Variable]

A list describing, for each attribute, the format in which it is displayed. In addition, the order of attributes in this list is the order in which they are displayed in a browser window. Each element is of the form (*ATTRIBUTE HEADER WIDTH FORMAT PROTOTYPE*), where

*ATTRIBUTE* is the name of the attribute, which must be a valid attribute for GETFILEINFO

*HEADER* is a string displayed in the header line above the main browser window

*WIDTH* is the total width in pixels to allocate for printing the values of this attribute, including trailing space

*FORMAT* is NIL for ordinary values, DATE for attributes whose value is a date, or (*FIX N*) for integer values

*PROTOTYPE* is a string describing the widest value you expect the field to have

All values are printed left-justified in the allotted space, except for format (*FIX N*), used for integer values, which are right-justified in a field *N* pixels wide, with *WIDTH-N* pixels left for trailing space.

If *PROTOTYPE* is present, then the *WIDTH* and *N* fields for the item are ignored, and the width is taken to be the width of the prototype string in the browser's font (FB.BROWSERFONT), plus two characters' worth of space between columns.

FB.DEFAULT.NAME.WIDTH [Variable]

The amount of space, in points, to use for displaying file names in a browser, initially 140. The name column is automatically expanded if enough names are too wide. You can set this larger if you routinely browse directories of long file names.

FB.ICONFONT [Variable]

The font in which the file pattern is displayed on the browser icon, initially eight-point Helvetica. The value of this variable should be a font descriptor, as returned by FONTCREATE.

For example,

```
(SETQ FB.ICONFONT (FONTCREATE 'MODERN 10 'BOLD))
```

FB.BROWSERFONT [Variable]

The font in which the information in the main display window is printed, initially 10-point Gacha.

FB.PROMPTFONT [Variable]

The font in which prompt messages are printed, initially eight-point Gacha.

FB.PROMPTLINES [Variable]

The number of lines in the prompt window, initially three.

FB.HARDCOPY.FONT [Variable]

Specifies the font to use when producing hardcopy directory listings. Initially NIL, which means to use the font class DEFAULTFONT.

FB.HARDCOPY.DIRECTORY.FONT [Variable]

Specifies the font to use for subdirectory names, if there are any, in hardcopy directory listings. Initially NIL, which means to use the font class ITALICFONT.

FB.DEFAULT.EDITOR [Variable]

Specifies the editor to call by default when the main Edit command is selected. Its value is one of the following:

TEDIT Use the TEdit text editor.

LISP Use the Lisp structure editor, as in the Lisp Edit subcommand.

NIL Use the Lisp structure editor if the selected file is a File Manager Lisp source file, TEdit otherwise.

Other Names the entrypoint function of the editor of choice. The editor is called with a single argument, the file name.

The initial value of FB.DEFAULT.EDITOR is TEDIT.

FILING.ENUMERATION.DEPTH [Variable]

The system variable that controls the default depth to which a directory is enumerated. The value is a positive integer, or T for unlimited depth. The initial value is T.

## Adding FileBrowser Commands

You can add your own commands to FileBrowser by adding items to FB.MENU.ITEMS and writing functions to handle the commands. This section describes the format of menu commands and a set of functions that are useful for the implementation of FileBrowser commands.

FB.MENU.ITEMS [Variable]

A list of the items that appear on FileBrowser's command menu. You can add new FileBrowser commands by adding new elements to the end of this list. After your change, any new FileBrowser will have the added commands.

Each element of `FB.MENU.ITEMS` is of the form `(LABEL YOURFN "EXPLANATION")`, where

*LABEL* is the name of the command, as it is to appear in the menu;

*YOURFN* is the name of the function to be called when the command is selected

*EXPLANATION* is the explanation to be printed when the mouse cursor is held over the command

While *YOURFN* is Executing, the menu command is grayed out, and FileBrowser is "locked" so that no other commands or processes can access it.

You can have subcommands as well if you make the menu command be of the form `(LABEL YOURFN "EXPLANATION" (SUBITEMS item1. . . itemN))`, where each *itemJ* is recursively of the same form as a menu command.

FileBrowser calls *YOURFN* with four arguments: `(YOURFN BROWSER KEY ITEM MENU)`, as follows:

*BROWSER* is FileBrowser object in control of this browser window.

*KEY* is the mouse key pressed (left or middle).

*ITEM* is the menu item that was selected.

*MENU* is the command menu.

*YOURFN* can also be a list of two elements, `(FN ARG)`, where *ARG* is an arbitrary value that is passed, unevaluated, as the fifth argument to *FN*. This is useful for writing one function that implements several subcommands that are similar to the original command.

Any additions to `FB.MENU.ITEMS` should be saved with the file manager command `APPENDVARS`, so that the new items are added to the end of the menu, and your changes will not interfere with any changes to built-in FileBrowser commands in new FileBrowser releases.

`(FB.TABLEBROWSER BROWSER)` [Function]

Returns the `TABLEBROWSER` object belonging to FileBrowser *BROWSER*. See the documentation for the module `TABLEBROWSER` for further operations you might perform on one of these browsers.

`(FB.SELECTEDFILES BROWSER NOERRORFLG)` [Function]

Returns a list of table items representing the files currently selected in *BROWSER*. If there are no selected files, this prints

**No files are selected**

in the prompt window, unless *NOERRORFLG* is true, in which case this function quietly returns `NIL`.

`(FB.FETCHFILENAME ITEM)` [Function]

Returns the full name of the file denoted by *ITEM*, one of the table items returned by `FB.SELECTEDFILES`.

## Troubleshooting Problems with FileBrowser

---

When FileBrowser returns the message "No files in group *FILENAMEPATTERN*" when you know those files exist, the file server is probably down or rejecting connections. If this is so, your only option is to wait until the server is functioning again, and then give the Recompute command. In the case of an NS file server, the enumeration of files can also fail if you do not have sufficient access privileges; this condition is usually noted by a message in the system prompt window.

When you try to expunge a file and FileBrowser displays the message "Can't expunge *FILENAME*," it may be because you don't have write access to the file, or someone else is reading the file. However, the most common reason is that the file is still open. Be sure to close any TEdit windows in which you may still be viewing the file. If you have recently issued a *HARDCOPY* command for the file, a background process may still be working on the file. If that's not the problem, you can get a list of open streams by typing *(OPENP)* at the prompt in an Interlisp Executive window. If one of them is open on the file you want to delete, you can close it by passing the stream to the function *CLOSEF*. To close all open streams (not recommended unless you're sure it is okay), you can type *(MAPCAR (OPENP) 'CLOSEF)*. If you don't find an open stream, and the server is a Leaf or XNS file server, you may have a disagreement between Lisp and the server on what files are open, something that can occur if you had aborted an *OPENSTREAM* operation. Call *(BREAKCONNECTION "servername")* to reset the connection, then try again.



[This page intentionally left blank]

# FontSAMPLE

---

FontSample provides a set of tools for generating Interpress masters for a font sampler, and is intended to allow you to see what results on a printer when you specify a font. This is useful because there may be several font substitutions between when you specify a font (for example in TEdit) and when a printer actually renders the font.

The set of font mappings is a function of the local font substitutions in a particular workstation, the workstation's environment (which fonts are available to it at that time, if the font file server is temporarily unavailable), which printer is being used, and which font files are currently installed on the printer.

For example, Lisp might substitute Terminal for Gacha, and the printer might substitute Modern for Terminal. Thus you could request Gacha and get Modern. The font sampler is intended to display the final result of these substitutions.

## Requirements

---

FontSHEETsx.IP (where x = 1, ... 7)

## Installation

---

Load FontSAMPLE.LCOM from the library.

## User Interface

---

To find the cumulative effect of all font substitutions at a given time, environment, and printer, you should generate the Interpress masters (which reflect the environmental and Lisp mappings) and then send them to various printers (which reflect the printer mappings).

```
(SEND.FILE.TO.PRINTER 'FontSHEET1.IP)
(SEND.FILE.TO.PRINTER 'FontSHEET2.IP)
etc.
```

Short sample masters for all currently supported InterPress fonts can be found in the Lisp Library with the names FontSHEET1.IP, FontSHEET2.IP ... FontSHEET7.IP. These reflect the default Lisp font mappings.

Note: These environmental mappings may change from release to release, so new masters should be printed for every release.

## Functions

---

(FNT.MAKEBOOK *OUTROOTNAME LISTOFFONTS PRINTFN*  
*PERPAGE*) [Function]

This function enumerates fonts across multiple output files. Multiple files are necessary because some printers cannot handle documents with many fonts. The function iterates over the fonts in *LISTOFFONTS*, invoking *PRINTFN*

(which has arguments as in `FNT.DISPLOOK`) *PERPAGE* times per output file. The files are named *OUTFROOTNAME* with the page number concatenated at the end. If *LISTOFFONTS* is the atom `ALL`, then `FONTSAVAILABLE` is invoked for all Interpress fonts (this is an extremely slow operation). *PRINTFN* defaults to `FNT.DISPLOOK`; *PERPAGE* defaults to 18. Each font in *LISTOFFONTS* should be a `FONTLIST`.

`(FNT.DISPTBLE STREAM FONT)` [Function]

This function prints a description of the font and characters 0 to 254 of the font specified by *FONT* on *STREAM*. It expects a letter-sized output stream. If used with `FNT.MAKEBOOK`, *PERPAGE* should be 1.

`(FNT.DISPLOOK STREAM FONT)` [Function]

This function prints a description of the font and representative characters of *FONT* on *STREAM*. If used with `FNT.MAKEBOOK`, *PERPAGE* should be 18.

## Limitations

---

The font sheets are applicable only to InterPress printers.

## Example

---

```
(SETQ FONTLISTLIST (LIST ' (MODERN 10 MRR 0 INTERPRESS)
' (CLASSIC 8 BRR 0 INTERPRESS)))
(FNT.MAKEBOOK 'FOO FONTLISTLIST 'FNT.DISPTBLE 1)
```

generates two files named `FOO1` and `FOO2` each containing output from one invocation of `FNT.DISPTBLE`. Thus `FOO1` has a font table for `(MODERN 10 MRR 0 INTERPRESS)`, `FOO2` has a font table for `(CLASSIC 8 BRR 0 INTERPRESS)`.

[This page intentionally left blank]

# FTPSEVER

---

FTPSever implements a simple PUP FTP server protocol for a Xerox workstation. The server is typically run as a background process on one machine to allow other machines remote access to the files on its disk.

## Requirements

---

Ethernet connection to a remote host.

## Installation

---

Load `FTPSEVER .LCOM` from the library.

## Functions

---

(`FTPSEVER FTPDEBUGLOG`)

[Function]

Creates a process named `FTPSEVER` that listens on the standard PUPFTP server socket for incoming connection requests. When one arrives, `FTPSEVER` services it, then returns to its listening state. The process continues to run until killed.

If `FTPDEBUGLOG` is non-NIL, it should be an open file/stream to which tracing information is printed during the life of the process.

If `FTPDEBUGLOG` is T, output goes to a newly created window.

`FTPDEBUGLOG` can also be a `REGION`, specifying where the window is to be created.

`FTPSEVER.DEFAULT.HOST`

[Variable]

Initially `DSK`. This is the default host for files requested of the server via FTP. Setting this to `FLOPPY`, for example, would serve files off the machine's floppy drive.

Note: `FTPSEVER.DEFAULT.HOST` can also be set to the name of a remote host, but this has limited utility, as it does not handle passwords correctly.

## Limitations

---

The current implementation is a simple tool which allows file transfer between Xerox machines and supports only one remote connection at a time. Because of this, files cannot be loaded indirectly, i.e., via the filecoms of another file.

For example, suppose `FOO` loads `BAR` which loads `WOO`. When `FOO` is being loaded, it will attempt to load `BAR`. But `FTPSever` cannot support the second connection

required to load BAR while the first connection is still open to load FOO. (This is similar to the case of trying to load FOO and BAR when they are on different floppies.)

Therefore, you should load files in an order that prevents recursive loads: in this example, load WOO, then BAR, then FOO.

Delete (DELFIL) operation is now supported. Rename (RENAMEFILE) operation is not implemented. FTPServer is best suited for simple COPYFILE operations.

## Examples

---

An alternative way of specifying the host from the remote machine is to make the host name be the "device" field of the file name specification.

For example, if machine M is running FTPServer, another machine could ask for directory of {M}FLOPPY:FOO.\* to get a listing of M's {FLOPPY}FOO\*.

To address your host, you may use the results of ETHERHOSTNAME. If on your host (ETHERHOSTNAME NIL T) evaluates to 123#456#, then on a remote machine you can access file FOO on the host by:

{123#456#}FOO

[This page intentionally left blank]

# FX-80DRIVER

---

FX-80Driver prints text and graphics on Epson FX-80-compatible printers. It implements a full device-independent graphics interface for the FX-80, and can print source code, TEdit documents, bitmaps and windows at a variety of qualities and speeds.

The FX-80Driver contains two printer drivers: a fast driver, for quick printing of draft-quality text, and a high-quality driver, for slower printing of mixed-font text and graphics. You can print early revisions of a document in fast mode, and then switch to high-quality mode for the final copy. The matrix shown in Figure 1 illustrates the capabilities of each mode:

|                  | Fast          | High-quality |
|------------------|---------------|--------------|
| TEdit            | monofont only | yes          |
| Sketch           |               | yes          |
| Windows          |               | yes          |
| Lisp source code | monofont only | yes          |
| Grapher          |               | yes          |

**Figure 1. FX-80 printer drivers**

For historical reasons, FX-80 in this document refers to any and all of the Epson FX-80 family of dot-matrix graphics printers. The module supports the FX-80, FX-85, FX-86 and FX-286. The Epson printers vary in speed and carriage width, but share a common command language.

## Requirements

---

RS232 or TTY cable (see the wiring diagrams in the Introduction of this manual).

Serial interface card in the printer.

DLRS23C or DLTTY.

## Installation

---

### FX-80 Serial Interface

The FX-80Driver module requires that your Epson be equipped with a suitable serial interface (such as the Hanzon Universal Card).



The interface should be set up with XOn/XOff flow control enabled, 9600 baud or slower, 1 stop bit, 8 bit characters, no parity.

(See *The Hanzon Universal Card* booklet for instructions on the DIP switch settings.)

## FX-80 DIP Switch Settings

The FX-80 should have its DIP switches set as shown in Figure 2.

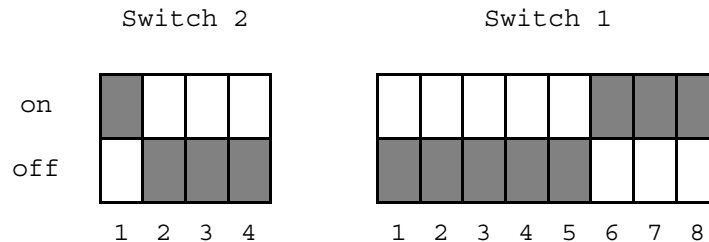


**Figure 2. FX-80 DIP switch settings**

Switch 1 says no automatic linefeed, no automatic paper feed, no buzz on paper-out, and to allow no software deactivation of the printer.

Switch 2 says to use the USA character set, Pica type, allocate 2KB for user-defined characters, allow paper-out detection, and print zeros as zeros.

Note: For the FX-85, -86 and -286 DIP switch settings, consult the corresponding *Epson User's Manual*.

## Software

Load `FX-80DRIVER.LCOM` and the required `.LCOM` modules from the library.

Store all of the font files (file names ending with `.displayfont`) corresponding to the fonts you wish to use on some convenient directory or directories.

`HQFX80-FONT-DIRECTORIES` should be a list that contains these directories; it should be the same as `DISPLAYFONTDIRECTORIES`.

Set `FASTFX80-DEFAULT-DESTINATION` (determines where output to the FASTFX80 lineprinter device goes) and `HQFX80-DEFAULT-DESTINATION` (determines where output to the HQFX80 lineprinter device goes) to one of the following values; they need not be the same:

| Destination | RS232 port | TTY port  | file     |
|-------------|------------|-----------|----------|
| Value       | {RS232}    | {TTY}     | FileName |
| Speed       | 9600 max.  | 4800 max. | n/a      |

Load the appropriate device driver for each of these destinations: `DLTTY.LCOM` for the TTY port, and `DLRS232C.LCOM` for the RS232C port.

Run the function `RS232C.INIT` or `TTY.INIT` (as appropriate), and set the baud rate to match the setting on the printer.

## User Interface

---

You can set up the FX-80 to be your default printer, send FX-80 output to a file for later printing, or programmatically open an image stream that produces output on the FX-80.

Having the FX-80 set up as your default printer means that you can print the contents of windows by selecting the `HARDCOPY` menu item on the window of interest. You can also use the `HARDCOPY - TO A FILE` submenu item to spool your output for later printing. And you can write programs that use the `OPENIMAGESTREAM` function to create FX-80 format graphics output.

## Printing in Fast Mode

---

You can print in fast mode by sending output to the printer `FASTFX80` or by opening an image stream to a file with extension `FASTFX80`. This mode is called fast because it uses the printer's built-in font, which allows a tight encoding of the document to be printed. Fidelity to the original document is not as good as in high-quality mode.

The following restrictions apply:

- Special characters (that is, most Xerox Network Systems extended characters, such as the bullet or dagger; see `CharCodeTables`, `VirtualKeyboards` in this manual) are ignored.
- Only one font is supported (though roman, italic, and bold typefaces do work).
- Graphics (lines, underlines, bitmaps) are ignored.
- Multiple column output does work.

### Set FX-80 Fast Mode

To set your default printer to be a fast mode FX-80, make the list `(FASTFX80 FASTFX80)` the `CAR` of the list `DEFAULTPRINTINGHOST`.

### Set FX-80 Destination

To set the default destination of all output to `{LPT}.fastfx80`, set the variable `FASTFX80-DEFAULT-DESTINATION` to an appropriate file name string. See the table above; the file name could be that of a regular file like `{DSK}SPOOLED-FAST-OUTPUT`.

### Set FX-80 Page Size

To set the driver's page size to match the paper in the printer, set the two variables `\FASTFX80.INCHES-PER-PAGE` (page height in inches) and `\FASTFX80.INCHES-PER-LINE` (page width in inches) to appropriate values. The defaults are 11 and 8.5, respectively. These can be set in your Lisp `INIT` file.

### Print a File

Select the `HARDCOPY` command from the background (right-button) menu. The system first formats the file for printing. Then, when the FX-80Driver actually starts transmitting to the printer, a small abort window, bearing the name of the document and the name of the printer, will appear near the top of your screen.

## Abort a Print Job

Clicking on the item marked `ABORT` in the print window will cleanly terminate the printing of the document.

Note: After aborting a print job, you may need to turn the printer off and back on to make sure that other files will print successfully.

## Printing in High-Quality Mode

---

Print in high-quality mode by sending output to the printer `HQFX80`, or by opening an image stream on a file with type `HQFX80`. High-quality mode printing supports all of Xerox Lisp's device-independent graphics operations, as well as multiple font printing and the XNS extended character set. It prints at 72 dot-per-inch resolution. Fidelity to the original document is better than in fast mode, though printing speed is slower.

### Set HQ Mode

To set your default printer to be a high-quality FX-80, make the list `(HQFX80 HQFX80)` the CAR of the list `DEFAULTPRINTINGHOST`. You can use the `PRINTERMENU` module or your favorite structure editor to do this.

### Set Destination

To set the default destination of all output to `{LPT}.hqfx80`, set the variable `HQFX80-DEFAULT-DESTINATION` to an appropriate file namestring. This could be `"{TTY}"`, `"{RS232}"`, or even the name of a regular file like `"{DSK}spooled-hq-output"`.

### Set Page Size

To set the driver's page size to match the paper in the printer, set the two variables `\HQFX80.INCHES-PER-PAGE` (page height in inches) and `\HQFX80.INCHES-PER-LINE` (page width in inches) to appropriate values. The defaults are 11 and 8.5, respectively. These can be set in your Lisp `INIT` file.

### Print a File

Select the `HARDCOPY` command. The system first formats the file for printing. Then, when the FX-80Driver actually starts transmitting to the printer, a small abort window, bearing the name of the document and the name of the printer, will appear near the top of your screen.

Note: After printing a document on `HQFX80`, you may need to turn the printer off and back on before you can print with `FASTFX80` on that printer.

### Abort a Print Job

See above.

## FX Printer Compatibility

(FX80-PRINT &KEY THING-TO-PRINT LANDSCAPE? COMPRESS?  
HIGH-QUALITY?)

[Function]

*THING-TO-PRINT* may be one of a window, bitmap, or file path name. If *THING-TO-PRINT* is a path name, the file will be treated as either a TEdit or Lisp source file, and printed in the appropriate style.

In the window or bitmap cases, *LANDSCAPE?* specifies landscape printing (X-coordinates run down the left margin) when non-NIL;

*COMPRESS?* specifies FX-80 compressed printing mode.

If *HIGH-QUALITY?* is non-NIL and *THING-TO-PRINT* is a path name, output will be sent to the default high-quality FX-80 printer, otherwise to the default fast FX-80 printer.

The *LANDSCAPE?*, *COMPRESS?*, and *HIGH-QUALITY?* arguments all default to NIL.

## Limitations

Landscape printing has not been implemented.

## Examples

Send text output to fast FX-80:

```
(SETQ FX-80 (OPENIMAGESTREAM "{LPT}.FASTFX80"))
(CL:FORMAT FX-80 "HELLO, WORLD~%%")
(CL:CLOSE FX-80)
```

Print source code on fast FX-80 (assuming the FastFX80 is not your default printer, but is on the list DEFAULTPRINTINGHOST):

```
(LISTFILES (HOST FASTFX80) "{DSK}MYPROGRAM")
```

Note: Source code is stored in pre-pretty-printed form on the file. The pretty-printer's default linelength (width of a line in characters) is normally 102, which is too wide for the FastFX-80's 8.5-inch wide page. To create source files which print nicely on the fast FX-80, you should set the variable `FILELINELENGTH` to a more appropriate value before you `MAKEFILE`. 70 works nicely on 8.5-inch paper with a standard font profile, though your mileage may vary.

Print source code in the the fast FX-80 mode, assuming the FastFX80 is your default printer:

```
(LISTFILES "{DSK}MYPROGRAM")
```

Print TEdit file in fast FX-80 mode, assuming the FastFX80 is your default printer:

```
(LISTFILES "{WAYCOOL:}<PUBLIC>GENSYM.TEDIT")
```

Print text and graphics in high-quality mode:

```
(SETQ FX-80 (OPENIMAGESTREAM "{LPT}" 'HQFX80))
(MOVETO 300 300 FX-80)
```

```
(CL:FORMAT FX-80 "HELLO, WORLD~%%")
(DRAWCIRCLE 300 300 230 ' (ROUND 8) NIL FX-80)
(CL:CLOSE FX-80)
```

Print source code in high-quality mode, assuming the high-quality FX-80 is not your default printer, but is on the list DEFAULTPRINTINGHOST:

```
(LISTFILES (HOST HQFX80) "{DSK}MYPROGRAM")
```

Note: See the previous note regarding FILELINELENGTH and the fast FX-80. The same holds for high-quality FX-80 printing, and we recommend 70 as the value for FILELINELENGTH.

Print source code in high-quality mode, assuming the high-quality FX-80 is your default printer:

```
(LISTFILES "{DSK}MYPROGRAM")
```

Print TEdit file in high-quality mode, assuming the high-quality FX-80 is your default printer:

```
(LISTFILES "{WAYGNARLY:}<PUBLIC>MAGNUMOPUS.TEDIT")
```

[This page intentionally left blank]



# GCHAX

---

GCHax contains functions that are useful for tracking down storage leaks, i.e., objects that should be garbage but do not get garbage collected. There are functions for examining reference counts, locating pointers to objects, and finding circularities (which are among the chief culprits in storage leaks).

Typically, you might turn to GCHax when you notice that `STORAGE` claims there are more instances of a data type in use than you believe there should be.

## Installation

---

Load `GCHAX.LCOM` from the library.

## Functions

---

### Storage

The function `STORAGE` displays statistics on the amounts and distribution of the virtual memory space that has been allocated. If you suspect your program may have storage leaks (e.g., because `(VMEMSIZE)` keeps growing without obvious reason), this function is the place to start to get an indication of which kinds of objects are not being reclaimed. `STORAGE` is part of the standard Lisp sysout; you need not have loaded GCHAX to use it.

`(STORAGE TYPES PAGE-THRESHOLD IN-USE-THRESHOLD)` [Function]

With no arguments, `STORAGE` displays statistics for all data types, along with some summary information about the space remaining. The optional arguments let you refine the display.

If *TYPES* is given, `STORAGE` only lists statistics for those types. *TYPES* should be a type name or list of type names.

If *PAGE-THRESHOLD* is given, then `STORAGE` omits types that have fewer than *PAGE-THRESHOLD* pages allocated to them. The default *PAGE-THRESHOLD* is 2, so types that are not currently in use (consume no storage) do not appear unless you specify a *PAGE-THRESHOLD* of zero.

If *IN-USE-THRESHOLD* is given, then `STORAGE` omits types that have fewer than *IN-USE-THRESHOLD* instances in use (allocated and not yet freed).

For example, `(STORAGE ' (ARRAYP BITMAP) )` lists only statistics for the types `ARRAYP` and `BITMAP`; `(STORAGE NIL 6)` lists only statistics for data types that have at least six pages allocated. `(STORAGE NIL NIL 100)` lists only statistics for data types that have at least 100 instances still in use.

The `STORAGE` function displays, for each Lisp data type, the amount of space allocated to the data type, and how much is currently in use. The display looks something like this:

| Type | Assigned<br>pages | [items] | Free items | In use | Total alloc |
|------|-------------------|---------|------------|--------|-------------|
| FIXP | 66                | 8448    | 7115       | 1333   | 447038      |



|        |      |         |      |         |         |
|--------|------|---------|------|---------|---------|
| FLOATP | 24   | 3072    | 2412 | 660     | 734877  |
| LISTP  | 2574 | ~298584 | 5294 | ~293290 | 3545071 |
| ARRAYP | 8    | 512     | 245  | 267     | 48199   |
| . . .  |      |         |      |         |         |

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Type        | Is the name of the data type, as given to DATATYPE or the Common Lisp DEFSTRUCT.                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Assigned    | Is how much of your virtual memory is set aside for items of this type. Memory is allocated in quanta of two pages (1024 bytes). The numbers under Assigned show the number of pages and the total number of items that fit on those pages. The tilde (~) on the LISTP line indicates that the number is approximate, since cdr-coding makes the precise counting of lists impossible—the amount of memory consumed by any particular list cell varies depending on its contents and how it was allocated. |
| Free items  | Shows how many of the assigned items are available to be allocated (by the Interlisp create or the Common Lisp make- constructs); these constitute the free list for that data type.                                                                                                                                                                                                                                                                                                                       |
| In Use      | Shows how many items of this type are currently in use, i.e., have pointers to them and hence have not been garbage collected. If this number is higher than your program seems to warrant, you may want to look for storage leaks. The sum of Free items and In Use is always the same as the total Assigned items.                                                                                                                                                                                       |
| Total Alloc | Is the total number of items of this type that you have ever allocated (created), or at least since the last call to BOXCOUNT that reset the counter.                                                                                                                                                                                                                                                                                                                                                      |

STORAGE also prints some summary information about how much space is allocated and available collectively for fixed-length items (mainly data types, both user and built-in), variable-length items (arrays, bitmaps, strings), and symbols. The variable-length items have fixed-length headers, which is why they also appear in the printout of fixed-length items. For example, the line printed for the data type BITMAP says how many bit maps have been allocated, but the figure displayed as "assigned pages" counts only the headers, not the space used by the variable-length part of the bitmap. The variable length portion is accounted in the summary statistics for "ArrayBlocks", where it is lumped with all other users of variable-length space, as it is not possible for the system to more finely discriminate the users of the space.

#### Data Spaces Summary

|                              | Allocated<br>Pages | Remaining<br>Pages |
|------------------------------|--------------------|--------------------|
| Datatypes (incl. LISTP etc.) | 4370               | \                  |
| ArrayBlocks (variable)       | 5770               | -- 47758           |
| ArrayBlocks (chunked)        | 2626               | /                  |
| Symbols                      | 1000               | 1048               |

#### variable-datum free list:

|          |           |              |
|----------|-----------|--------------|
| le 4     | 18 items; | 72 cells.    |
| le 16    | 84 items; | 865 cells.   |
| le 64    | 38 items; | 1019 cells.  |
| le 256   | 76 items; | 7580 cells.  |
| le 1024  | 2 items;  | 1548 cells.  |
| le 4096  | 11 items; | 18568 cells. |
| le 16384 | 1 items;  | 4864 cells.  |
| others   | 2 items;  | 59565 cells. |

Total cells free: 94081 total pages: 736

In the summary, Remaining Pages indicates how many more pages are available to be allocated to each type of datum. There is a single figure for both fixed- and variable-length objects, because they are allocated out of the same pool of storage.

Variable-length objects are allocated in two different ways, reflected in the items "variable" and "chunked." The distribution of the former among several different sized free lists is shown next.

## Storage Leak Tracking Functions

The functions in GCHax are oriented toward finding leaks that involve items of some data type not getting garbage collected.

There are two main kinds of leaks:

- Items that are unintentionally being held onto
- Items that no user structure is pointing to but are not collected because of the nature of the garbage collector

Examples of the former are structures assigned to global variables and left there after the program finishes.

Examples of the latter are principally circular structures — structures where you can follow a chain of pointers from an object that eventually returns to the same object. Circular lists, such as you get from (NCONC A A), are a special case of circular structures. See comments in the Limitations section below.

Note: All functions listed below have names beginning with \ to remind you that you are dealing with system internals, and to proceed with at least a little caution. Although these functions are generally safe, in that their casual use will not cause arbitrary damage, you certainly can produce unintended side effects.

In particular, the functions \SHOWGC and \COLLECTINUSE have modes in which they return a list of some kind of pointer; beware of unintentionally holding on to such a list (e.g., by having it get onto the history list), thereby preventing the eventual garbage collection of any of those pointers.

Useful for keeping values off the history list are the Executive command SHH for completely inhibiting history list entry, and the idiom (PROG1 NIL operation), e.g., (PROG1 NIL (INSPECT value)) to inspect a structure without holding on to a pointer to the inspect window. You may find it convenient to define your own Exec command to do inspection, e.g.,

```
(DEFCOMMAND IN (OBJ TYPE)
 (PROG1 NIL (INSPECT OBJ TYPE)))
```

The reference counts of all objects in the system are maintained in a global hash table, called the GC reference count table. Some or all of its contents can be viewed with the following function:

(\SHOWGC ONLYTYPES COLLECT FILE CARLVL CDRLVL MINCNT) [Function]

Displays on *FILE* (default T) all objects in the GC reference count table whose reference count is at least *MINCNT*, whose default value is 2.

If *ONLYTYPES* is given, it is a list of data type names to which \SHOWGC confines itself.

If *COLLECT* is T, \SHOWGC returns a list of all the objects it displays.

*CARLVL* and *CDRLVL* are print levels affecting the displaying of lists; they default to two and six, respectively. In the listing, collision entries in the reference count table are tagged with a \*. Reference count operations on pointers in collision entries are much slower than on noncollision entries.

Objects with reference count of one (1) do not appear explicitly in the reference count table, so cannot be viewed with `\SHOWGC`, even if you set *MINCNT* as low as 1.

Note that if *COLLECT* is *T*, then the reference count of all the collected items is now one greater, due to the pointer to each from the returned list.

`(\REFCNT PTR)` [Function]

Returns the current reference count of *PTR*. Pointers that are not reference counted (e.g., symbols and small integers) are considered to have reference count 1. Since pointers from the stack (e.g., *PROG* variables) do not affect reference counts, it is possible for the reference count of an object to be zero without the object being garbage collected.

Note: If you call `\REFCNT` from the Common Lisp interpreter, e.g., by typing it at top-level, the answer is almost always too large by 1, as the interpreter itself holds reference-counted pointers to the arguments to the function it is calling. The same problem besets `\FINDPOINTER` (below). The problem does not exist from the Old Interlisp Exec, which uses the Interlisp interpreter. You can also avoid the problem by explicitly invoking the Interlisp interpreter; e.g.,

`(EVAL '(\REFCNT expression)).`

`(\#COLLISIONS)` [Function]

Returns a list of four elements:

- Number of entries in the reference-count table, i.e., the number of objects in memory whose reference count is not 1
- Number of entries that are in collision chains
- Ratio of these numbers, i.e., the fraction of all entries that are in collision chains
- Ratio of the number of entries to the size of the hash table

`(\#OVERFLOWS)` [Function]

Returns a list of four elements like `\#COLLISIONS`, but instead counts only objects whose reference count has overflowed (is greater than 62). Reference count operations on such objects are significantly slower than on other objects.

`(\COLLECTINUSE TYPE PRED)` [Function]

Is useful when (*STORAGE TYPE*) shows more objects in use than you think is right, but you can't find any such pointers yourself.

*TYPE* is a data type name or number other than *LISTP*. `\COLLECTINUSE` returns a list of all objects of that type that are thought to be in use, i.e., not free.

If *PRED* is supplied, it is a function of one argument. `\COLLECTINUSE` returns only objects for which *PRED* returns true. *PRED* must not allocate storage; you probably want it to be a compiled function.

Note: `\COLLECTINUSE` should be used with care. In a correctly functioning system, `\COLLECTINUSE` is generally safe. However, if the free list of *TYPE* has been smashed so that some free objects are not on it, this function can make matters much more confused, especially if the first 32-bit field of the data type in question contains a pointer field.

(`\FINDPOINTER PTR COLLECT/INSPECT? ALLFLG MARGIN ALLBACKFLG`) [Function]

Provides a brute-force approach to answering the question, "Who has a pointer to *x*?" `\FINDPOINTER` searches virtual memory, looking for places where *PTR* is stored. The search is not completely blind: unless *ALLFLG* is true, it does not look in places that cannot have reference-counted pointers, such as *pname* space or the stack. However, if the reference count of the object is zero, `\FINDPOINTER` searches the stack (and only the stack, if *ALLFLG* is *NIL*), since in this case there is no hope of finding pointers in the usual reference-counted spaces. If *ALLFLG* = *:STACK*, then `\FINDPOINTER` searches the stack in addition to places that contain reference-counted pointers, but not other unlikely places.

`\FINDPOINTER` prints out a description of each place *PTR* is found. If it is found in a list, it asks whether to recursively search for pointers to the list, so you can track lists back to a more identifying place, such as a symbol value cell or some data type. It recurses without asking if *ALLBACKFLG* is true. If *PTR* is found in a typed object, `\FINDPOINTER` names the field, if the data type declaration is available, and asks if you want to recursively search for pointers to this object. In either case, the search stops once enough places have been found to account for *PTR*'s reference count (unless *ALLFLG* is *T*).

If *COLLECT/INSPECT?* is true, `\FINDPOINTER` saves the identifiable pointers in a list. If *COLLECT/INSPECT?* = *COLLECT*, the list of pointers is returned as value; otherwise, it is offered for inspection.

*MARGIN* is the left margin (in units of characters) by which the reports of locations are initially indented. The default is zero. Recursive searches for pointers are indented relative to this position.

The current version does not know how to parse array space, so if *PTR* is found in an array, the best it can do is print the memory address where it found it, usually something of the form `{ }#nn,nnnnn`. In addition, `\FINDPOINTER` doesn't even try to find *PTR* as a literal inside a compiled code object, since such references are not cell-aligned. Thus, `\FINDPOINTER` is really most helpful if the pointer is stored in fixed-length data space (e.g., in a field of a data type, or as the top-level value of a symbol); fortunately, this handles most of the interesting cases in practice.

Note: Of course, since it touches (potentially) a huge percentage of your virtual memory, `\FINDPOINTER` is completely disruptive of your working set.

(`\FINDPOINTERS.OF.TYPE TYPE FILTER`) [Function]

Calls `\FINDPOINTER` on each pointer in use of type *TYPE* that satisfies *FILTER*, a function of one argument, the pointer. A *FILTER* of *NIL* is

considered the true predicate. *FILTER* can also be a list form to evaluate in which the variable *PTR* is used to refer to the pointer in question.

`\FINDPOINTERS.OF.TYPE` is essentially the same as

```
(for PTR in (\COLLECTINUSE TYPE)
 when <FILTER is satisfied>
 do (\FINDPOINTER PTR))
```

except that it takes care to discard the cells of the list returned from `\COLLECTINUSE` before calling `\FINDPOINTER`, to avoid seeing one extra reference per object.

For example,

```
(\FINDPOINTERS.OF.TYPE 'STREAM' (NOT (OPENP PTR)))
```

searches for pointers to all streams that are not currently open.

```
(\SHOW.CLOSED.WINDOWS)
```

[Function]

Collects all windows that are not currently open or icons of open windows, then opens each window one by one.

For each window, you are prompted to press the left mouse button to close the window and go on to the next, or press right to do something different. In the latter case, you are prompted again to press the left button if you would like to search for pointers to the window, using `\FINDPOINTER`, or press the right button to just leave the window open on the screen and proceed.

Returns the total number of windows examined.

```
(\SHOWCIRCULARITY OBJECT MAXLEVEL)
```

[Function]

Follows pointers from *OBJECT*. If it finds a path back to itself, it prints that path. This function is not exceptionally fast, and deliberately (for performance reasons) does not detect circularities in lists; it simply bottoms out on lists at *MAXLEVEL*, which defaults to 1,000. Circular lists are usually obvious enough anyway.

```
(\MAPGC MAPFN INCLUDEZEROCNT)
```

[Function]

Maps over all entries in the GC reference count table, applying *MAPFN* to three arguments: the pointer, its reference count (an integer), and *COLLISIONP*, a flag that is T if the entry is a collision entry. Entries with reference count zero are not included unless *INCLUDEZEROCNT* is T. This function underlies `\SHOWGC`. Some care is required in the writing of *MAPFN*; it should try to minimize any reference-counting activity of its own, and in particular avoid anything that would decrement the reference count of the pointer passed to it.

## Limitations

GCHax is not very useful for finding ordinary circular lists, as the typical system has vast amounts of list structure, with nothing to distinguish the interesting ones.

However, if the circular list also contains instances of user data types, then those data types will tend to show up as overallocated, and hence amenable to the search functions in this module.

`\FINDPOINTER` does not know how to locate pointer arrays of more than 64 elements, so it is not helpful if a pointer you seek is located only in such an array.

[This page intentionally left blank]

# GRAPHER

---

Grapher contains a collection of functions and an interface for laying out, displaying, and editing graphs, that is, networks of nodes connected by links. Graphs have node labels but not link labels. Links are drawn by default as straight lines without arrowheads, but you can control the appearance of individual links. Node labels can be single lines of text, bitmaps of arbitrary size, or image objects. Facilities exist for calling functions at the nodes in a graph, and image objects containing graphs can be constructed so you can include graphs in documents and other image structures.

For instance, the Browser module uses graphs to represent function-calling structures (from MasterScope). Such a partially specified node list need have only the graph labels and the links specified. It is given to the `LAYOUTGRAPH` function along with some formatting information. `LAYOUTGRAPH` is a Grapher function which assigns a position to each node. There are formats for laying out trees, lattices, and cyclic graphs. `LAYOUTGRAPH` returns an instance of the `GRAPH` record, which is usually given to the function `SHOWGRAPH`. `SHOWGRAPH` displays a graph in a window.

## Installation

---

Load `GRAPHER.LCOM` from the library.

## User Interface

---

A typical way to use Grapher is to implement a function that creates a partially specified list of graph nodes representing some user data (or control) structure. Then you can use Grapher to display and manipulate or explore that structure.

Displayed graphs can be edited using the right button on the mouse. Nodes can be added, deleted, moved, enlarged, or shrunk. Links can be added or deleted.

Displayed graphs are often used as menus: selecting a node with the left or middle button can cause user-provided functions to be called on that node.

## Functions

---

Grapher functions perform the following tasks:

- Creating a graph
- Laying out a graph for display
- Displaying a graph
- Editing a graph
- Inserting a graph into a document

These tasks are described in the following subsections. An additional subsection describes Grapher functions that perform other tasks.



## Creating a Graph

Start by creating a list of nodes for the graph. You can create them directly (see the GRAPHNODE Record section), or you can use the NODECREATE function.

(NODECREATE *ID LABEL POSITION TONODEIDS FROMNODEIDS*  
*FONT BORDER LABELSHADE*) [Function]

This function returns a GRAPHNODE record. The arguments of this function are the same as the corresponding fields of the GRAPHNODE record, as follows:

| Argument           | GRAPHNODE Field |
|--------------------|-----------------|
| <i>ID</i>          | NODEID          |
| <i>LABEL</i>       | NODELABEL       |
| <i>POSITION</i>    | NODEPOSITION    |
| <i>TONODEIDS</i>   | TONODE          |
| <i>FROMNODEIDS</i> | FROMNODE        |
| <i>FONT</i>        | NODEFONT        |
| <i>BORDER</i>      | NODEBORDER      |
| <i>LABELSHADE</i>  | NODELABELSHADE  |

*ID* and *LABEL* are required; *BORDER* defaults to 0, *LABELSHADE* defaults to WHITESHADE, and *POSITION* defaults to wherever it seems convenient.

You need to specify how the nodes are connected by providing a list of *TONODEIDS* and *FROMNODEIDS* for each node.

## Laying Out a Graph for Display

(LAYOUTGRAPH *NODELST ROOTIDS FORMAT FONT*  
*MOTHERD PERSONALD FAMILYD*) [Function]

Lays out a partially specified graph by assigning positions to its graph nodes. It returns a GRAPH record suitable for displaying with SHOWGRAPH. An example appears after the description of the arguments.

- NODELST* Is a list of partially specified GRAPHNODEs: only their NODELABEL, NODEID, and TONODE fields need to be filled in. NODEFONT fields may also contain font specifications to be used instead of the default supplied by the *FONT* argument. These optional fields are filled in appropriately if they are NIL. All other fields are ignored and/or overwritten.
- ROOTIDS* Is a list of the node identifiers of the nodes that become the roots.
- The rest of the arguments are optional and control the format of the layout.
- FORMAT* Controls the layout of the graph. It is an unordered list of atoms or lists. The following options control the structure of the graph:
- COMPACT, the default, which lays out the graph as a forest (that is, a set of disjoint trees) using the minimal amount of screen space.

- FAST, which lays out the graph as a forest, sacrificing screen space for speed.
- LATTICE, which lays out the graph as a directed acyclic graph, that is, a lattice.

In addition, the following options control the direction of the graph:

- HORIZONTAL, the default, has roots at the left and links that run left-to-right.
- VERTICAL has roots at the top and links that run top-to-bottom.

The directions can be reversed by including the atom REVERSE in *FORMAT*.

For example:

- *FORMAT*=(LATTICE HORIZONTAL REVERSE) lays out horizontal lattices that have the roots on the right, with the links running right-to-left.
- *FORMAT*=(VERTICAL REVERSE) lays out vertical trees that have the roots at the bottom, with links running bottom-to-top.
- *FORMAT*=NIL lays out horizontal trees that have the roots on the left.

LAYOUTGRAPH creates virtual graph nodes to avoid drawing a tangle of messy lines in cases where the graph is not a forest or a lattice to begin with. It modifies the nodes of NODELST, which may involve changing some of the TONODES fields to point to new nodes. The modified NODELST is set into the GRAPHNODES field of a newly created GRAPH record, which is returned as the value of LAYOUTGRAPH. The creation of virtual nodes depends on whether LATTICE is a member of *FORMAT*.

In a forest, nodes are laid out by traversing the forest top-down, depth-first. If a node already has been laid out, LAYOUTGRAPH creates a copy of the node (the same NODELABEL, different NODEID, and no TONODES), lays it down, and marks both it and the original node by setting their NODEBORDER fields and NODELABELSHADE fields. This occurs instead of drawing a link that might cut across arbitrary parts of the graph. Hence, a marked node occurs at least twice in the forest.

The default for marked nodes is to leave the shade alone and set the border to 1. To alter this appearance, add the (MARK . PROPS) to the *FORMAT* argument. PROPS is a property list. If it is NIL, marking is suppressed altogether. If it contains BORDER or LABELSHADE properties, those values are used in the corresponding fields of marked nodes. For example, a format of (MARK BORDER 5) would cause duplicated nodes to be boxed with borders 5 points wide.

*FORMAT* adds a few enhancements to this basic marking strategy, and can include one or both of these atoms:

- *COPIES/ONLY*—Only the new virtual nodes are marked. The original is left unmarked.
- *NOT/LEAVES*—Marking is suppressed when the node has no daughters.

For example:

- *FORMAT*=(*COPIES/ONLY* *NOT/LEAVES*) marks nodes that are copies of nodes that have daughters (for example, if you see a mark, the node has daughters that are not drawn).
- *FORMAT*=(*NOT/LEAVES*) marks both copies and originals, but only when they have daughters.
- *FORMAT*=*NIL* marks originals and copies regardless of progeny.

If *FORMAT* includes *LATTICE*, then a node that is the daughter of more than one node is not marked. Instead, links from all its parents are drawn to it. No attempt is made to avoid drawing lines through nodes or to minimize line crossings. However, in *HORIZONTAL* format, nodes are positioned so that From is always left of To.

Similar conventions hold for the other formats. In *VERTICAL* format, for instance, the *TONODES* of a node are positioned beneath it, and the *FROMNODES* are positioned above it.

Cyclic graphs cannot be drawn using this convention, since a node cannot be left of itself. When *LAYOUTGRAPH* detects a node that points to itself, directly or indirectly, it creates a virtual node, as described above, and marks both the original and the copy. If *FORMAT* includes *COPIES/ONLY*, then only the newly created node is marked.

*FONT* Is a font specification for use as the default *NODEFONT*.

The remaining three arguments control the distances between nodes. *NILs* cause “pretty” defaults based on the size of *FONT*.

*PERSONALD* Is specified in points; it controls the minimum distance between any two nodes.

*MOTHERD* Is the minimum distance between a mother and her daughters.

*FAMILYD* Controls the minimum distance between nodes from different nuclear families. The closest two sister nodes can be is *PERSONALD*. The closest that two nodes that are not sisters can be is *PERSONALD+FAMILYD*.

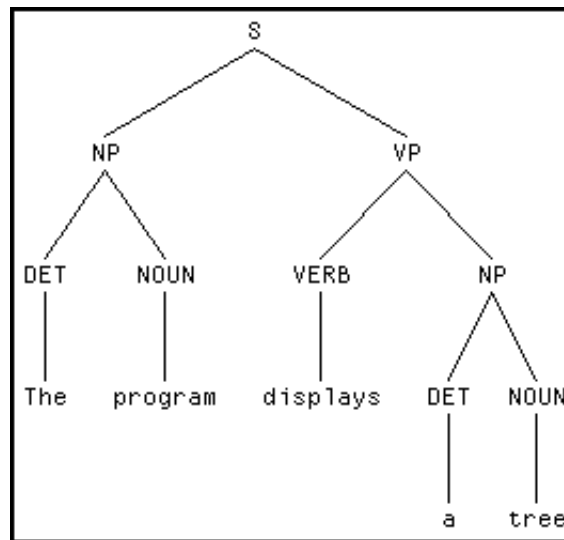
*LAYOUTGRAPH* reads but does not change the fields *NODEBORDER* and *NODELABELSHADE* of the nodes given it. The marked nodes are an exception. Thus, if you plan to install black borders around the nodes after the nodes have been laid out (for example, by *RESET/NODE/BORDER*, described in the Performing Other Tasks section), it is a good idea to give *LAYOUTGRAPH* nodes that have white borders. This causes the nodes to be laid

out far enough apart that when you blacken the borders later, the labels of adjacent nodes are not overwritten.

As an example, to create and display the following parse tree for the sentence "The program displays a tree.", enter the following:

```
(SETQ Snode (NODECREATE 'S 'S NIL '(NP1 VP)))
(SETQ NP1node (NODECREATE 'NP1 'NP NIL '(DET1 NOUN1) '(S)))
(SETQ DET1node (NODECREATE 'DET1 'DET NIL '(THE) '(NP1)))
(SETQ THENode (NODECREATE 'THE 'The NIL NIL '(DET1)))
(SETQ NOUN1node (NODECREATE 'NOUN1 'NOUN NIL '(PROGRAM) '(NP1)))
(SETQ PROGRAMnode (NODECREATE 'PROGRAM 'program NIL NIL '(NOUN1)))
(SETQ VPnode (NODECREATE 'VP 'VP NIL '(VERB NP2) '(S)))
(SETQ VERBnode (NODECREATE 'VERB 'VERB NIL '(DISPLAYS) '(VP)))
(SETQ DISPLAYSnode (NODECREATE 'DISPLAYS 'displays NIL NIL '(VERB)))
(SETQ NP2node (NODECREATE 'NP2 'NP NIL '(DET2 NOUN2) '(VP)))
(SETQ DET2node (NODECREATE 'DET2 'DET NIL '(A) '(NP2)))
(SETQ Anode (NODECREATE 'A 'a NIL NIL '(DET2)))
(SETQ NOUN2node (NODECREATE 'NOUN2 'NOUN NIL '(TREE) '(NP2)))
(SETQ TREENode (NODECREATE 'TREE 'tree NIL NIL '(NOUN2)))

(SHOWGRAPH (LAYOUTGRAPH (LIST Snode NP1node DET1node THENode NOUN1node
PROGRAMnode VPnode VERBnode DISPLAYSnode NP2node DET2node Anode NOUN2node
TREENode) '(S) '(VERTICAL)))
```



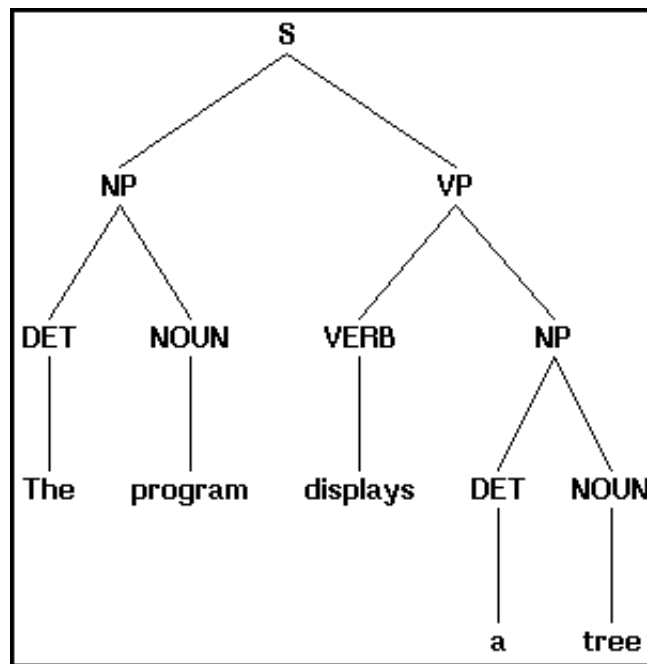
(LAYOUTSEXPR *SEXPR* *FORMAT* *BOXING* *FONT* *MOTHERD* *PERSONALD* *FAMILYD*) [Function]

Is just like LAYOUTGRAPH, except it gets its graph as an s-expression rather than a list of GRAPHNODEs. Its first argument is recursively interpreted as follows: If the s-expression is a non-list, its NODELABEL is itself and it has no TONODEs; else its CAR is taken as its NODELABEL and its CDR, which must be a list of s-expressions, is taken as its TONODEs.

Note: Circular s-expressions are allowed.

For example, to display the following parse tree for the sentence "The program displays a tree.", enter:

```
[SHOWGRAPH (LAYOUTSEXPR '(S (NP (DET The) (NOUN program)) (VP
(VERB displays) (NP (DET a) (NOUN tree)))) '(VERTICAL) NIL
'(HELVETICA 12 BRR)]
```



## Displaying a Graph

(SHOWGRAPH GRAPH W LEFTBUTTONFN MIDDLEBUTTONFN TOPJUSTIFYFLG  
ALLOWEDITFLG COPYBUTTONEVENTFN) [Function]

Displays the nodes in *GRAPH*.

If *W* is a window, the graph is displayed in it. If the graph is larger than the window, the window is made a scrolling window. If *W* is *NIL*, the graph is displayed in a window large enough to hold it. If *W* is a string, the graph is displayed in a window large enough to hold it, and the window uses the string for the window title. The graph is stored on the *GRAPH* property of the window. *SHOWGRAPH* returns the window.

If either *LEFTBUTTONFN* or *MIDDLEBUTTONFN* is non-*NIL*, the window is given a *BUTTONEVENTFN* that, in effect, turns the graph into a menu. Whenever you press left or middle mouse button and the cursor is over a node, that node is displayed inverted, indicating that it is selected. Releasing the mouse button calls either the *LEFTBUTTONFN* or the *MIDDLEBUTTONFN* with two arguments: the selected node and the window. The node is a *GRAPHNODE*, or *NIL* if the cursor was not over a node when the button was released. The function can access the graph via the window's *GRAPH* property.

The graph's initial position in the window is determined by *TOPJUSTIFYFLG*. If *T*, the graph's top edge is positioned at the top edge of the window; if *NIL*, the graph's bottom edge is positioned at the bottom edge of the window.

*ALLOWEDITFLG* and *COPYBOTTONFLG* are described under "Editing a Graph," below.

Note: The node labels are reprinted whenever the graph is redisplayed. If this makes scrolling of a large graph unacceptably slow, some speedup may be achieved by instructing Grapher to cache bitmaps of the labels with the nodes so they can be rapidly *BITBLT*ed to the screen (set the variable *CACHE/NODE/LABEL/BITMAPS* to *T*). The possible gain in time,

however, may be offset by the increased storage required for the cached bitmaps.

(DISPLAYGRAPH *GRAPH STREAM CLIP/REG TRANS*) [Function]

Displays the specified graph on *STREAM*, which can be any image stream, with coordinates translated to *TRANS*. Some streams might also implement *CLIP/REG* as a clipping region. This is primarily to improve efficiency for the display.

(HARDCOPYGRAPH *GRAPH/WINDOW FILE IMAGETYPE TRANS*) [Function]

Produces a file containing an image of *GRAPH*, that is, like SHOWGRAPH, but for files. If *GRAPH/WINDOW* is a window, HARDCOPYGRAPH operates on its GRAPH window property. The *FILE* and *IMAGETYPE* argument are given to OPENIMAGESTREAM to obtain a stream on which the graph is displayed. *TRANS* is the position in screen points (that is, it is scaled by the image stream's DSPSCALE) of the lower-left corner of the graph relative to the lower-left corner of the piece of paper.

GRAPH/HARDCOPY/FORMAT [Variable]

Is used to control the format of the graph when printing to paper. It is a property list that contains the following properties.

- MODE

Determines the orientation of the hardcopy of the graph. The value can be LANDSCAPE, or PORTRAIT (the default). If LANDSCAPE, the graph is shown with the longer paper edge as the major axis. If PORTRAIT, graph is shown with the shorter paper edge as the major axis. If you use the window menu command to hardcopy, the graph is shown in PORTRAIT mode.

- PAGENUMBERS

Determines whether to print the page number. The value can be T or NIL. If T, GRAPHER prints the page number in X-Y format on the upper right corner of each page. If NIL, no page number is printed.

- TRANS

Determines where to position the graph on paper. The value can be NIL or a position. If NIL, each graph is positioned at the center of the paper. If a position, GRAPHER determines the location in screen points of the lower left corner of the graph relative to the lower left corner of the paper.

The initial value of GRAPH/HARDCOPY/FORMAT is set to  
(MODE PORTRAIT PAGENUMBERS T TRANS NIL)

DEFAULT.GRAPH.WINDOWSIZE [Variable]

Contains a list of two numbers in screen points. The first number indicates the window width. The second number indicates the window height. This variable is used to control the maximum size of a graph window when it first gets displayed.

## Editing a Graph

If `ALLOWEDITFLG` in the `SHOWGRAPH` function is non-NIL, you can position the cursor over a node and use the right mouse button to edit the graph. (The normal window commands can be accessed by right-clicking (pressing the right mouse button) in the border or title regions.) Holding down the `CONTROL` key and simultaneously pressing the right mouse button allows you to position nodes by tracking the cursor. Pressing the right mouse button without the control key pops up the following menu of edit operations.

```

Move Node
Add Node
Delete Node
Add Link
Delete Link
Change label
label smaller
label larger
<-> Directed
<-> Sides
<-> Border
<-> Shade
STOP

```

The edit operations allow moving, adding, and deleting of nodes and links.

- Adding a node prompts for a `NODELABEL` creates a new node with that label, adds it to the graph, and allows you to position it.
- Deleting a node removes it (using `DREMOVE`) from the graph after deleting all of the links to and from it.
- Selecting `Directed` or `Sides` allows you to control how links are drawn between nodes. See the section "Graph Record" for more information.
- Selecting `Border` allows you to invert the border around a node's label.
- Selecting `Shades` allows you to invert a node's label.

When you select the `STOP` menu command, the graph window is closed.

`COPYBUTTONEVENTFN` is a function to be run when you copy-select from the Grapher window. If this is not specified, the default simply `COPYINSERTs` a Grapher image object.

Certain fields of the `GRAPH` record contain functions that are called from the graph editor menu to perform actions on an element in the displayed graph. They allow the graph to serve as a simple edit interface to the structure being graphed.

The following fields of a graph contain editing functions and the arguments that are passed to those functions when they are called. In all cases, `GRAPH` is the graph being displayed, and `WINDOW` is the window in which it is displayed.

Even if you do not specify any of the following fields (except `GRAPH.ADDNODEFN`), the system provides a set of functions which allows you to edit the graph. Any functions you specify are executed *after* the default function is executed. For example, if you supply the `ADDLINKFN` and then add a link using the edit menu, your function is called after the link is added. Exception: for `ADDNODEFN`, the function is called and must return a node to be added to the graph; this function is executed *instead* of the default function.



GRAPH.MOVENODEFN *NODE NEWPOS GRAPH WINDOW OLDPOS* [Record field]

Contains a function that is called with the arguments shown after you have stopped moving a node interactively; that is, it is not called as the node is being moved. *NEWPOS* is the new position of the node, *OLDPOS* its original position. The difference between them can be used, for example, to move other related nodes by the same distance.

GRAPH.ADDNODEFN *GRAPH WINDOW* [Record field]

Is called when you select ADD A NODE. Returns a node, or NIL if no new node is to be added. A node-moving operation is called on the new node after it is created to determine its position.

GRAPH.DELETENODEFN *NODE GRAPH WINDOW* [Record field]

Is called when a node is deleted. Before this function is called, all of the links to or from the node are deleted.

GRAPH.ADDLINKFN *FROM TO GRAPH WINDOW* [Record field]

Is called when a link is added.

GRAPH.DELETELINKFN *FROM TO GRAPH WINDOW* [Record field]

Is called when a link is deleted, which can be either directly or from deleting a node.

GRAPH.FONTCHANGEFN *HOW NODE GRAPH WINDOW* [Record field]

Is called for side effect only when you ask for the label on a node to be made larger or smaller. *HOW* is either LARGER or SMALLER.

GRAPH.CHANGELABELFN *GRAPH NODE* [Record field]

Is called for side effect only when you ask to change the label of a node, for example, using EDITGRAPH.

GRAPH.INVERTBORDERFN *NODE GRAPH* [Record field]

Is called for side effect only when you ask to invert the border of a node, for example, using EDITGRAPH.

GRAPH.INVERTLABELFN *NODE GRAPH* [Record field]

Is called for side effect only when you ask to invert the label of a node, for example, using EDITGRAPH.

## Editing Menu

The editing menu is controlled by the following two variables:

EDITGRAPHMENU [Variable]

Contains the editing menu, if it exists, or NIL. If you press the right button and it is NIL, a fresh menu is created from EDITGRAPHMENUCOMMANDS.

EDITGRAPHMENUCOMMANDS

[Variable]

A list of menu items used to create EDITGRAPHMENU. The contents of EDITGRAPHMENUCOMMANDS must be a list that can be used as the ITEMS field of a MENU; see MENU in the *Interlisp-D Reference Manual* for details.

## Inserting a Graph into a TEdit Document

A graph data structure can be encapsulated in a Grapher image object so that it can be inserted in a TEdit document or other image structure. Grapher image objects are constructed by the following function.

(GRAPHEROBJ GRAPH HALIGN VALIGN)

[Function]

Returns a Grapher-type image object that displays GRAPH.

HALIGN and VALIGN specify how the graph is to be aligned with respect to the reference point in its host, for example, a TEdit file or image object window. They can be numbers between zero and one, specifying as a proportion of the width/height of the graph the point in the graph that overlays the reference point; zero means that the graph sits completely above and to the left of the reference point, and one means it sits completely below and to the right.

They can also be pairs of the form (NODESPEC POS), where

NODESPEC specifies a node that the graph is to be aligned by, and POS specifies where in the node the alignment point is. The NODESPEC can be either a NODEID or one of the atoms \*TOP\*, \*BOTTOM\*, \*LEFT\*, or \*RIGHT\*, indicating the topmost, bottommost, etc., node of the graph.

POS can be a number specifying proportional distances from the lower-left corner of the node, or the atom BASELINE, indicating the character baseline (for VALIGN, or simply zero for HALIGN).

For example, to align a linguistic tree so that the baseline of the root node is at the reference point, VALIGN is (\*TOP\* BASELINE).

The BUTTONEVENTINFN of the image object pops up a single-item menu, which, if selected, causes the graph editor to be run.

## Performing Other Tasks

Grapher functions also allow you to return the smallest region containing all nodes, invert a node region, reset fields in a node, print a graph to a stream, read a graph from a stream, and edit a graph.

(GRAPHREGION GRAPH)

[Function]

Returns the smallest region containing all of the nodes in GRAPH.

(FLIPNODE NODE DS)

[Function]

Inverts a region in the stream DS that is one pixel bigger all around than NODE's region. This makes it possible to see black borders after the node has been flipped.

(RESET/NODE/BORDER NODE BORDER STREAM GRAPH)

[Function]

and

(RESET/NODE/LABELSHADE NODE SHADE STREAM)

[Function]

Reset the appropriate fields in the node. If *STREAM* is a display stream or a window, the old node is erased and the new node is displayed. Changing the border may change the size of the node, in which case the lines to and from the node are redrawn. The entire graph must be available to *RESET/NODE/BORDER* for this purpose, either supplied as the *GRAPH* argument or obtained from the *GRAPH* property of *STREAM*, if it is a window. Both functions take the atom *INVERT* as a special value for *BORDER* and *SHADE*. They read the node's current border or shade, calculate what is needed to invert it, and do so.

(DUMPGRAPH *GRAPH STREAM*) [Function]

Prints *GRAPH* out on *STREAM* in a special, relatively compact encoding that can be interpreted by the function *READGRAPH*, below. Graphs cannot be saved on files simply by ordinary print functions such as *PRIN2*. This is because the Grapher functions use *FASSOC* (that is, *EQ*, not *EQUAL*) to fetch a graph node given its ID, so reading it back in gives the right result only if the IDs are atomic. *HPRINT* resolves this problem, but it tends to dump too much information: it dumps a complete description of the node font, for example, including the character bit maps. *DUMPGRAPH* and *READGRAPH* are used in the implementation of Grapher image objects.

(READGRAPH *STREAM*) [Function]

Reads information from *STREAM* starting at the current file pointer and returns a graph structure equivalent to the one that was given to *DUMPGRAPH*.

(EDITGRAPH *GRAPH WINDOW*) [Function]

Enables editing *GRAPH* in *WINDOW*. If *GRAPH* is *NIL*, an empty graph is created for editing. If *WINDOW* is *NIL*, a window of appropriate size is created.

(GRAPHERPROP *GRAPH PROP NEWVALUE*) [Function]

Accesses *GRAPH.PROPS* field of *GRAPH* record. The function returns the previous value of *GRAPH*'s *PROP* aspect. If *NEWVALUE* is given, it is stored as the new *PROP* aspect.

---

## Grapher Record Structure

Grapher has *GRAPH* records which represent graphs. Within these records are *GRAPHNODE* records which are lists of graph records.

### GRAPH Record

A graph is represented by a *GRAPH* record, which has the following fields:

- GRAPHNODE
- DIRECTEDFLG
- SIDESFLG
- GRAPH.MOVENODEFN
- GRAPH.ADDNODEFN
- GRAPH.DELETENODEFN
- GRAPH.ADDLINKFN
- GRAPH.DELETELINKFN

```

GRAPH.CHANGELABELFN
GRAPH.INVERTBORDERFN
GRAPH.INVERTLABELFN
GRAPH.FONTCHANGEFN
GRAPH.PROPS

```

GRAPHNODE is a list of graph nodes, and is described below.

DIRECTEDFLG and SIDESFLG are flags that control how links are drawn between the nodes. If DIRECTEDFLG is NIL, Grapher draws each link in such a way that it does not cross the node labels of the nodes it runs between. Often, this leaves some ambiguity, which is settled by SIDESFLG. If SIDESFLG is NIL, Grapher prefers to draw links that go between the top and bottom edges of nodes. If SIDESFLG is non-NIL, Grapher prefers to draw links between the sides of the nodes.

If DIRECTEDFLG is non-NIL, the edges are fixed, for example, always to the left edge of the To node. This can cause links to cross the labels of the nodes they run between. In this case, if SIDESFLG is NIL, the From end of the link is attached to the bottom edge of the From node; the To end of the link is attached to the top edge of the To node. If DIRECTEDFLG is non-NIL and SIDESFLG is non-NIL, the From end of the link is attached to the right edge of the From node; the To end of the link is attached to the left edge of the To node.

GRAPH.PROPS is a list in property-list format, and is accessed by the function GRAPHERPROP.

The remaining fields give you hooks into the graph editor, and are described in the section "Editing a Graph".

## GRAPHNODE Record

The GRAPHNODE record has the following fields of interest:

|           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NODELABEL | Is what gets displayed as the node. If this is a bit map, BITBLT is used; if it is an image object, its <i>IMAGEBOXFN</i> and <i>DISPLAYFN</i> are used. Anything else is printed with PRIN3; see the <i>Interlisp-D Reference Manual</i> . Image objects can be used to give a node a larger-than-normal margin around its text label.                                                                                                                                         |
| NODEID    | Is a unique identifier. NODEIDs are used in the link fields instead of pointers to the nodes themselves, so that circular Lisp structures can be avoided. NODEIDs are often used as pointers to the structure represented by the graph.                                                                                                                                                                                                                                         |
| TONODE    | Is a list of NODEIDs. A link runs from the currently selected node to each node in TONODEs. Entries in this field can be used to specify properties of the lines drawn between nodes.<br><br>If an item in the TONODEs of the current node N1 is not a NODEID but rather a list of the form:<br><br>(LINK% PARAMETERS TONODEID . PARAMLIST)<br><br>then <i>PARAMLIST</i> is interpreted as a property list specifying properties of the link drawn from N1 to <i>TONODEID</i> . |

Properties of *PARAMLIST* currently noticed are *LINEWIDTH*, *DASHING*, *COLOR*, and *DRAWLINKFN*. The first three are passed directly to *DRAWLINE*.

For example, if the *TONODES* for A is:

```
((LINK% PARAMETERS B LINEWIDTH 4 DASHING (3 3))
(LINK% PARAMETERS C DASHING (5 1) COLOR 12))
```

then two dashed lines emanate from A, with the one to B having width 4 and dashing (3 3), and the one to C having the default width 1, dashing (5 1), and color (if implemented) 12.

If the property *DRAWLINKFN* is on the list, then its value must be a function to be called instead of *DRAWLINE*. It is passed all the arguments of *DRAWLINE* plus the *PARAMLIST* as a last argument.

For convenience, the variable *LINKPARAMS* is set to the constant value *LINK% PARAMETERS*. When *DISPLAYGRAPH* scales the graph to the units of a particular output stream, the properties whose names are found on *SPECVAR SCALABLELINKPARAMETERS* are also scaled.

|                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FROMNODE       | Is a list of <i>NODEIDS</i> . A link runs to the currently selected node from each node in <i>FROMNODES</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| NODEPOSITION   | Is the location of the center of the node (a <i>POSITION</i> ).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| NODEFONT       | Specifies the font in which this node's label is displayed. It can be any font specification acceptable to <i>FONTCREATE</i> , including a <i>FONTDESCRIPTOR</i> . <i>NODEFONT</i> is changed by the graph edit operations <i>Larger</i> and <i>Smaller</i> . When this happens, the font family may be changed as well as the size. Default is the value of <i>DEFAULT.GRAPH.NODEFONT</i> (initially <i>NIL</i> , which specifies the system <i>DEFAULTFONT</i> ).                                                                                 |
| NODEBORDER     | Specifies the shade and width of the border around a node via the following values:<br>NIL, 0     No border; equals border of width zero<br>T           Black border, one pixel wide<br>1, 2, 3... Black border of the given width<br>-1, -2... White border of the given width<br>(W S)     Where W is an integer and S is a texture or a shade; yields a border W pixels wide filled with the given shade S; see the <i>Interlisp-D Referene Manual</i> .<br><br>Default is the value of <i>DEFAULT.GRAPH.NODEBORDER</i> (initially <i>NIL</i> ). |
| NODELABELSHADE | Contains the background shade of the node. If this field is non-NIL, then when a node is displayed, the label area for the node is first painted as specified by this field, then the                                                                                                                                                                                                                                                                                                                                                               |

label is printed in `INVERT` mode. This does not apply to labels that are bit maps or image objects. The legal values for the field are: `NIL` (same as `WHITESHADE`), `T` (same as `BLACKSHADE`), a texture, or a bitmap. Default is the value of `DEFAULT.GRAPH.NODELABELSHADE`, which is initially `NIL`.

`NODEWIDTH` and `NODEHEIGHT` Are initially set by Grapher to be the width and height of the node's `NODELABEL`.

## Limitations

---

Grapher does not work well with packages. Because the node labels are printed with `PRIN3`, Grapher does not visually distinguish nodes whose labels are symbols in different packages; you do not see that fact displayed in the graph.

# GRAPHZOOM

---

GraphZoom allows you to work with graphs (from Grapher) at different scales.

## Requirements

---

GRAPHER

## Installation

---

Load `GRAPHER.LCOM` and `GRAPHZOOM.LCOM` from the library.

## User Interface

---

The interface to GraphZoom is through the function `SHOWZOOMGRAPH` which is used instead of `SHOWGRAPH`. The resulting grapher window has the capability of being zoomed by using a menu that is attached to the top. To use, call `SHOWZOOMGRAPH`:

The window has a menu above it that contains the items `LARGER` and `SMALLER`. Selecting in the menu with the left button causes a small change in scale; selecting with the middle button causes a larger change in scale.

The graph zooms toward the center of the window.

## Function

---

`(SHOWZOOMGRAPH GRAPH WINDOW LEFTBUTTONFN MIDDLEBUTTONFN  
TOPJUSTIFYFLG ALLOWEDITFLG INITSCALE)` [Function]

The arguments are the same as in the Grapher function `SHOWGRAPH`. `INITSCALE` is the initial scale at which the graph is shown. The default, `1.0`, is the same as in `SHOWGRAPH`. Larger scales make the graph appear smaller.

## Limitations

---

The zooming of text is driven from a list of font descriptors stored as the value of the variable `DECREASING.FONT.LIST`. The values on this list are Times Roman 72, 36, 30, Helvetica 24, 18, 14, 12, 10, 8, 5, 4, and 3. Graphs that have fonts other than Helvetica are printed in the closest Helvetica size. This is often smaller than the corresponding Gacha font.



[This page intentionally left blank]



# HASH

---

Note: This module is provided for backwards compatibility. New applications should use the HASH-FILE Library Module instead of this module.

Hash permits information associated with string or atom keys to be stored on and retrieved from files. The information (or values) associated with the keys in a file may be numbers, strings, or arbitrary Lisp expressions. The associates are maintained by a hashing scheme that minimizes the number of file operations it takes to access a value from its key.

Information is saved in a hash file, which is analogous to a hash array. Actually, a hash file can be either the file itself, or the handle on that file which is used by the Hash module. The latter, of data type HashFile, is the datum returned by CREATEHASHFILE or OPENHASHFILE, currently an array record containing the hash file name, the number of slots in the file, the used slots, and other details. All other functions with hash file arguments use this datum.

In older implementations (e.g., for Interlisp-10), hash files came in several varieties, according to the types of value stored in them. The EMYCIN system provided even more flexibility.

This system only supports the most general EXPR type of hash files and EMYCIN-style TEXT entries, in the same file. The *VALUETYPE* and *ITEMLENGTH* arguments are for the most part ignored. Two-key hashing is supported in this system, but it is discouraged as it is only used in EMYCIN, not in the Interlisp-10 system. The functions GETPAGE, DELPAGE, and GETPNAME, which manipulate secret pages, do not exist in this implementation. However, it is permissible to write data at the end of a hash file. That data will be ignored by the Hash module, and can be used to store additional data.

The Hash module views files as a sequence of bytes, randomly accessible. No notice is made of pages, and it is assumed that the host computer buffers I/O sufficiently.

Hash files consist of a short header section (8 bytes), a layer of pointers ( $4 * \text{HASHFILE:Size}$  bytes), followed by ASCII data. Pointers are 3 bytes wide, preceded by a status byte. The pointers point to key PNAMEs in the data section, where each key is followed by its value.

Deleted key pointers are reused but deleted data space is not, so rehashing is required if many items have been replaced.

The data section starts at  $4 * \text{HASHFILE:Size} + 9$ , and consists of alternating keys and values. As deleted data is not rewritten, not all data in the data section is valid.

When a key hashes into a used slot, a probe value is added to it to find the next slot to search. The probe value is a small prime derived from the original hash key.

---

## Requirements

Hash files must reside on a random-access device (not a TCP/IP file server).

---

## Installation

Load HASH.LCOM from the Library.

## Functions

---

### Creating a Hash File

(CREATEHASHFILE *FILE* *VALUETYPE* *ITEMLENGTH* *#ENTRIES* *SMASH* *COPYFN*) [Function]

Creates a new hash file named *FILE*. All other arguments are optional.

*VALUETYPE* is ignored in this implementation; any hash file can accommodate both Lisp expressions and text.

*ITEMLENGTH* is not used by the system but is currently saved on the file (if less than 256) for future use.

*#ENTRIES* is an estimate of the number of entries the file will have. (This should be a realistic guess.)

*SMASH* is a hash file datum to reuse.

*COPYFN* is a function to be applied to entries when the file is rehashed (see the description of REHASHFILE below).

### Opening and Closing Hash Files

Before you can use a hashfile with this module, you have to open it using the following function.

(OPENHASHFILE *FILE* *ACCESS* *ITEMLENGTH* *#ENTRIES* *SMASH*) [Function]

Reopens the previously existing hash file *FILE*.

Access may be INPUT (or NIL), in which case *FILE* is opened for reading only, or BOTH, in which case *FILE* is open for both input and output. Causes the error "not a hashfile" if *FILE* is not recognized as a hash file.

*ITEMLENGTH* and *#ENTRIES* are for backward compatibility with EMYCIN where OPENHASHFILE also created new hash files; these arguments should be avoided.

*SMASH* is a hash file datum to reuse.

If *ACCESS* is BOTH and *FILE* is a hash file open for reading only, OPENHASHFILE attempts to close it and reopen it for writing. Otherwise, if *FILE* designates an already open hash file, OPENHASHFILE is a no-op.

OPENHASHFILE returns a hash file datum.

(CLOSEHASHFILE *HASHFILE* *REOPEN*) [Function]

Closes HASHFILE (when you are finished using a hash file, you should close it). If *REOPEN* is non-NIL, it should be one of the accepted access types. In this case, the file is closed and then immediately reopened with *ACCESS* = *REOPEN*. This is used to make sure the hash file is valid on the disk.

### Storing and Retrieving Data

(PUTHASHFILE *KEY* *VALUE* *HASHFILE* *KEY2*) [Function]

Puts *VALUE* under *KEY* in *HASHFILE*. If *VALUE* is NIL, any previous entry for *KEY* is deleted. *KEY2* is for EMYCIN two-key hashing; *KEY2* is internally appended to *KEY* and they are treated as a single key.

(GETHASHVILE *KEY HASHFILE KEY2*) [Function]

Gets the value stored under *KEY* in *HASHFILE*. *KEY2* is necessary if it was supplied to PUTHASHFILE.

(LOOKUPHASHFILE *KEY VALUE HASHFILE CALLTYPE KEY2*) [Function]

A generalized entry for inserting and retrieving values; provides certain options not available with GETHASHFILE or PUTHASHFILE. LOOKUPHASHFILE looks up *KEY* in *HASHFILE*.

*CALLTYPE* is an atom or a list of atoms. The keywords are interpreted as follows:

|          |                                                                                                                                             |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------|
| RETRIEVE | If <i>KEY</i> is found, then if <i>CALLTYPE</i> is or contains RETRIEVE the old value is returned from LOOKUPHASHFILE; otherwise returns T. |
| DELETE   | If <i>CALLTYPE</i> is or contains DELETE, the value associated with <i>KEY</i> is deleted from the file.                                    |
| REPLACE  | If <i>CALLTYPE</i> is or contains REPLACE, the old value is replaced with <i>VALUE</i> .                                                    |
| INSERT   | If <i>CALLTYPE</i> is or contains INSERT, LOOKUPHASHFILE inserts <i>VALUE</i> as the value associated with <i>KEY</i> .                     |

Combinations are possible. For example, (RETRIEVE DELETE) deletes a key and returns the old value.

(PUTHASHTEXT *KEY SRCFIL HASHFILE START END*) [Function]

Puts text from stream *SRCFIL* onto *HASHFILE* under *KEY*. *START* and *END* are passed directly to COPYBYTES.

(GETHASHTEXT *KEY HASHFILE DSTFIL*) [Function]

Uses COPYBYTES to retrieve text stored under *KEY* on *HASHFILE*. The bytes are output to the stream *DSTFIL*.

## Functions for Manipulating Hash Files

(HASHFILEP *HASHFILE WRITE?*) [Function]

Returns *HASHFILE* if it is a valid, open hash file datum, or returns the hash file datum associated with *HASHFILE* if it is the name of an open hash file. If *WRITE?* is non-NIL, *HASHFILE* must also be open for write access.

(HASHFILEPROP *HASHFILE PROPERTY*) [Function]

Returns the value of a *PROPERTY* of a *HASHFILE* datum. Currently accepted properties are: NAME, ACCESS, VALUETYPE, ITEMLength, SIZE, #ENTRIES, COPYFN and STREAM.

(HASHFILENAME *HASHFILE*)

[Function]

Same as (HASHFILEPROP *HASHFILE* 'NAME).

(MAPHASHFILE HASHFILE MAPFN DOUBLE)

[Function]

Maps over *HASHFILE* applying *MAPFN*. If *MAPFN* takes two arguments, it is applied to *KEY* and *VALUE*. If *MAPFN* only takes one argument, it is only applied to *KEY* and saves the cost of reading the value from the file. If *DOUBLE* is non-NIL, then *MAPFN* is applied to (*KEY1 KEY2 VALUE*), or (*KEY1 KEY2*) if the *MAPFN* only takes two arguments.

(REHASHFILE HASHFILE NEWNAME)

[Function]

As keys are replaced, space in the data section of the file is not reused (through space in the key section is). Eventually the file may need rehashing to reclaim the wasted data space. *REHASHFILE* is really a special case of *COPYHASHFILE*, and creates a new file. If *NEWNAME* is non-NIL, it is taken as the name of the rehashed file.

The system automatically rehashes files when 7/8 of the key section is filled. The system prints a message when automatically rehashing a file if the global variable *REHASHGAG* is non-NIL.

Certain applications save data outside Hash's normal framework. Hash files for those applications need a custom *COPYFN* (supplied in the call to *CREATEHASHFILE*), which is used to copy data during the rehashing process. The *COPYFN* is used as the *FN* argument to *COPYHASHFILE* during the rehashing.

(COPYHASHFILE HASHFILE NEWNAME FN VALUETYPE LEAVEOPEN) [Function]

Makes a copy of *HASHFILE* under *NEWNAME*.

Each key and value pair is moved individually, and, if *FN* is supplied, is applied to (*KEY VALUE HASHFILE NEWHASHFILE*).

What is returned is used as the value of the key in the new hash file. (This lets you intervene, perhaps to copy out-of-bank data associated with *VALUE*.)

*VALUETYPE* is a no-op.

If *LEAVEOPEN* is non-NIL, the new hash file datum is returned open. Otherwise, the new hash file is closed and the name is returned.

(HASHFILEPLST HASHFILE XWORD)

[Function]

Returns a Lisp generator for the keys in *HASHFILE*, usable with the spelling corrector. If *XWORD* is supplied, only keys starting with the prefix in *XWORD* are generated.

## Global Variables of Hash

HASHFILEDEFAULTSIZE

[Variable]

Size used when *#ENTRIES* is omitted or is too small. Default is 512.

HASHFILEDTBL

[Variable]

The hash file read table. Default is *ORIG*.

HASHLOADFACTOR

[Variable]

The ration, used slots/total slots, at which the system rehashes the file. Default is  $\uparrow A$ .

|                                                                                   |            |
|-----------------------------------------------------------------------------------|------------|
| HFGROWTHFACTOR                                                                    | [Variable] |
| The ratio of total slots to used slots when a hash file is created. Default is 3. |            |
| REHASHGAG                                                                         | [Variable] |
| Flags whether to print message when rehashing; initially off. Default is NIL.     |            |
| SYSHASHFILE                                                                       | [Variable] |
| The current hash file. Default is NIL.                                            |            |
| SYSHASHFILELST                                                                    | [Variable] |
| An Alist of open hash files. Default is NIL.                                      |            |

---

## Limitations

The system currently is able to manipulate files on CORE, DSK, FLOPPY, and over the network, via leaf servers. Hash files can be used with NS servers only if they support random access files.

Due to the pointer size, only hash files of less than 6 million initial entries can be created, though these can grow to 14 million entries before automatic rehashing exceeds the pointer limit. The total file length is limited to 16 milion bytes. No range checking is done for these limits.

Two-key files operate on pnames only, without regard to packages.



[This page intentionally left blank]

# HASH-FILE

---

Hash-File is similar to but not compatible with the library module, Hash. Hash-File is modeled after the Common Lisp hash table facility, and Hash was modeled after the Interlisp hash array facility.

Hash files, like hash tables, are objects which efficiently map from a given Lisp object, called the *key*, to another Lisp object, called the *value*. Hash tables store this mapping in memory, while hash files store the mapping in a specially formatted file. Hash files are generally slower to access than hash tables, but they do not absorb memory and they are persistent over Lisp images. Hash files are recommended for large databases which do not change very often.

Since hash files are not stored in memory, hashing for EQ or EQL keys does not make sense. Memory references written to file in one session will probably not be valid in another. For this reason, the default hashing is for EQUAL keys, and then only those which can be dependably printed and read.

All of the code for Hash-File is in a package called Hash-File. Throughout this document Lisp symbols are printed as though in a package which uses the packages Hash-File and Lisp.

## Requirements

---

Hash files must reside on a random-access device (not a TCP/IP file server).

## Installation

---

Load HASH-FILE.DFASL from the Library.

## Functions

---

Hash-File has functions to create a new hash file, to open and close existing hash files, and to store and retrieve data in hash files.

### Creating a Hash File

(make-hash-file *file-name* *size* &*key* . *keys*) [Function]

Creates and returns a new hash file in *file-name* opened for input and output. *Size* indicates the table size and should be an integer somewhat larger than the maximum number of keys under which you expect to store values in this hash file. (The hash file grows as required, so this number need not be accurate. See the section , "Rehashing," below.) The keyword arguments are explained as this document progresses.

### Opening and Closing Hash Files

(open-hash-file *file-name* &*key* :*direction* . *other-keys* ) [Function]

Opens an existing hash file and returns it. The *:direction* argument must be one of *:input* or *:io*. If opened for *:input* then storing values in the hash file is disallowed. The default for *:direction* is *:input*. Other key arguments are the same as for *make-hash-file* and are explained as this document progresses.

(close-hash-file *hash-file*) [Function]

Closes the file for *hash-file*, ensuring that all data has been saved. The backing file is always kept coherent; thus the only reason to close the *hash-file* is to ensure that the backing file is properly written to disk. All the functions mentioned in this document which operate on hash files open the file when necessary; thus it is safe to call *close-hash-file* at almost any time.

## Storing and Retrieving Data

(get-hash-file *key hash-file &optional default*) [Function]

Retrieves the value stored under *key* in *hash-file*. Returns *default* if there is nothing stored under *key*. The default for *default* is *nil*. Also returns a second value which is *true* if something was found under *key* and *false* otherwise.

(get-hash-file *key hash-file*) [Setf place]

Values can be stores in a hash file with:

(setf (get-hash-file *key hash-file*) *new-value*)

Accordingly, *incf*, *decf*, *push*, *pop* and any other macro that accepts generalized variables work with *get-hash-file*.

(map-hash-file *function hash-file*) [Function]

For each entry in *hash-file*, *function* is called with the key and value stored.

Note: It is unsafe to change a hash file while mapping over it. The integrity of the file may be lost.

(rem-hash-file *key hash-file*) [Function]

Removes any entry for *key* in *hash-file*. Returns *t* if there was such an entry, *nil* otherwise.

## Other Functions

(copy-hash-file *hash-file file-name &optional new-size*) [Function]

Makes and returns a hash file in *file-name* with the same contents as *hash-file*. Much slower than *il:copyfile*, but performs garbage collection, often resulting in a smaller file.

(hash-file-count *hash-file*) [Function]

Returns the number of entries in *hash-file*.

(hash-file-p *object*) [Function]

Returns *t* if *object* is a hash file, *nil* otherwise.

(hash-file-p *object*) ≡ (typep *object* 'hash-file)

## File Format

Hash-File uses a linked bucket implementation as illustrated in Figure 3.

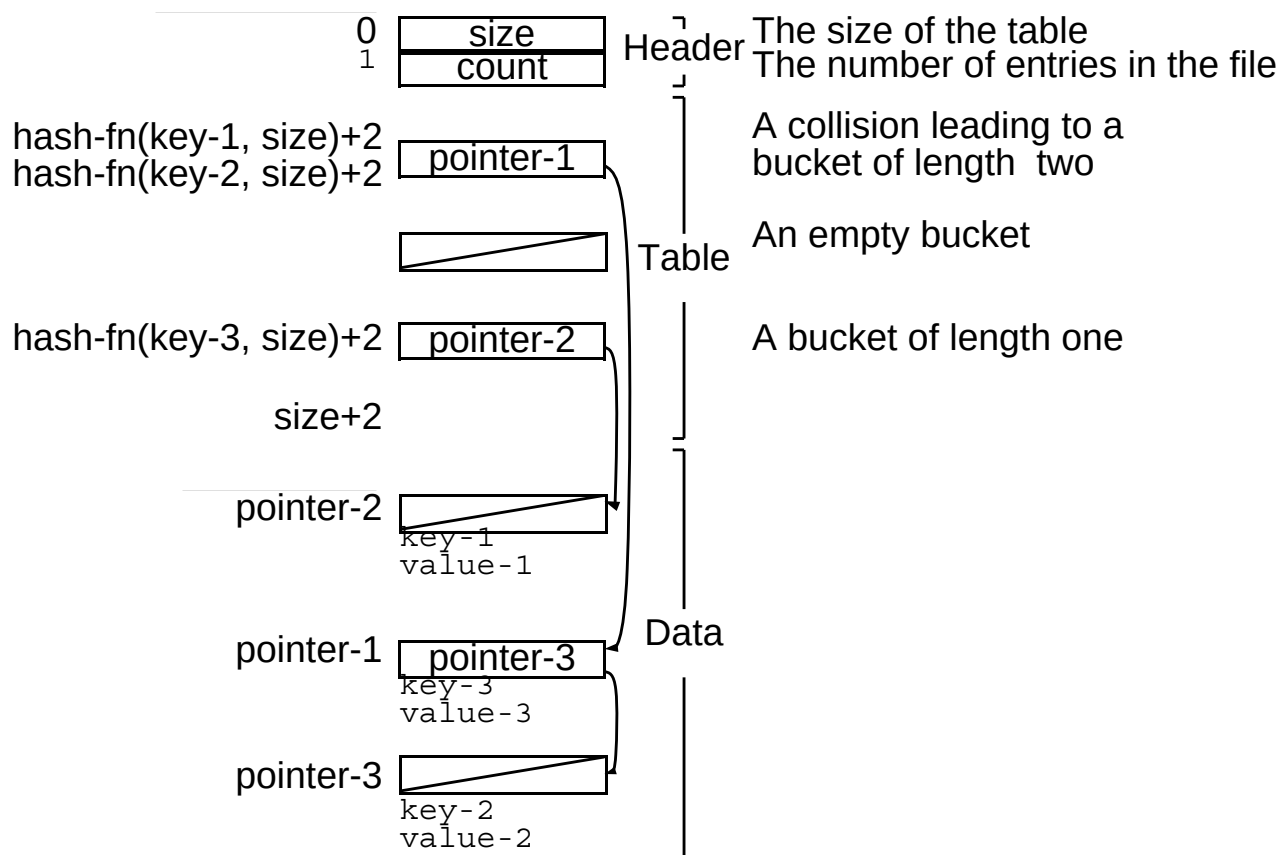


**Figure 3. Hash File Format**

Pointers are 32-bit integers written as four 8-bit bytes. There are two pointers of header (holding the size and count) followed by *size* pointers of table. Except for in the header and null pointers, all pointers are file-positions in bytes. Every such pointer points to the position on the file of the next pointer in the bucket. Immediately following the next pointer on the file are the printed representation of the key and value for the entry. New entries, including ones for old keys, are always added at the end of the file.

## Rehashing

When the number of keys with values in the file reaches a threshold, rehashing is performed to keep bucket lengths from getting too long. This threshold is expressed as a fraction of the table size.

rehash-threshold

[Keyword argument]

Should be floating point number between zero and one. When the product of the table size and the rehash threshold of a hash file is greater than its `hash-file-count` then the hash file is automatically rehashed. The default for this

keyword argument is the value of the special variable `hash-file::*rehash-threshold*` whose global binding is by default 0.875.

Rehashing is accomplished by having `copy-hash-file` make a new hash file with a larger size in a new version of the file. The new hash file structure is then smashed into the old one so that pointers to the old one are still valid.

`rehash-size` [Keyword argument]

Should be floating point number larger than one. The next prime larger than the product of this and the old table size is used to as the size for the new table. The default for this keyword argument is the value of the special variable `hash-file::*rehash-size*` whose global binding is by default 2.0.

`hash-file::*delete-old-version-on-rehash*` [Special variable]

If true, when rehashing generates a new version of the backing file the old version is automatically deleted. The default top-level value for this variable is `nil`.

Rehashing is very expensive. Thus, when possible, you should attempt to make good estimates for the `size` argument to `make-hash-file`.

---

## Programmer's Interface

---

There may be applications in which you want to store things in hash files but which could not be printed and read by the functions `print` and `read`. The following hooks are provided for this purpose.

`value-read-fn` [Keyword argument]

Called by `get-hash-file` with one argument of a stream to read a value. The file position is set to the same position as it was when this value was written. Default is `hash-file::default-read-fn` which binds `*package*` to the XCL package and `*readtable*` to the XCL readtable before calling `read`.

`value-print-fn` [Keyword argument]

Called by the `setf` method for `get-hash-file` with the object to be stored and the stream to print it on. The file position of the stream will be at the end of the file and there are no limitations as to how much can be printed. Default is `hash-file::default-print-fn` which binds `*package*` to the XCL package, `*readtable*` to the XCL readtable and `*print-base*` to 10 before calling `print`.

Example: A hash file with circular values.

```
(defun print-circular-object (object stream)
 (let ((*print-circle* t))
 (hash-file::default-print-fn object stream)))
(setq hash-file-with-circular-values
 (make-hash-file "{core}foo" 10
 :value-print-fn #'print-circular-object))
(setq l (list "foo"))
(setf (cdr l) l) ⇒ #1= ("foo" . #1#)
```

```
(setf (get-hash-file "bar" hash-file-with-circular-values) 1)
(get-hash-file "bar" hash-file-with-circular-values)
⇒ #1= ("foo" . #1#)
(eq * 1) ⇒ nil
```

key-read-fn [Keyword argument]

Called by `get-hash-file` with one argument of a stream to read a key. The file position is set to the same position as it was when this key was written. Default is `hash-file::default-read-fn`, described above.

key-print-fn [Keyword argument]

Called by the `setf` method for `get-hash-file` with the object to be stored and the stream to print it on. The file position of the stream is at the end of the file and there are no limitations as to how much can be printed. Default is `hash-file::default-print-fn`, described above.

Note: The value reader is called immediately after the key reader. Thus, the key reader must be sure to read all that the key printer printed so that the file position is appropriate for the value reader. However, the value reader is free to not read all that the value printer printed.

You might now think that you could make a hash file whose keys were circular by simply specifying our circular reader and printer for the key print and read functions, but this would not be sufficient. You also need the following hooks:

key-compare-fn [Keyword argument]

Called when searching a bucket to determine whether the correct key/value pair has been reached yet. Default is `equal`.

key-hash-fn [Keyword argument]

Called with a key and a range. Should return an integer between zero and `range-1` with the following property:

$$\text{key-hash-fn}(x) = \text{key-hash-fn}(y) \text{ iff } \text{key-compare-fn}(x, y)$$

The default `key-hash-fn` is `hash-file::hash-object` which works on symbols, strings, lists, bit-vectors, pathnames, characters and numbers. (Any object whose printed representation can be dependably read in as an object equal to the original.)

Note: This function will work on circular lists, as it only proceeds a fixed depth down a structure. Thus to hash on circular keys you also need to provide a key comparer which is able to compare circular keys, as most definitions of `equal` are not.

## Performance

A linked bucket implementation generally gives shorter bucket lengths, but uses more file space. The effects of this upon performance are difficult to judge.

The following table shows the distribution of bucket lengths in a Where-Is hash file containing 27,157 entries with a table size of 50,021.



| <b>length</b> | <b>number of buckets this length</b> |
|---------------|--------------------------------------|
| 0             | 29,279 (empty buckets)               |
| 1             | 15,461                               |
| 2             | 4334                                 |
| 3             | 794                                  |
| 4             | 125                                  |
| 5             | 23                                   |
| 6             | 4                                    |
| 7             | 1                                    |

This information was gathered by the function `hash-file::histogram`.



[This page intentionally left blank]

# HRULE

---

HRule is a module that lets you create horizontal rules (solid horizontal lines of various thicknesses) in a TEdit document. Rules are often used to set off titles and page headings from regular text, and to create decorative effects.

## Requirements

---

IMAGEOBJ  
EDITBITMAP  
TEDIT

## Installation

---

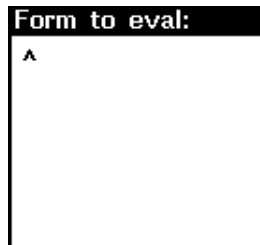
Load `HRULE.LCOM` and the required `.LCOM` modules from the library.

## Creating Horizontal Rules

---

You specify a rule's thickness in decimal fractions of a printer's point (1/72 of an inch).

To create a horizontal rule, place the caret at the point in your document where you want the rule to begin, then type Control-O. This will bring up a small window titled "Form to Eval" that contains a blinking caret. Type `(HRULE.CREATE N)` after the caret, with *N* indicating the thickness of the rule.



For example, to create a 4-point rule you would type `(HRULE.CREATE 4)`; to create a 2½-point rule you would type `(HRULE.CREATE 2.5)`. Then press the carriage return. The window closes, and a rule of the specified size is created, extending from the TEdit caret to the right margin of the paragraph.

Note: This means that nothing can appear to the right of a rule on the same line.

So, for example if you type the following paragraph

This is an example of a paragraph that is about to  
have a horizontal rule inserted in it, to show what  
happens.

and insert a 2½-point rule after the word "rule," you end up with

This is an example of a paragraph that is about to have a horizontal rule inserted in it, to show what happens.

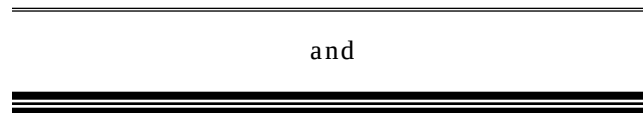
Like other image objects in TEdit, a rule is a single character that can be deleted, moved, and copied like any other character.

You can use the TEdit Paragraph Looks menu to change the width of a rule if you don't want it to extend to the normal right margin of your document.

## Stacking Several Rules in a Single Object

---

Sometimes, you may want to stack several rules atop one another, with space between them. This can be used to achieve effects like



To create built-up rules of this type, follow the same procedure as above, but provide a list of rule widths and spacings in place of the single rule width. The first example above was created using the form (HRule.CREATE ' (.5 .5 .5)), and the second example was created using the form (HRule.CREATE ' (3 1 1 1 3)). The first number in the list is the thickness of the topmost rule, the next number is the space below it, the third number is the next rule, and so on.

## Limitations

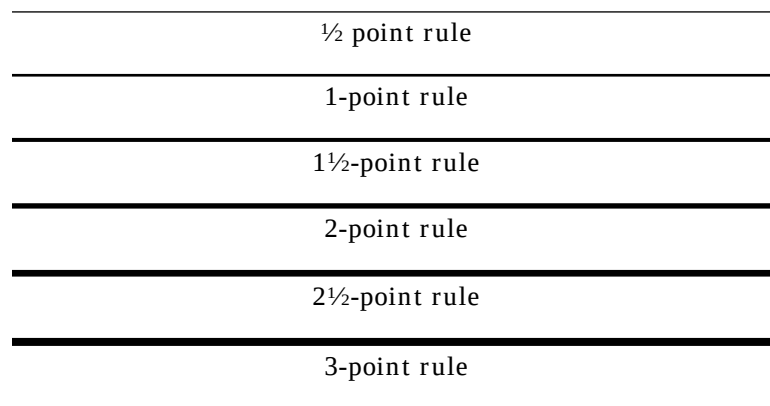
---

Theoretically, a rule can be infinitely small or infinitely large. For most documents, however, you will probably want to create rules that are between half a point and six points thick. On printers, you usually cannot tell the difference between rules that are less than ½ point apart in thickness.

## Examples

---

Shown in Figure 4 are some examples of horizontal rules. In addition, you might want to look at the rules in this document, which were all created with HRule.



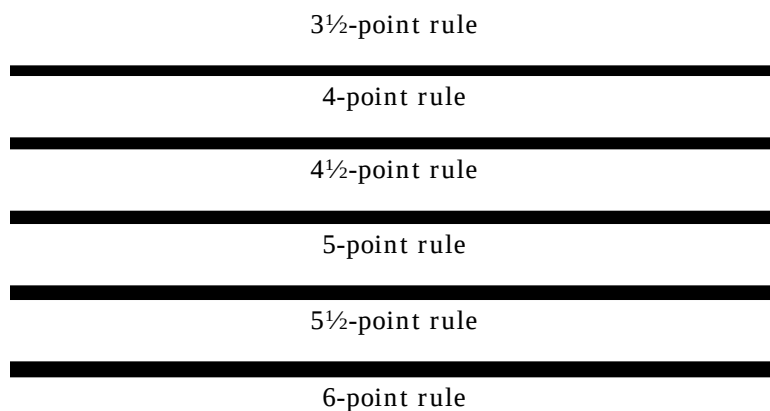


**Figure 4. Horizontal rules**

Shown in Figure 5 are some examples of built-up rules, along with what you would type to create them:

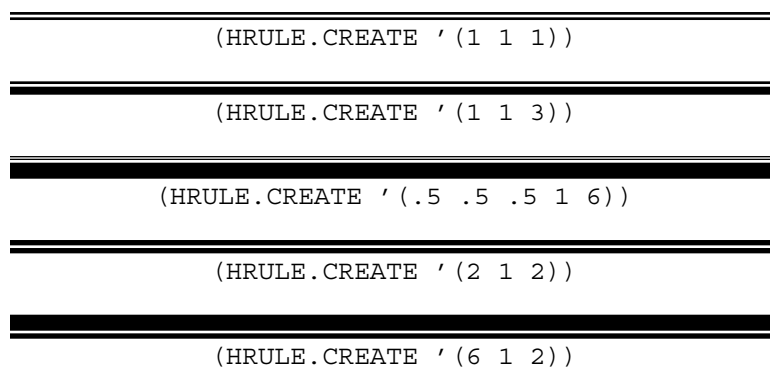


**Figure 5. Built-up rules**

[This page intentionally left blank]

# KERMIT AND MODEM

---

Kermit and Modem are utilities for transferring files between computers using ordinary RS232 and modem connections.

The file `KERMIT.LCOM` contains both the Kermit and Modem protocols. Once loaded, it provides a means of transferring files between a Xerox workstation and any other computer that supports either Kermit or Modem, and to which Lisp is able to open a Chat connection.

Of these two file transfer protocols, Kermit is preferred. Modem is much less flexible than Kermit, and cannot be used on RS232 connections requiring parity or flow control. Modem was developed primarily to support file transfers to and from microcomputers running the CP/M operating system. Modem implementations are available for Tops-20, VAX/UNIX, and VAX/VMS. Kermit, on the other hand, was designed for file transfers between computers of many types, and there exist implementations of the Kermit protocol on machines ranging in size from eight-bit microcomputers to large IBM mainframes.

For a detailed discussion and tutorial on Kermit, see *Kermit: A File Transfer Protocol* by Frank Da Cruz, Digital Press, 1987.

## Requirements

---

The machine must run Kermit or Modem, and you need the means of reaching it, typically via Chat over an RS232 or a network connection.

You also need the following `.LCOM` files to run this module successfully:

- `KERMIT`, `KERMITMENU`
- `CHAT`

and either the RS232C or TCP-IP protocols, or the built-in NS or PUP protocols.

## Installation

---

Load `KERMIT.LCOM` and the required `.LCOM` modules from the library.

## Establishing a Connection

---

The first step in using Kermit or Modem is to establish a Chat connection with a desired host. You may use any sort of Chat connection (e.g., NS, TCP, PUP, or RS232). See the Chat module in this manual.

If you are using an RS232 connection, and plan to transfer files with the Modem protocol, do not establish a connection that requires parity to be used; establish the connection with eight bits per character and no parity (see the RS232 module in this manual). Disable flow control (XOn/XOff) when using Modem.

When you have established a Chat connection to a remote host, log in (if necessary) and start the remote host's Kermit or Modem program. The details of running these

programs differ slightly between implementations; you should obtain documentation specific to the version of Kermit or Modem running on the remote host.

## Kermit

---

### Remote Kermit in Server Mode

Most mainframe implementations of Kermit have a server mode. This mode causes the remote Kermit to listen for either send or receive requests without your having to type additional commands to the remote Kermit. If the version of Kermit you are using on the remote host does support server mode, give the server mode command to place the program in this mode. In most implementations of Kermit, server mode is entered by your typing **SERVER** to the Kermit prompt:

```
Kermit>SERVER
```

### Remote Kermit Not in Server Mode

If the remote Kermit does not support server mode, you must issue individual send and receive requests for each file you transfer. To send a file to a remote Kermit, issue the **RECEIVE** command to the remote Kermit. To receive a file from a remote Kermit, issue the **SEND** command to the remote Kermit. In most cases, these commands are followed by the name of the file to be sent or received.

For example:

```
Kermit> RECEIVE FILENAME
```

or

```
Kermit> SEND FILENAME
```

If you are transferring files between two Xerox workstations connected by an RS232 connection, call (**CHAT 'RS232**) on each machine to establish the connection.

Currently, Lisp Kermit does not support a server mode, so you must issue a receive request on one machine, followed by a send request on the other (see below).

After you have started the remote Kermit program, you need to start the local Lisp Kermit program. Lisp provides both functional and interactive interfaces for Kermit (and Modem).

### Local Kermit

To start the local side of the Kermit file transfer, use the **KERMIT.SEND** or **KERMIT.RECEIVE** functions:

```
(KERMIT.SEND LOCALFILE REMOTEFILE WINDOW TYPE) [Function]
```

*LOCALFILE* is the name of the file being sent to the remote Kermit.

*REMOTEFILE* is the name under which the file should be stored remotely. In most implementations of Kermit, this name overrides any name you specified in the remote receive command.

*WINDOW* is a pointer to the Chat window over which the transfer takes place. If *WINDOW* is **NIL**, the value of **CHATWINDOW** (the first Chat window to be opened) is used in its place.

*TYPE* is the type of the file. It should be set to either TEXT or BINARY.



(KERMIT.RECEIVE *REMOTEFILE LOCALFILE WINDOW TYPE*) [Function]

*LOCALFILE* is the local name of the file to be received from the remote Kermit.

*REMOTEFILE* is the name of the file on the remote machine.

*WINDOW* is a pointer to the Chat window over which the transfer takes place. If *WINDOW* is NIL, the value of CHATWINDOW (the first Chat window to be opened) is used in its place.

*TYPE* is the type of the file. It should be set to either TEXT or BINARY.

While the file transfer is in progress, the associated Chat window is blank, and cumulative packet counts and other messages are displayed in a one-line prompt window above the Chat window.

---

## Modem

To transfer files with the Modem protocol, you must run the Modem program on the remote machine. Modem does not support a server mode. Typically, you run the program once per file transferred, with instructions in the command line to indicate whether the file is being sent or received. There are a number of versions of the Modem protocol. On some systems, you run the program called Modem; on other systems, the program is called UModem or XModem.

On UNIX, for instance, to send a text file to a Xerox workstation, you would type:

%XMODEM -ST *FILENAME*

On Tops-20, you would type:

@MODEM SA *FILENAME*

Note: % and @ are host system prompts.

As with Kermit, after you have started the remote side of the file transfer, you must start the local (Lisp) side. To do this, use either of the functions MODEM.SEND or MODEM.RECEIVE:

(MODEM.SEND *LOCALFILE WINDOW TYPE EOLCONVENTION*) [Function]

*LOCALFILE* is the name of the file to send to the remote Modem program.

*WINDOW* is the Chat window over which the transfer takes place.

*TYPE* is the file type, either TEXT or BINARY.

*EOLCONVENTION* is the end-of-line convention used by the operating system on which the remote Modem program is running. *EOLCONVENTION* should be one of CR, LF, or CRLF. Typically, UNIX and VMS require LF, Tops-20 requires CRLF, and other Xerox machines require CR.

(MODEM.RECEIVE *LOCALFILE WINDOW TYPE EOLCONVENTION*) [Function]

*LOCALFILE* is the name of the file to receive from the remote Modem program.

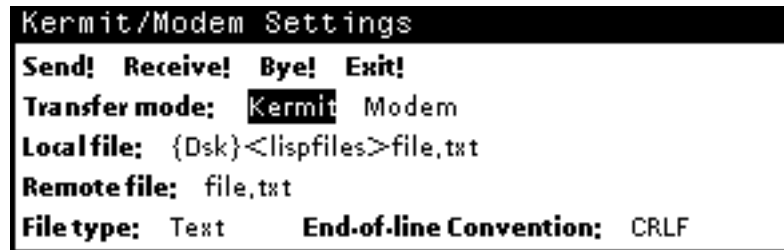
*WINDOW* is the Chat window over which the transfer takes place.

*TYPE* is the file type, either TEXT or BINARY.

*EOLCONVENTION* is the end-of-line convention used by the operating system on which the remote Modem program is running (see above).

## Interactive File Transfers With Kermit or Modem

A more convenient user interface for Kermit and Modem is available via the module `KERMITMENU.LCOM`. It provides a menu-oriented interface for issuing Kermit or Modem commands. To obtain the menu interface, press the middle mouse button in a live Chat window. The standard middle-button Chat menu contains an entry labeled "Kermit" near its top. If you select this entry, a Kermit menu appears at the top of the associated Chat window:



The entries on the top line of the menu are action commands:

|               |                                                                                                                                                                                                                                         |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SEND          | Starts sending a file to the remote Kermit or Modem program. The remote program must be prepared to receive the file.                                                                                                                   |
| RECEIVE       | Starts receiving a file from the remote Kermit or Modem program. The remote program must already be attempting to send the file.                                                                                                        |
| BYE           | Closes (severs) the connection.                                                                                                                                                                                                         |
| EXIT          | Closes the window containing the menu, but does not close the connection.                                                                                                                                                               |
| TRANSFER MODE | Controls whether files are transferred using Kermit or Modem. You may set the state of this entry by selecting either of the Kermit or Modem labels with the mouse. The current transfer mode choice is displayed inverted in the menu. |
| LOCAL FILE    | Holds the name of the local file being sent or received. You may set the contents of this field by selecting the LOCAL FILE label and typing the name.                                                                                  |
| REMOTE FILE   | Holds the name of the remote file being stored or retrieved. You may set the contents of this field by selecting the REMOTE FILE label and typing the name. The Modem protocol does not use the contents of this field.                 |
| FILE TYPE     | Controls whether files are sent in binary or text (ASCII) mode. To set this field, select the FILE TYPE label and choose an entry from the menu that appears.                                                                           |

END-OF-LINE CONVENTION    Sets the end-of-line convention being used by the remote Modem program (it is not used when files are transferred in Binary mode or with the Kermit protocol). The contents of this field must match the conventions of the operating system on which the remote Modem program is running. To set this field, select the END-OF-LINE CONVENTION label, and choose an entry from the menu that appears.

## Limitations

---

Transfer files between two Xerox machines using the Kermit protocol.

Modem cannot be used on RS232 connections requiring parity or flow control.

[This page intentionally left blank]

# KEYBOARDEDITOR

---

KeyboardEditor is intended for use with the VirtualKeyboards module. You should read that module's documentation before reading this. The KeyboardEditor module lets you create new virtual keyboards and change existing ones to suit your needs.

## Requirements

---

VIRTUALKEYBOARDS

## Installation

---

Load KEYBOARDEDITOR.LCOM and VIRTUALKEYBOARDS.LCOM from the library.

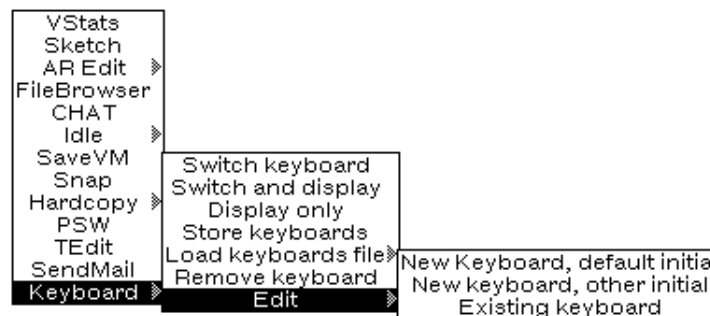
## User Interface

---

Loading KeyboardEditor adds EDIT to the Virtual Keyboard submenu on the background menu.

### Background Menu

The keyboard editor is used to modify and create virtual keyboards. You can call it by selecting EDIT from the main KeyboardEditor/VirtualKeyboards menu and sliding the cursor to the right to bring up the editor menu. You can also simply select EDIT, which gives you the same options as NEW KEYBOARD, DEFAULT INITIAL.



### Creating a New Keyboard From a Copy of the Default Keyboard

Choose NEW KEYBOARD, DEFAULT INITIAL to create a keyboard from a copy of the default keyboard (which initially has the same key assignments as the 1108 keyboard). The system prompts you for a name for the new keyboard, then call the editor with a copy of the default keyboard as the initial keyboard. The key assignments that are not changed during the editing session remain as they are in the default keyboard.

## Creating a New Keyboard From a Copy of Any Known Keyboard

To create a new keyboard from a copy of a known keyboard other than the default keyboard, select `NEW KEYBOARD, OTHER INITIAL` from the Edit submenu. You are prompted for a name for the new keyboard. The system then displays a menu of the known keyboards from which to choose the initial keyboard.

```
Quit
DEFAULT
EUROPEAN
logic
MATH
OFFICE
DVORAK
GREEK
ITALIAN
SPANISH
FRENCH
GERMAN
STANDARD-RUSSIAN
```

## Changing an Existing Keyboard

You can change an existing keyboard by selecting `EXISTING KEYBOARD` from the Edit submenu. Like the `NEW KEYBOARD, OTHER INITIAL` command, this brings up a menu of known keyboards from which you can choose a keyboard for editing. However, you are not prompted for a keyboard name first, because you are editing the actual keyboard rather than using it as a base for a new keyboard.

## Calling the Keyboard Editor From Lisp

The editor can also be called using the function

`(EDITKEYBOARD KEYBOARD INITIALKEYBOARD)` [Function]

where *KEYBOARD* is either a virtual keyboard (i.e., a list) or the name of a virtual keyboard. If *KEYBOARD* is a virtual keyboard or the name of a known keyboard (a keyboard that was defined before), the editing is done on that keyboard and the second argument is ignored.

If *KEYBOARD* is a new name, the editing is done on a copy of *INITIALKEYBOARD*, with *KEYBOARD* as its new name. If *INITIALKEYBOARD* is NIL, the default keyboard is used as a base keyboard.

Examples:

To create a totally new virtual keyboard, call `(EDITKEYBOARD NEWNAME)`.

To create a new keyboard that is similar to a keyboard with the name K1, call `(EDITKEYBOARD NEWNAME 'K1)`

To modify a keyboard with the name GREEK, call `(EDITKEYBOARD 'GREEK)`.

---

## Using the Keyboard Editor

There are four different keyboard editor menus, three of them displayed at any given time. After you call the editor, the command menu is at the top, the character menu in the middle, and the keys menu at the bottom.

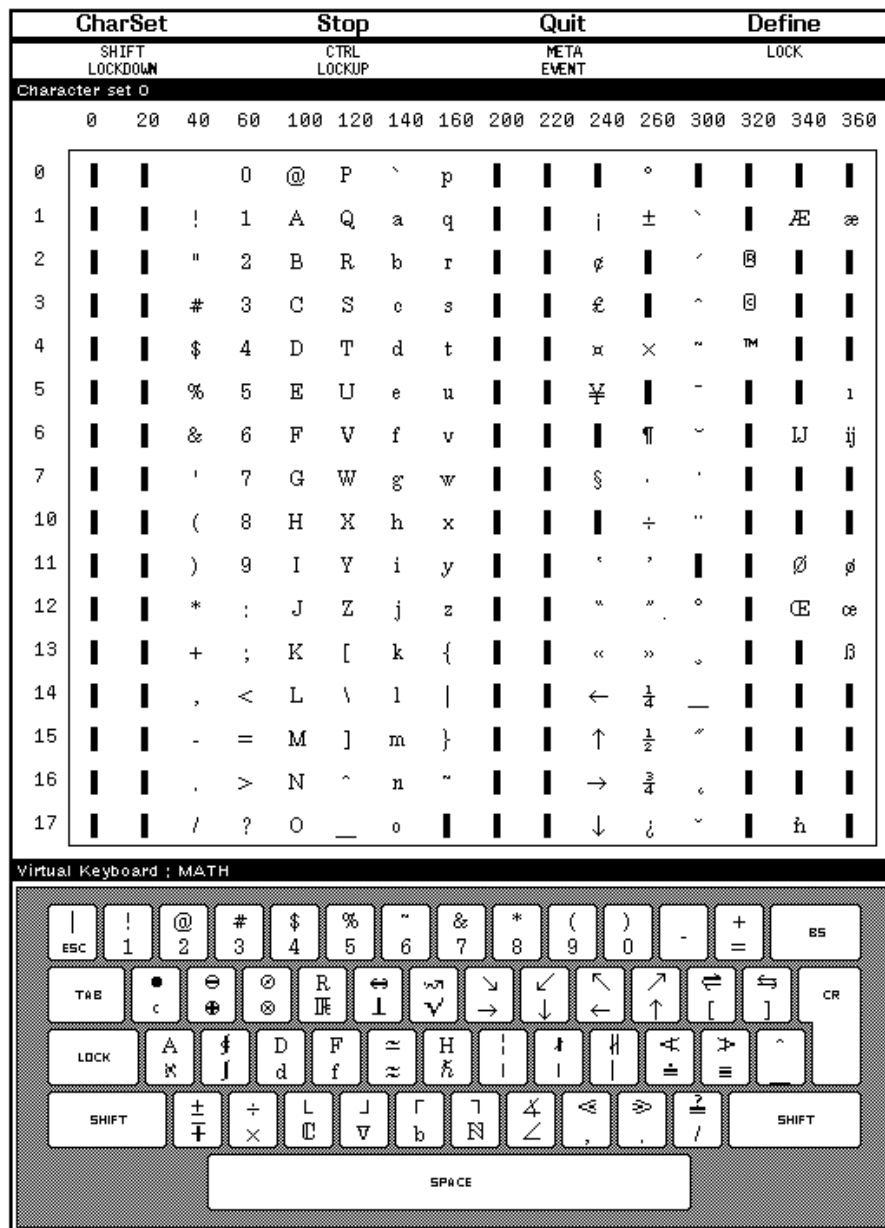


**Figure 6. Character Display**

The character menu is a 16-by-16-character display of the 256 characters available in the current character set. The set that is displayed when you enter the editor is character set 0, which includes all of the ASCII characters plus many other symbols. See Figure 6. If you need characters from other character sets, you have to select Char Set from the command menu. A new menu pops up that contains numbers from 0 to 377 octal. This is the character set menu, and it lets you switch the character menu to display characters from other sets. Most of the character set numbers are not currently implemented. The most useful ones are shown in Figure 7.

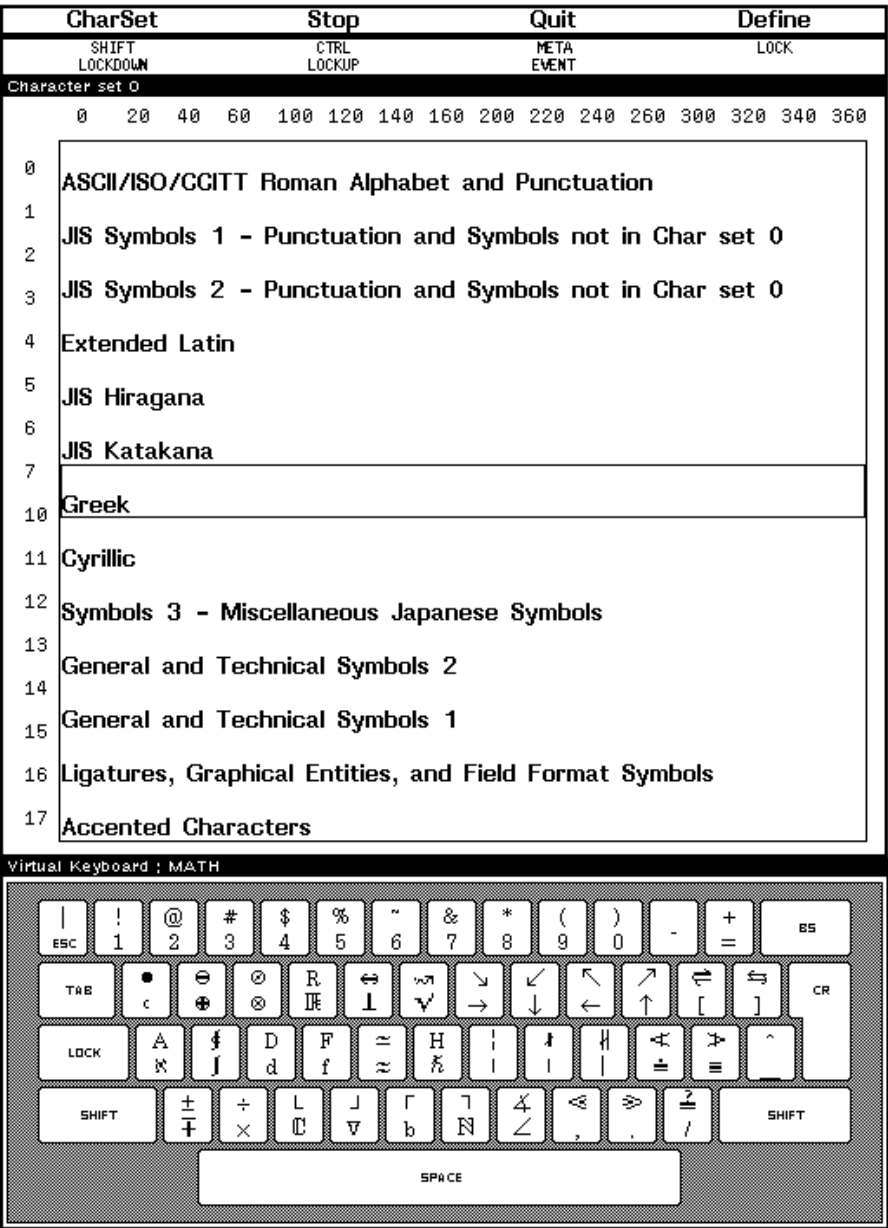


Figure 7. Character Sets

The keys menu lets you make a key the current key by selecting it. A selected key is marked by a black frame. To make a shifted key the current key, Shift-select the key (hold the Shift key down and click on the icon with the left button); it is marked by inverted Shift keys in addition to the black frame.

The basic operation of editing is assigning a character to a key. You can only assign character keys; keys other than character keys retain their current definitions. You assign a character to a key by selecting the key from the keys menu, then selecting the character from the character menu. If the character is to be assigned to the shifted key, select the shifted key as the current key.

A second type of editing operation is to change the LOCKSHIFT state of a key. Each key either has or does not have a LOCKSHIFT property. If a key has a LOCKSHIFT property and the shift lock key of the keyboard is down, typing the key on your



workstation keyboard sends the shifted character of the key, regardless of the state of the shift keys. The same rule applies to a virtual displayed keyboard; if the LOCK item is inverted and the key has a LOCKSHIFT property, selecting a key sends the shifted character to the current input stream.

If a key has the LOCKSHIFT property, the lock key is inverted in the keys menu. To change the LOCKSHIFT property of a key, first make the shifted key the current key. Then set or unset the LOCKSHIFT property by selecting the lock key from the keys menu.

If you are creating a new keyboard and you are satisfied with the key assignments, select Define from the command menu. This adds the newly created keyboard to the list of known keyboards (it will thus appear on future menus). Quit exits after modifying the virtual keyboard; Stop exits without modifying the keyboard. In both cases the new keyboard is returned to the caller of EDITKEYBOARD function (above).

## Creating New Keyboard Configurations

KEYBOARDCONFIGURATION

[Record]

Describes a physical keyboard: its layout, the key numbers that are used with KEYACTION. It also describes each key: its default meaning, its default label, whether you can change the key's meaning with the keyboard editor.

A configuration consists of a number of parts:

CONFIGURATIONNAME

[Record field]

The name of this configuration.

For example, KeyboardEditor comes with configurations named DANDELION (1108), DORADO (1132), DOVE (1186), and FULL-IBMPC.

KEYSIDLIST

[Record field]

An Alist of the IDs you use for the keys in the rest of the configuration; i.e., your names for the keys. For simplicity, these are usually numbers starting beyond 100 (to avoid overlapping the true range of key numbers).

KEYREGIONS

[Record field]

An Alist of key IDs and the regions they occupy in the keyboard's image when it is displayed. For example, the alphabetic keys in the DANDELION keyboard are 29 screen points wide and 33 high.

DEFAULTASSIGNMENT

[Record field]

An Alist of key IDs and their default KEYACTIONS (see *IRM*).

KEYNAMESMAPPING

[Record field]

An Alist of key names to key IDs. The key names should be mnemonic, and should distinguish relevant differences; e.g., the 7 on the 1186's numeric keypad is named NUMERIC7, while the 7 key in the main keyboard cluster is named 7.

MACHINETYPE [Record field]

The kind of machine for which this configuration is intended.

For example, the FULL-IBMPC configuration is meant to be used with a DAYBREAK keyboard, so its MACHINETYPE is DAYBREAK.

KEYLABELS [Record field]

An Alist of key numbers to special labels. This is used to label keys such as the “Next” key, where the key assignment may not be a printable character.

KEYLABELSFONT [Record field]

The font you want to use for the key labels. The default value is Helvetica 5.

BACKGROUNDSHADE [Record field]

The shading for the non-key parts of the virtual keyboard’s image. This defaults to a reasonable gray value.

KEYBOARDDISPLAYFONT [Record field]

The font used to display actual character assignments. This should probably be Classic 12, since it is the most complete font.

CHARLABELS [Record field]

An Alist from character codes to names. Used to give symbolic names to characters such as ESCAPE, which do not otherwise print.

ACTUALKEYSMAPPING [Record field]

A function that takes one of your key IDs and returns a true key number, for use by KEYACTION.

Note: To create a new configuration, create an instance of the KEYBOARDCONFIGURATION record, using the field names shown above. Then add it to the list VKBD.CONFIGURATIONS. You may then edit it using the configuration editor described below.

Note: You must save your own configurations. There is no user interface for saving them, nor any automatic scheme.

---

## Editing a Keyboard Configuration

Once you have created a KEYBOARDCONFIGURATION, you can make modest changes to it using the function:

(EDITCONFIGURATION *CONFIGNAME*) [Function]

where *CONFIGNAME* is the CONFIGURATIONNAME you have assigned to your new configuration. This creates a virtual keyboard editing window with a menu on top of it as shown in Figure 8.

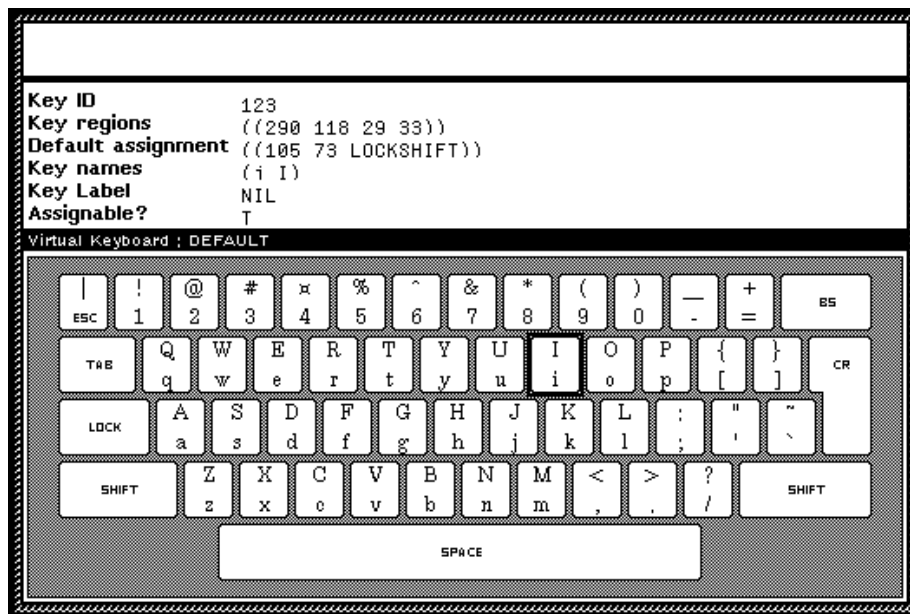


**Figure 8. Virtual Keyboard Editing Window**

Selecting a key with the mouse fills in the fields in the menu. The figure shows the 1108's configuration being edited, with the I key selected. To change one of the values, select the label at the left edge of the menu (e.g., ASSIGNABLE?). You are prompted to edit the existing value using TTYIN.

The keyboard image is not automatically updated. To refresh it, select REDISPLAY in the right-button window menu.

When you have finished editing, simply close the keyboard window.

[This page intentionally left blank]

## Lafite Changes

Folder names -> strings  
Profile handling (LAFITE.PROFILE.VARS, "Lafite.info")  
Binding \*UPPER-CASE-FILE-NAMES\*  
Lafite modes all active. LAFITE.USE.ALL.MODES = T or list of operations from {:GETMAIL, :POLL}  
Mail retrieved only from servers that claim to have new mail as of most recent poll.  
Handling NS chars, eol conventions in parser  
BADTOCFILE returns nil so don't get bogus "Folder is Empty"  
LAFITE.BROWSER.LAYOUTS, LAFITE.DISPLAY.SIZE  
\LAFITE.BROWSE no longer takes file & options  
new fn LAFITE.BROWSE.FOLDER  
toc handles ns strings  
LAFITE folder arg can be list of (folder region displayregion iconpos . options)  
LAFITE options can include :ACTIVE, :GETMAIL, :SHRINK  
Browser menu labels narrower  
LAFITE.FOLDER.MENU.FONT  
LAFITE.MIDDLE.UPDATE = ({:shrink|:close} {:update|:expunge})  
LAFITE.BROWSER.ICON.PREFERENCE is NIL (default), :ASK (prompt for position) or a fn.  
Recache command  
LAFITE.DONT.DISPLAY.HEADERS, LAFITE.DONT.FORWARD.HEADERS, LAFITE.DONT.HARDCOPY.HEADERS  
Fixed hardcopy toc  
Message display titlebar menu [LAFITE.EXTRA.DISPLAY.COMMANDS, LAFITE.LOOKS.SUBCOMMANDS]  
Middle-button display windows stay up  
Middle-button browser icon offers to GetMail if "active" (has had a getmail done to it or was the active file browsed when Lafite was turned on).  
LAFITE.SEND.FORMATTED = NIL|:ASK|T or list of (type answer).  
Can now send plain text msg with ns chars  
NS mail handles attachments, including references.  
NS mail handles TransportProblem.  
Copy selection in browser.  
Browse & Forget.  
MOVETOCONFIRMFLG only affects menu selections.  
EOM works again in LispCore  
LAFITE.SIGNATURE, LAFITE.GV.FROM.FIELD & handling of same when username changes.  
\*GV-SHOW-POSTMARK\* = T retains GV information.

## Deleted or renamed fns/vars:

LA.REMOVEDUPLICATES  
LA.SETDIFFERENCE  
PROFILEFILENAME  
\LAFITE.MERGE.PROFILES -> \LAFITE.MERGE.NAMELISTS  
LAFITEDISPLAYREGION  
LAFITEIMMEDIATECHANGESFLG, CHANGEFLAGINFOLDER  
MSGFOLDERTEMPLATE, MSGFOLDERICON, MSGFOLDERMASK -> LAFITE.FOLDER.ICON  
MSGUNSENTICON, MSGUNSENTMASK, MSGUNSENTREGION -> LAFITE.MSG.ICON  
use of FOLDERDISPLAYWINDOWS field, now uses FOLDERDISPLAYREGIONS (replace BROWSERSELECTIONREGION)  
\LAFITE.BROWSE family completely changed  
\LAFITE.MOVETO[.PROC] changed  
\LAFITE.GETMAILFOLDER deleted.

# Lafite

## The Interlisp Mail System

Documentation: {Eris}<Lisp>Library>Lafite.Press  
Program: {Eris}<Lisp>Library>Lafite.Dcom  
Revised: February 22, 1984 by Bill van Melle

### General comments

Lafite is the Interlisp system for reading and sending mail. Lafite retrieves inbound mail from the user's *inboxes* on one or more mail servers. Mail is retrieved into one or more *mail folders*, which can be any Interlisp-accessible file. One can open a *browser* on a mail folder, which allows the folder's contents to be displayed, deleted, moved into other folders, hardcopied, etc. Messages can be composed using the standard Interlisp text editor and all its facilities, then sent to other users.

Lafite permits communication with various sorts of mail servers. Currently implemented are interfaces to the Grapevine mail system and the NS mail system.

Mail systems are notorious for inspiring lengthy wish lists for new functionality or user interface embellishments. Feel free to send your suggestions to LafiteSupport (using Lafite's "Lafite Report" form, described below).

### Compatibility with Laurel

Previous users of the Laurel mail program will find Lafite to have a similar user interface. Some of the ways in which it differs from Laurel are the following:

Laurel can only access mail folders on the local disk; Lafite can access folders on remote file servers. Thus, it is not necessary to transfer mail folders back and forth between your local disk and your file servers.

Laurel can only "browse" one mail folder at a time; Lafite can have several mail folders "opened" at the same time. Utilizing the Interlisp window system, you can view the table of contents of many mail folders and refer to messages in them independently.

You may have multiple windows displaying messages and multiple windows for sending messages and you may move text freely among them.

Lafite can read mail files written by the Laurel and Hardy mail programs; the files it writes are in Laurel format.

### Loading Lafite

Lafite consists of two parts: a generic part that manages browsers, mail folders and message composition; and a protocol-specific part that manages communication between Lafite and the remote mail system. You have to load both parts. The first is the file `LAFITE.DCOM`, plus the files that it automatically loads. The second depends on your network environment. For sites that use the Xerox NS Mail system, you should load the file `NSMAIL.DCOM`. For Grapevine sites, load the file `MAILCLIENT.DCOM`. If your site has a choice of mail server types, you can load all the relevant protocol files and then choose among them by setting the mode (see **Lafite Modes**, below).

### Lafite operation

When Lafite is loaded, it can be started by calling `(LAFITE 'ON MAILFOLDER)`. Lafite will attempt to establish the user's mail server identity, read stored user data from the file `LAFITE.PROFILE` (if it exists) and bring up the Lafite Status window. It will then establish a browser for the file `MAILFOLDER`, creating an empty mail folder of that

name if one does not exist. If *MAILFOLDER* is not supplied then *ACTIVE.MAIL* will be used (see *DEFAULTMAILFOLDERNAME* below). If *MAILFOLDER* is supplied but is the atom *NIL*, then no mail folder is opened but you are free to send mail and open any mail folder at a later time.

Mail files must reside on randomly-accessible devices. In the current Interlisp environment, this means they must reside on {*DSK*}, {*FLOPPY*}, or a file server that supports the Leaf protocol.

There are three major types of windows used by Lafite: the Status Window; Browser Windows, which are views on particular mail folders; and Message Composition Windows. Each type of window has its own fixed menu of commands. In general, while a command is "in progress", the menu item that invoked it is greyed out. Commands may be selected with the *LEFT* or *MIDDLE* mouse buttons; in some cases, the *MIDDLE* button provides some sort of special treatment, described in the documentation of such commands.

## The Lafite Status Window

The Lafite Status window contains a small, fixed menu and a region for Lafite status information. While Lafite is in operation, it runs a background process—*LAFITEMAILWATCH*—that polls the user's inboxes periodically and reports in the status region if there is new mail. Clicking in the status region causes the background process to wake up and report status immediately, instead of waiting its normal interval. The commands in this window's menu are as follows:

**Browse** — pops up a menu of the mail folders that Lafite is aware that you have. Selecting 'Another Folder' prompts you for a folder name in the prompt window. After a folder is selected, a *browser* window onto that mail folder is opened.

Selecting the **Browse** command with the *MIDDLE* button brings up a menu of Browse-related commands. In addition to simple **Browse** there are these commands:

**Browse Laurel Folder** — Browses a file that was produced by the Laurel mail reader version 6.1 or later. Laurel 6.1 files are almost the same as Lafite files, but contain some line-formatting information that is stripped out by this command. After you have applied this command once to a file, you can subsequently browse the file with the normal **Browse** command (unless you use Laurel on it again, of course). **Important Note:** this command is not intended for repeated use, but simply to make mail files built by Laurel more pleasing to browse. This command has the side effect of destroying any Tedit formatted messages, so should be used with care.

**Forget Folder** — Removes a folder from the list of known mail folders.

**Forget Message Form** — Removes a message form from the list of known message forms (see **Save Form** command).

**Send Mail** — brings up an Interlisp text editor window on a message form. The form is a canonical "empty message" if **Send Mail** is selected with the *LEFT* button. If **Send Mail** is selected with the *MIDDLE* button, a menu is presented with the following choices:

**Standard Form**—provides an empty message template (same as using the *LEFT* button).

**Lisp Report**—provides a message template to report an Interlisp bug or make a suggestion.

**Lafite Report**—provides a message template similar to **Lisp Report** but sent to Lafite maintainers.

**Saved Form**—prompts for a form name, which can be any text file, or a form created by the **Save Form** command (below).

Also in the menu are any names of known user-defined message forms created by the **Save Form** command. Each message form runs in its own process, so you can have several in progress at once. When you have finished composing the message, click **Deliver** in the message's menu.

**Quit** — Stops Lafite and closes all browser windows. If any of the associated mail folders need updating, prompts with a menu asking what degree of updating should be performed (see **Update** command). It also saves the names of known mail folders and form files on the file `LAFITE.PROFILE` so that this information will be available when you next run Lafite. You may achieve the same result under program control by calling `(LAFITE 'OFF)`, rather than buttoning this menu item. Most users find that they never invoke 'Quit', but rather keep Lafite always active in the background.

Selecting the **Quit** command with the MIDDLE button brings up a menu of status changing commands. This menu includes items for changing Lafite's mode (see **Lafite Mode**), and the command **Restart**, which is equivalent to `(LAFITE 'OFF)` followed by `(LAFITE 'ON NIL)`.

## Browser Windows

A Browser window is a view onto a mail folder. The main part of the window displays the table of contents, a one-line summary of each message. This window is scrollable in both dimensions. Above the table of contents is a horizontal menu containing commands specific to the mail folder associated with the browser window. Above the menu is a prompt window, in which various status information related to the browser is printed, and where some information is prompted for.

Browser commands operate on the currently selected set of messages. A selected message is indicated by a black triangle to the left of its message number. A message can be selected by clicking anywhere inside its summary line. The type of selection depends on which button is used:

**left button** — selects just this single message, deselecting any other selected message.

**middle button** — adds a message to the current selection.

**right button** — extends the selection up or down. Deleted messages are not included in this extension unless the control key (CTRL) is down.

**shift key (SHIFT) and any button** — removes this message from the current selection.

Laurel users note: messages can be selected by clicking *anywhere* within the summary line (except for the mark area, see below), unlike in Laurel, where you must select at the left end.

The commands in the browser menu are as follows:

**Display** — displays the selected message. If the selected message is already displayed, the selection is advanced to the next undeleted message, and this message is displayed. If there is more than one message selected, buttoning **Display** cycles through the messages.

If you select **Display** with the LEFT button, the message is displayed in the primary message display window for the browser, replacing any previously displayed message. If you select **Display** with the MIDDLE button, the message is displayed in a newly created window, for which you will be prompted. Using the MIDDLE button you can make multiple windows containing messages for further reference (e.g., to use in composing your own message).

**Delete** — deletes the selected messages. A deleted message is indicated by a black line through its summary line. The message is not actually removed from the mail folder until you Expunge (see Update, below).

**Undelete** — undeletes the selected messages.



**Answer** — constructs a Message Composition Window containing an *answer template* for the current message. After you deliver the answer, an "a" will appear in the browser window as the message's mark.

**Forward** — similar to 'Answer' but the message form is a *forward template* for the currently selected messages. After you deliver the forwarded the messages, an "f" will appear as the message's mark.

**Hardcopy** — prints the selected messages on your local printing device. When the hardcopy is complete, the message's mark is changed to "h" if the message didn't already have a more interesting mark. Messages can be marked for hardcopy, but the actual printing deferred until later; see description of LAFITEHARDCOPYBATCHFLG. Currently, one should exercise care in hardcopying large sets of messages, as Lafite makes no attempt to perform the hardcopying in smaller pieces; too large a body of messages can make printers unhappy, or exhaust local disk space in the process.

**Move To** — pops up a menu of known mail folders and moves the selected messages to the chosen folder. A new mail folder can be created by selecting 'Another Folder' and typing in the mail folder name in the prompt window. You will then be asked to confirm the move. When the move is successful, the messages are marked deleted in the source browser window, and given the "m" mark.

The name of the mail folder you most recently moved messages to appears in the title bar of the browser window as the "Default 'Move To':" folder. You can "accelerate" subsequent Move operations by selecting the 'Move To' command with the MIDDLE button. This will perform the move to the Default 'Move To' folder without bringing up a menu.

**Update** — The changes that you make to your mail folder (deletions, changes of message marks, etc) are not actually transmitted to the physical mail file until you perform the Update command. There are two subcommand choices for Update:

**Write out Changes Only** — makes the browser and the mail file completely consistent with one another: if you were at this point to logout from Lisp, run Lafite in another incarnation of Lisp, and browse the same mail folder, you would get a browser that was in exactly the same state, deleted messages and all. This command involves writing out to your mail file and its table of contents the information about what has changed: new marks, deletions, newly parsed messages.

**Expunge Deleted Messages** — in addition to making the browser and the mail file completely consistent with one another, this command compacts your physical mail file so as to remove all messages marked deleted.

Which to choose? You eventually want to Expunge, so as to reduce the amount of file space your mail requires, but Expunge is not always fast. **Expunge** requires rewriting your mail file starting at the first deleted message, so is somehow "proportional" to the number of undeleted messages beyond that point. **Write out Changes Only** is proportional to the number of changes, and is thus usually faster than **Expunge**, except when the changes consist primarily of a string of deleted messages at the end of the folder.

If you Close or Shrink a Browser window that has had changes to it, you are prompted with a menu offering to **Write out Changes**, **Expunge**, or make no change before the window is closed/shrunk.

If you have deferred hardcopy, the Update menu also includes a choice **Do Hardcopy Only** if you want to print out your messages but not actually update the file.

**Get Mail** — brings new mail into the folder that this browser is viewing.

## Changing the Message Mark

Each message in a mail folder has a "message mark", which is an additional tidbit of information about the message displayed in the summary line in the browser. Messages originally come in with the mark "?", meaning they are unexamined. Some Lafite commands change the mark automatically. For example, displaying a message changes its mark from "?" to a blank; answering a message changes its mark to "a". You can change the mark directly by selecting with the mouse in the narrow area immediately to the left of the message number, where the mark is printed. Simply click in the mark position, and type the new mark (a single character).

## The Table of Contents

To speed the browse operation, Lafite maintains, for each mail folder you browse, a "table of contents" file, which contains all the information displayed in the browser window. This file is named by concatenating the name of the mail file with "-LAFITE-TOC". This file is completely redundant, in that if it doesn't exist, Lafite simply recomputes it by parsing the entire mail file. Lafite has some rudimentary checks to guarantee the consistency of the contents file with the mail file it describes; Lafite deletes and recomputes the contents if it suspects that the table of contents file is out of date.

## Message Composition Windows

On top of the text editor window is a horizontal menu for telling Lafite what to do with the message being created in the text editor window. When you have transformed the text to the desired message, select one of the following menu items:

**Deliver** — sends your message. This process happens in background, so you can proceed with anything else you desire while delivery proceeds. If successful, the window closes and an entry is made in your *outbox* (see below). If the delivery fails, for any of a variety of reasons (e.g. bad address fields, the mail server timed out) which will be displayed in the prompt window, the message is redisplayed and you are back in the text editor to re-edit and then resend the message. During the delivery process, the menu atop the message window changes into a single item, **Abort**; if you click this item, the delivery is aborted, and you are returned to editing your message.

**Save Form** — asks you for a file name on which to save this message for later use. This is the way to save a message form for later repetitive use. It does *not* send it. If no extension is given for the file name, it defaults to the value of LAFITEFORM.EXT (initially LAFITE-FORM).

Of course, you can always bring up the text editor command menu by MIDDLE buttoning the title bar of the editor window. You can then do any text editor command. If you select 'Quit' you will immediately leave the text editor, bypassing the above commands. You can also simply close a window using the normal window command menu if you decide you don't want to do anything with a message you started composing.

When editing message forms, fields that should be filled in are enclosed in ">>" and "<<"; e.g., >>Recipients<<, >>Subject<<. To make it easier to fill these in, the first field is highlighted in delete mode. Each successive field can be reached by typing the middle-blank key on the keyboard (or OPEN on the Dandelion keyboard, or whichever key you have assigned the TEdit "Next" syntax to). See the TEdit documentation for details.

In a saved form, if you insert a field ">>Self<<", it will be automatically filled with the name of the currently logged in user when retrieved into a message composition window. This facilitates making forms that many different people can use. Of course, if you edit an already saved form containing ">>Self<<" in a message composition window, you will have to perform the inverse, replacing your name with ">>Self<<", before saving it away again.

After a message is successfully delivered, it is entered into your *outbox*, a window attached to the bottom of your status window. This window contains a one line description of each of the most recent *n* messages you have sent,

where  $n$  is the value of the variable `LAFITEOUTBOXSIZE`, initially 2. The outbox is treated as a menu—selecting a line in it brings up the corresponding message for further editing and delivery. The outbox can be independently closed, if you are no longer interested in the messages displayed therein and want to free up the resources that they are tying down.

## Format of Message Headers

The header of a message contains a number of fields, each on a separate line, followed by a blank line to separate the header from the message body. Each header line consists of a field name followed by a colon, then the contents of the field. Messages standardly have "To:" and "cc:" fields consisting of one or more recipient names, separated by commas (and spaces if desired). Lafite automatically supplies a "From" field containing your name. If you want the message to appear to be "from" someone else (e.g., if you are sending the message from someone else's logged in Lafite), or from more than one user, you can supply your own "From" field. In this case, Lafite will supply a "Sender" field to show who actually sent the message.

## Sending Formatted Messages

You have available to you the full power of TEdit when you are composing a message. This means you can change fonts, format paragraphs, and insert "image objects". If you try to deliver such a formatted message, Lafite will ask if you want to retain the formatting information, putting up a menu with these choices:

**Send Formatted Message** — retains all the formatting information. Note that only Lafite users can read formatted messages; all other mail readers will see the plain text of the message. Even if all of the recipients are Lafite users, keep in mind that retaining the formatting information means in particular retaining the fonts you used in composing the message; not everyone likes to read mail in the font you chose, so if the only "formatting" is a nonstandard font that isn't important to the appearance of the message, do not make this choice.

This choice is made automatically if the message contains image objects, as there is no way to send the images without the formatting.

**Send Plain Text** — sends only the text of the message. The message will appear to the recipient in whatever font her mail reader standardly uses, and all paragraph formatting (centering, justification, special tabstops) will vanish.

**Abort** — does not send the message, but returns to the message editor to allow you to continue editing the message.

## Lafite Modes

Lafite is capable of sending and retrieving mail via different protocols. At any one time, however, it only operates with a single protocol set, which is considered its current "mode". The currently implemented modes are GV (Grapevine) and NS. The easiest way to change the mode is to use the MIDDLE-button menu under **Quit**. Lafite's mode can also be changed programmatically with the function `LAFITEMODE`:

```
(LAFITEMODE MODE)
```

Returns the current mode, a litatom. In addition, if *MODE* is non-NIL, changes Lafite's mode to be *MODE*.

When Lafite is first turned on, it chooses its mode as follows: If there is only one mode supported (i.e., only the files supporting one protocol have been loaded), it chooses that mode; otherwise, if `LAFITEMODEDEFAULT` is non-NIL, it chooses that mode; otherwise, the user must set the mode manually. The current mode is displayed in the

status window if `LAFITESHOWMODEFLG` is true and more than one set of mail protocol implementations has been loaded.

You can freely intermix mail of the various modes in one folder, but the current implementation is not very clever about it. For example, the `Answer` command always treats the selected message as if it were one in the current mode. So if you try to answer a Grapevine message while in NS mode, some confusion may result.

## Grapevine

The Grapevine implementation of mail protocols is obtained by loading the library file `MAILCLIENT.DCOM`.

Grapevine addresses are of the form "name.registry", e.g., "Carstairs.pa". If you omit the registry, it is defaulted to the value of `DEFAULTREGISTRY`. Arpanet recipients are of the form "name@host". In addition, the following forms of address are recognized, where "actual address" must be a valid Grapevine or Arpanet address:

Human sensible name <actual address>  
 actual address (random comments)  
 "Comments, including parentheses and commas" <actual address>

If you address a message to a public distribution list (a Grapevine name ending in the character "↑") and have not included a Reply-to field, Lafite will prompt you to supply one. Your choices are to include a Reply-to: self field, so that answers to this message will be sent only to you; a Reply-to: other field (you get to fill it in yourself), or no Reply-to: field at all. It is important to have a Reply-to field in messages sent to distribution lists, so that casual users will not inadvertently reply to the entire distribution list when a reply to you only was intended.

Please be careful in using public distribution lists. Keep in mind that your one message will be received by many people—is what you have to say important enough to be worth taking their time? There are certain organizational distribution lists, e.g., `AllPA↑.pa`, or `ISL↑.pa`, that you should definitely avoid sending casual messages to, as they are large and their members are not voluntary. Other distribution lists are really "interest lists" whose members have voluntarily consented to receive messages in the general category of movies, concert announcements, humorous anecdotes, or whatever. Messages to these lists are less constrained, but you should still take a moment to think about whether your message is appropriate. I highly recommend that all electronic mail users read Chapter 6 of the Laurel manual, "Message system mores".

*Shortcomings as of this writing: Private distribution lists are not yet supported. And the Answer command does not preserve the entire text of the variant forms listed above, only the "actual address".*

## NS Mail

The NS implementation of mail protocols is obtained by loading the library file `NSMAIL.DCOM`.

NS addresses are of the form "name:domain:organization", e.g., "Carstairs:PARC:Xerox". If you omit the domain and/or organization, they are defaulted to your own domain and organization. NS mail does not support the many variant forms of address that Grapevine does, because the header is not actually sent as text, but as a structured set of formally named recipients.

Some users of the NS Mail system send messages in a format that Lisp does not understand. The current Lafite implementation leaves such messages in your inbox to be retrieved by other means; all that you see back from GetMail is a header and a note that there was an attachment that Lafite couldn't read.

## Changing the Lafite User—Lafite customization

(LAFITE ON/OFF MAILFOLDER . OPTIONS)

Used for starting and stopping Lafite with an optional mail folder. If *ON/OFF* is the atom *ON* then Lafite will start processing using *MAILFOLDER*. If *MAILFOLDER* is not supplied then Lafite will use your *ACTIVE.MAIL* mail folder (actually, the value of *DEFAULTMAILFOLDERNAME*). If *MAILFOLDER* is supplied but is the atom *NIL* then no browser will be created but you can send messages and start browsing mail files later using the 'Browse' menu item.

If *ON/OFF* is the atom *OFF* then Lafite will close all mail folders, expunging all messages marked for deletion; close all windows associated with Lafite; and remove the mail watching process from the active process list. This is the same as invoking 'Quit' in the Lafite Status Window.

The third and subsequent arguments are options. The only one currently recognized is the atom *SHRINK*, which instructs Lafite to immediately shrink the browser window brought up on *MAILFOLDER*. In this case, if *MAILFOLDER* is (explicitly) *NIL*, it still defaults to *DEFAULTMAILFOLDERNAME*, since *SHRINK* makes no sense otherwise.

Lafite uses its own host and directory names for mail folders, *LAFITE.PROFILE*, etc., rather than the current connected directory because you may want to keep your mail folders someplace special (e.g., the local disk or your login directory), and the connected directory can change. The global variable *LAFITEDEFAULTHOST&DIR* is provided to tell Lafite where you generally keep your mail. *LAFITEDEFAULTHOST&DIR* should be atomic, in the same form as *LOGINHOST/DIR* (e.g. {*PHYLUM*}<*CARSTAIRS*>). If *LAFITEDEFAULTHOST&DIR* is *NIL*, then the value of *LOGINHOST/DIR* is used—i.e., your login host and directory.

If you are running in a Lisp in which some previous user has been using Lafite, you need to take some action to get Lafite to work on your mail files and in your name. When you change your user identity by calling (*LOGIN*) to log in as yourself, Lafite notices that the current user has changed, and attempts to authenticate the new user. However, Lafite still doesn't know how you want your Lafite customized; in particular, Lafite notices the value of *LAFITEDEFAULTHOST&DIR*, and loads your profile of known folders only when Lafite is first started. Thus to change the identity of the Lafite user, you should follow the following procedure:

- 1) Turn Lafite off, by clicking Quit in the status window, or calling (*LAFITE 'OFF*).
- 2) Log in as the new user by calling (*LOGIN*).
- 3) Set *LAFITEDEFAULTHOST&DIR* and/or *LOGINHOST/DIR* as appropriate, and any other personal variables that affect Lafite that matter to you. A typical way to do this step is to call (*GREET*) to get your personal initialization loaded.
- 4) Restart Lafite by calling (*LAFITE 'ON*).

## When things go wrong...

Lafite tries fairly hard not to break, but it is running in a very open environment, where users are free to interrupt arbitrary processes, network servers come and go, etc. As of this writing, the following are some abnormal states of note:

**Mail file does not parse.** This shows up when you browse a folder, Lafite starts parsing the folder, and encounters an error in the file, typically an incorrect message length. Lafite prints a message to this effect in the browser window and aborts the browse. The information Lafite prints includes the header of the last message parsed, and a byte pointer where the problem was encountered. The byte pointer is such that if you truncated the file to that pointer position, the mail file would be valid.

Solution: scavenge the mail file. The Library package *MAILSCAVENGE* contains a mail file scavenger. The Laurel MailFileScavenger program also works on Lafite files, as does the Hardy scavenger (I think).

The most common way to get a mail file into an inconsistent state is to abort a `MoveTo` or `Update` command somewhere in the middle (either manually, or because a remote server crashed). In the case of `MoveTo`, the problem is usually that the last message in the file is "too short", since it never was completely written. If the first operation you perform on the destination file after the crash is to browse it, Lafite will (usually) detect this situation and let you browse the file anyway, with a warning that the last message is truncated. Updating the file will then correct the length of the last message. Thus, in this case, you will not need a scavenger. If you neglect to browse the destination file before moving additional messages to it, however, you will need to scavenge.

**Table of contents inconsistent with mail file.** In theory, this should never happen; however, practice shows that it does. Lafite breaks with a message to this effect when it tries to operate on a message that isn't where it thought it was in the file. The appropriate action to take is to close the browser, selecting **Don't Update**, delete the table of contents file (the file with `-LAFITE-TOC` appended to the name), and then browse the file again. If this is not successful, you may need to scavenge the file. If you had made many changes to the browser (deletions, for example) that you would rather not lose, you can try selecting **Write out Changes Only** when you close the browser; this may succeed if the inconsistencies in the table of contents did not intersect with your changes.

**Slow file server.** If your mail files live on a remote file server that is particularly unresponsive, it may happen that the mail server connection over which new mail is being retrieved times out before the file server acknowledges receipt of the messages. The usual consequence of this is that your inbox is not flushed, so your new mail is in two places: your inbox, awaiting retrieval, and your mail file, to which it was just retrieved. A less common occurrence is that the mail server times out partway through the retrieval process, resulting in a Lisp break. You can `↑` out of the break to return to the state before the `GetMail` started.

If this is often a problem for you, you may want to adopt the following idiom, which several people have found useful in maintaining the flexibility of remote mail files while utilizing the speed and reliability of the local disk. Keep most of your mail files on the remote server, as usual, but keep your "active" mail file, the one to which you standardly retrieve mail, on your local disk. Retrieve mail to this file, and dispatch from there to your remote files (using `MoveTo`) some or all of the messages you wish to keep. Mail files on `{DSK}` have very predictable performance during `GetMail`, which is good for both you and the mail server. Files on `{DSK}` are also less subject to other vagaries of remote servers (e.g., sudden crashes) that sometimes cause problems with mail files. And if you tend to delete much of your incoming mail after reading it once, you will find it much faster to keep your active mail on `{DSK}`, even if your remote server isn't flaky.

You can also choose to keep most or all of your mail files on `{DSK}`, backing them up to a file server periodically. The `LispUsers` package `COPYFILES` is helpful for doing the backup automatically.

## Lafite Profile Variables

Below are the global variables that control Lafite's behavior.

### DEFAULTREGISTRY

The name of your local registry, used if your login name does not include a registry. This is normally set in the local site-specific `INIT.LISP` file.

### LAFITEDEFAULTHOST&DIR

The directory on which Lafite looks for `LAFITE.PROFILE` and all mail folders and message forms you access if you don't supply an explicit directory. See "**Changing the Lafite User**" above.

**DEFAULTMAILFOLDERNAME**

If no mail folder is supplied to the function LAFITE, i.e., you call (LAFITE 'ON), then the value of this variable is used. Initially, ACTIVE.MAIL.

**LAFITEMAIL.EXT**

The default extension for names of mail folders. Initially, MAIL.

**LAFITETOC.EXT**

The string appended to the name of a mail folder to produce the name of its table of contents file. Initially, -LAFITE-TOC.

**LAFITEFORM.EXT**

The default extension for names of user-defined form files. Initially, LAFITE-FORM.

**LAFITEFORMDIRECTORIES**

A search path for Lafite forms, initially NIL. When you choose the **Saved Form** command underneath **Send Mail**, the form name that you enter is first searched for on your default directory (LAFITEDEFAULTHOST&DIR), and if not found there, Lafite searches the directories in the list LAFITEFORMDIRECTORIES. LAFITEFORMDIRECTORIES is typically set to a list of one or more public directories on which generally useful forms have been collected.

**MAILWATCHWAITTIME**

The number of minutes between polling for new mail from your mail servers. Initially set to 5.

**LAFITEFLUSHMAILFLG**

If NIL, Lafite won't flush your inbox when retrieving new mail, so the mail will still be there when you invoke Get Mail again. Initially, T.

**LAFITENEWPAGEFLG**

If T, then the Hardcopy command will start each message on a new page. Otherwise it will separate each message by a line of dashes. Initially, T.

**LAFITEHARDCOPYBATCHFLG**

If you often request hardcopy of single messages, one at a time, you may notice some disadvantages: short messages sometimes get lost in amongst other users long output, and they are wasteful of paper. And if you hardcopy large messages, you may not always care to wait around while the message is formatted for hardcopy. LAFITEHARDCOPYBATCHFLG is provided to allow you to postpone the hardcopying until it can be done all at once.

When this flag is true, Lafite "batches" your hardcopy requests, and doesn't actually print them until you do an Update, at which point it sends them all to the printer in one batch. When you have hardcopy pending, the Hardcopy button is speckled to remind you of this fact. The Update button has an additional choice **Do Hardcopy Only** in case you want to get your batched hardcopy printed without doing an actual Update.

The behavior of LAFITENEWPAGEFLG when batching hardcopy is that it applies only to the messages selected at each Hardcopy invocation; each new set of messages starts on a new page, independent of the setting of LAFITENEWPAGEFLG.

LAFITEHARDCOPYBATCHFLG is initially NIL.

LAFITEHARDCOPY.MIN.TOC

If non-NIL, is a positive number. Whenever Lafite is instructed to produce hardcopy for more than LAFITEHARDCOPY.MIN.TOC messages, it also produces a table of contents as a cover page for the hardcopy. Currently, this flag is only noticed if LAFITEHARDCOPYBATCHFLG is NIL. Initially NIL.

LAFITEDISPLAYAFTERDELETEFLG

If T, Lafite will display the next message if you delete the one that is in the message display window and the next message is undeleted and has not been examined (i.e., it is marked with a "?"). If ALWAYS, then it will display the next undeleted message even if it has already been seen. Initially, T. T is roughly Laurel semantics, ALWAYS is Hardy semantics.

LAFITEMOVETOCONFIRMFLG

Controls whether Lafite requires confirmation of the Move To command. If ALWAYS, all moves require confirmation; if LEFT, then only left-button moves (selecting the destination from a menu) require confirmation; if MIDDLE, then only middle-button moves (using the "default Move To" folder) require confirmation; if NIL, then no moves require confirmation. Initially ALWAYS.

LAFITEBROWSERREGION, LAFITEDISPLAYREGION, LAFITEEDITORREGION

These are REGIONS which are used to describe where the primary (i.e. first) window of each type is to be placed on the screen. Initially, they are set to something "reasonable" for the standard initial display configuration. If you set them to NIL then you will be asked to specify a region (via GETREGION) the first time any such window is created.

LAFITEDISPLAYFONT, LAFITEEDITORFONT, LAFITEHARDCOPYFONT

These are the fonts used for displaying messages, composing messages, and making hardcopy. You may change them individually. They should be FONTDESCRIPTOR's as returned by FONTCREATE. Initially, they are all TimesRoman12.

LAFITEBROWSERFONT

The font used for displaying the table of contents in the browser window. Initially, Gacha10.

LAFITEMENUFONT

The font used for the items in all Lafite menus. Initially Helvetica10 Bold.

LAFITETITLEFONT

The font used for the title bar ("Lafite") in the Lafite status window. Initially Helvetica12 Bold.

LAFITESTATUSWINDOWPOSITION

Specifies where the status window appears when Lafite is invoked. It is a POSITION or NIL (in which case you will be asked to specify a position when Lafite starts).

LAFITEOUTBOXSIZE

Specifies the number of delivered messages that should be retained in your outbox. As you send more messages, older ones fall off the end. Increasing this number gives you a longer



"history" from which you can select and re-edit old messages, but this desire should be balanced with the knowledge that you are tying down the resources used by each of those messages. Setting `LAFITEOUTBOXSIZE` to zero or `NIL` disables the outbox feature: after delivery, messages completely vanish. `LAFITEOUTBOXSIZE` is initially 2.

`LAFITENEWMAILTUNE`, `LAFITEGETMAILTUNE`

(Dandelion only) These are lists of the form acceptable to the function `PLAYTUNE`, or `NIL`, in which case they are ignored. `LAFITENEWMAILTUNE` is played when Lafite discovers you have new mail waiting; `LAFITEGETMAILTUNE` is played when a Get Mail command completes.

`LAFITEENDOFMESSAGESTR`, `LAFITEENDOFMESSAGEFONT`

`LAFITEENDOFMESSAGESTR` is a string containing the text of the "End of Message" token displayed at the end of a message; `LAFITEENDOFMESSAGEFONT` is the font in which it is displayed. If `LAFITEENDOFMESSAGESTR` is `NIL`, then no "End of Message" token will appear.

`LAFITEIFFROMMETHENSEENFLG`

If true, then messages sent from you are considered "Seen" (and hence do not have the mark '?'), even though you have not yet displayed them. Initially `T`.

`LAFITEBUFFERSIZE`

The number of 512-character buffers used by the stream managing the file behind an open browser window. If you regularly receive very long messages, you might want to increase this to improve performance of Display followed by HardCopy or Move. Initially 20, which handles up to about 10,000-character messages.

`LAFITEMODEDEFAULT`

Determines the mode Lafite comes up in when more than one set of protocol implementations has been loaded. Initially `NIL`, which means that if there is a choice, Lafite simply waits for the user to choose a mode.

`LAFITESHOWMODEFLG`

Determines whether Lafite displays the current mode in the status window. If `ALWAYS`, it always does; if `NIL`, it never does; if `T`, it does only if there is any ambiguity (more than one set of protocol implementations has been loaded). Initially, `T`.

## Adding New Message Forms to Lafite

The normal way to add new message forms to Lafite is to edit an existing form (or build one from scratch) and save it away using the 'Save Form' menu item. You can also provide message forms that compute the text on the fly, as, for example, the 'Lisp Support' selection does. To add your own items to the 'Message Forms' menu, add a standard three-element menu item to the variable `LAFITESPECIALFORMS` and then set the variable `LAFITEFORMSMENU` to `NIL` (this is where the menu is cached). The three-element menu item should yield a `LITATOM` as its "value", that atom being interpreted as follows:

1. If the atom has a function definition, the function is called (with no arguments) and the returned value (a string or a TEdit TextStream) is used;
2. If the atom has a value, its value (a string or a TEdit TextStream) is used;

3. otherwise, a copy of the file by that name is used.

For example, if TEdit wanted to add a message form that contained the date the TEdit was made (similar to Lafite's bug report form) it could add

```
("TEdit Support" (QUOTE MAKETEDITFORM)
 "Make a form to report a problem with TEdit")
```

to LAFITESPECIALFORMS; MAKETEDITFORM could be defined to be

```
[LAMBDA NIL
 (PROG (OUTSTREAM)
 (SETQ OUTSTREAM (OPENTEXTSTREAM ""))
 (printout OUTSTREAM "Subject: TEdit: >>Subject<<" T)
 (printout OUTSTREAM "To: " TEDITSUPPORT T)
 (printout OUTSTREAM "cc: " (USERNAME) T)
 (printout OUTSTREAM "TEdit-System-Date: " TEDITSYSTEMDATE T T)
 (printout OUTSTREAM ">>Message<<" T)
 (RETURN OUTSTREAM])
```

where TEDITSUPPORT and TEDITSYSTEMDATE are variables set by TEdit. Lafite supplies one function to make this kind of message form easier to construct:

```
(MAKEXXXSUPPORTFORM SYSTEMNAME ADDRESS SYSTEMDATE)
```

Creates a message form (a TEdit stream) to be mailed to the maintainers of SYSTEMNAME. SYSTEMNAME is the name of the subsystem (a string); SYSTEMDATE, if non-NIL, is a date (string) of importance to include in the message; and ADDRESS is the mail system address of the intended recipient(s).

For example, if MAKETEDITFORM were defined as

```
[LAMBDA NIL
 (MAKEXXXSUPPORTFORM "TEdit" "TEditSupport" TEDITSYSTEMDATE)]
```

Then selecting "TEdit Support" in the Message Forms menu would produce a form such as the following:

```
Subject: TEdit: >>Subject<<
To: TEditSupport
cc: vanMelle.pa
TEdit-System-Date: 3-Feb-84 12:23:49
Lisp-System-Date: 3-Feb-84 18:13:22
Machine-Type: Dorado

>>Message<<
```



There is a new and substantially revised version of Lafite on [Phylum]<Lisp>Library>Lafite.dcom and in [Phylum]<Lisp>Current>Full.SYSOUT. If you want to load the new Lafite on top of an old Lafite, please Quit the old Lafite first.

The changes in this version are mostly in the internal implementation, rather than the user interface, but there are a few user-visible changes. The major such change is that Lafite no longer attempts to keep your mail browser and the mail file itself consistent with each other at all times. In particular, changes (deletions, marks changed) only get written out to the file when you do an Update, and "MoveTo"s are only guaranteed to have finished then, too. To make this more feasible, there are now two flavors of update: "Update by writing out changes only" and "Expunge deleted messages". The latter is like the old Update, the former can be thought of like a SAVEVM: it does not compact out deleted messages, but does write out changed marks and an updated contents file, assuring that if you booted your machine after the Update that you could browse the mail file again and get back to your previous state.

When you close or shrink a browser that has changed in any way needing update, you get a menu of choices--the two flavors of update, or no update at all.

The value of the variable LAFITESTATUSWINDOWPOSITION is now the lower left corner of the entire window, as you would expect, rather than the lower left corner of the "You have new mail" part of it.

The variable LAFITEUSEHIGHESTVERSIONFLG has gone away. Lafite always reads the highest extent version of a file, and Update goes to that version. That version is, of course, ;1 if you are using Laurel on the same disk.

There is a completely new table of contents format, which is more compact and allows a more efficient update, one that is only proportional to the number of changed messages, rather than to the total number of messages. If you have an old table of contents for a file you browse with the new Lafite, it will be discarded.

Assorted bugs have been fixed, including some relating to resuming over Logout or SaveVm, or clicking GetMail too soon. I would be surprised if I'd caught all such bugs, though. If you have some favorite old bug (as opposed to suggestion) that hasn't been fixed, you might want to retransmit a bug report for it.

Many internal data structures have been revised; some algorithms that took time proportional to the number of messages in a file are now constant-time algorithms; there is a completely new message header parser. If you rely on any undocumented features of Lafite, let me know, as they may well have changed.

If you are a Laurel 6.1 user, you may find that Laurel has transformed some spaces in your messages from character 40Q to 240Q as some sort of line-breaking hack. There is a little function COPY7BITFILE that can be used to copy such a file removing the high-order bits. Args are SRCFIL and DSTFIL, with DSTFIL defaulting to a new version of SRCFIL.

There will be updated documentation some day soon.

Complaints and suggestions to LafiteSupport as usual.

## **Changes to Lafite in the version of January, 1990.**

**New NS Mail implementation.** There is a new module called NEWNSMAIL, which implements the new mail protocols supported in Services 11.2. This module must be loaded in addition to (and after) NSMAIL. From the user's point of view the two implementations are similar, except that with the new protocols you can retrieve parts of messages not available with the old protocols (such messages when retrieved with the old protocols contain a notation "NOTE: Additional parts of this message are available only via the new Mailing Protocols.", and the Lafite status window subsequently over-reports the amount of new mail you have by the number of such messages).

Whether you use the new protocols is controlled by the variable \*USE-NEW-NSMAIL\*: if true use the new protocols, if NIL use the old. After changing this variable, you must Recache (under middle-button Quit) or restart Lafite in order for it to notice. The variable is initially T.

If any of your mailbox servers does not support the new protocols, Lafite sets \*USE-NEW-NSMAIL\* to NIL and starts over. All the servers at PARC support the new protocols.

Changes to Lafite since October, 1988.

There is an additional command on the message composition window: **Change Mode**. This lets you change the mode of a message from GV to NS or vice versa. This can be useful if you clicked **SendMail** while in the wrong mode, or you want to Answer a message but do so from the other side of the fence. This command does not make any attempt to fix addresses already in the header to their other form; you have to do this yourself. The one exception is that it will change "cc: *Your old-mode name*" to "cc: *Your new-mode name*".

There is a new command on the middle-button titlebar menu of the browser: **Cancel Pending Hardcopy**. Of course, this is only interesting when LAFITEHARDCOPYBATCHFLG is T.

You can now send to private dl's in NS mode as well as GV mode. The syntax is the same: "To: dlnam;;". In NS, this parses as the NS name "dlnam;;Xerox" (and will look like that to the recipients), but Lafite recognizes this kind of address and instead looks for a file named dlnam.ext on your Lafite directory.

Changes to Lafite during February, 1989.

There is a new kind of filter for NS recipients of GV messages. If you include the symbol GV in the list LAFITE.DONT.DISPLAY.HEADERS, then messages sent from GV to NS will have their ugly "GVGVGVGV..." section hidden (this section includes any headers that didn't translate into NS land, such as comments in addresses ("Fred Carstairs <fc@foo.bar.com>") and all those awful "Received:" lines in arpa mail. As usual, the Unhide command on the window will reveal the truth. You can also add GV to LAFITE.DONT.FORWARD.HEADERS and LAFITE.DONT.HARDCOPY.HEADERS.

Lafite canonicalizes your LAFITEDEFAULTHOST&DIR. This should reduce confusion among users living on NS servers, who might have set the variable to, say, "{EG:}<Carstairs>Mail>", only to have Lafite confused when file names came back in the form "{EG:PARC:Xerox}..."

Lafite handles the form files menu better.

The "Reply-To" and "Send Formatted?" menus no longer step on each other if you deliver two messages requiring them at the same time.

MAKEXXXSUPPORTFORM, when given an address in a-list form, chooses the first address in the list that is for a supported mode. Thus, we can arrange to have LISPSUPPORT always go to an NS address (assuming NS mail loaded). Special forms that call MAKEXXXSUPPORTFORM can also encourage a particular mode this way without having to switch the mode themselves.

NS mail no longer holds a session open on the mail server in certain odd cases.

NS mail header fields are displayed in the order specified by the variable NSMAIL.HEADER.ORDER, independent of the order in which the mail system may have delivered them (the new Services 11 servers deliver some headers in an odd order).

## Changes to Lafite in the version of May, 1989.

**New files.** There are 3 new files: LAFITEFOLDERS, LAFITESORT, MAILSCAVENGE. In addition, Lafite requires the file DATEPATCH for improved date parsing. These are all loaded automatically by LAFITE.

**Folder hierarchy.** [This feature derives from a package written by Mike Dixon.] You can now organize your folders into a hierarchy, so that the folder menus presented by **Browse** and **MoveTo** are easier to use. Folders are organized into *groups*. A folder can be in any number of groups. Groups can have subgroups. The top level of the folder menu is composed of all the groups that are not a subgroup of some other group, plus all folders that are not in any group. The **Edit Folder Hierarchy** command on middle-button Browse lets you define new groups and change existing ones.

Note: the folder hierarchy is saved on your Lafite.info file, along with folder names and form names. Older versions of Lafite do not know about this structure, but those of the last year will at least preserve it. However, if you delete folders and then come back to a new version of Lafite, the deleted folders may still appear in the hierarchy; you must manually delete them either by editing the variable LAFITE.FOLDER.STRUCTURE or by editing the file Lafite.info.

**Sorting.** There is a new command on the middle-button browser titlebar menu: **Sort by Date**. This command sorts the messages in the browser in order of their "Date:" fields. Messages lacking a Date field or having an unparseable Date field are sorted so as to remain next to the preceding message. There is a subcommand **Sort Selected Range**, which only sorts between the first and last selected messages. In either case, the messages are sorted in the browser only, until you next **Update**, at which time the messages are written to the file in the sorted order. When messages have been rearranged, the **Update** option **Write Out Changes Only** is not available, as there is no file format for out of order messages; you must **Expunge**, even if there are no deleted messages.

**Sorting New Mail.** You can have Lafite automatically sort new mail by setting the variable LAFITE.SORT.NEW.MAIL. If the value is T, Lafite always sorts new mail; if :MULTIPLE, Lafite only sorts when the mail was retrieved from more than one server. Note that even when using a single mail server, mail can easily be out of order, due to the varying speeds at which messages pass through gateways, etc. Sorting assures that the messages will appear in the order in which they were posted. Sorting new mail seems to be fast enough (at least on my Dorado) that it is probably always worth doing.

**Dates.** As you might guess from the above, Lafite now actually parses the Date field of messages. Aside from enabling Sort, this allows Lafite to display the date in the browser in a canonical form, rather than one that varies with the whim of the sender's software. Lisp's date parser (IDATE) was substantially beefed up to handle a wider variety of dates. It now claims to handle all dates legal in RFC822 syntax (except the silly single-digit military time zones), plus a fair variety of other formats found in mailers of recent years. The parser changes are in a module DATEPATCH loadable separately from Lafite.

Lafite considers dates older than 1970 spurious, so as to avoid being fooled by messages from machines that neglected to set the time.

With a new representation of dates, Lafite also has a new table of contents format. If you browse a folder with an old-style toc, Lafite will print "(older format)" when it reads it. Next time you do an Update, the toc will be written in the new format. Lafite does not automatically parse dates on old messages, so until you issue a Sort command, old message dates will still be displayed as whatever string the old toc saved, not the new canonical form. In addition, when you *do* issue the Sort command, it must first go back and retrieve afresh all the date fields and parse them. Old versions of Lafite cannot read the toc files saved by this new version of Lafite, so if you try to browse a new folder with an old Lafite it will be forced to parse the file from scratch.

**Scavenger.** The mail scavenger has been completely rewritten. It now handles most simple cases of bad message lengths by altering the file in place, rather than laboriously copying the entire file to a scratch location. The diagnostic output is, I hope, more informative. The scavenger is also integrated with the



browser, so that if you browse a malformed folder and the browser reports "unable to parse", you are immediately offered the option of scavenging.

**Expunge.** Lafite now gets less confused if you abort it in the middle of an Expunge. Either you killed it before it got to the point of irrevocably altering the file, in which case it is as if you had never started the Expunge, or it will report that the file is inconsistent and must be rebrowsed.

**Long messages.** The Lafite message file format has been very slightly extended to permit you to receive messages of up to 99,999,999 bytes (about 100MB!) in length. This means it no longer is forced to split apart messages longer than 99,999 bytes, which did very bad things to large Viewpoint attachments and sometimes caused loss of mail server connection. It also means that Lafite files are no longer exactly compatible with Laurel or Hardy format, in case anyone cares. As mitigation, however, messages that fit in the old format (99,999 bytes or less) are converted back to the old format whenever they are moved (either by **MoveTo** or **Expunge**), so you can arrange for a Hardy-compatible file by moving all messages into another folder (assuming your messages are all short enough).

**NS Mail.** Lafite accepts messages sent in serialized format 3, which it turns out is identical to format 2.

Lafite now sends plain text messages as "text attachments" rather than "mail notes", unless \*NSMAIL-SEND-MAIL-NOTES\* is true. This avoids a nasty bug in Services 11.3 mail servers.

An incomplete and not very well tested feature: you can send attachments. The attachment must be a file on an NS file server. To send an attachment, place a line "Attached-File: *XNS Filename*" in the header of your message. To send the file as a reference (this sends only a pointer to the file, not the file's content, so the recipient must have access to your server), use the line "Attached-Reference: *XNS Filename*". You can have only one such line in the header, and the body of the message must be small enough to send as a mail note (less than 8000 characters). Probably the only interesting kind of attachment to send currently is an Interpress master, which both Lafite and Viewpoint recipients will be able to print. To "forward" an attachment you received in NS mail, you can use **Put to File** on the attachment, then compose a message with "Attached-Reference" or "Attached-File" referring to the resulting file. Incompleteness: Most of the attributes of the file are lost when you send the file itself as an attachment, rather than a reference. You cannot send directories as attachments.

## **Changes to Lafite in the version of August, 1989.**

The major changes in this version all affect NS mail:

You can now send NS mail even if you don't have an NS mailbox, as long as you have an NS identity. You must be sure you have forwarding from this NS identity, however, since otherwise there is no place for the mail system to send failure replies.

The system automatically detects when your mailbox has been moved from one server to another and adjusts accordingly, instead of reporting the server's incomprehensible error message "ACCESS.ERROR NoSuchRecipients".

The names in the To and Cc fields of NS messages are now abbreviated where possible. For example, if the sender is Burwell:PARC:Xerox, then recipient Carstairs:PARC:Xerox is displayed as "Carstairs", and ClockWatchers:All Areas:Xerox as "ClockWatchers:All Areas".

The "View as Text" command on NS attachments is slightly smarter about filtering out binary garbage, and recognizing the Viewpoint end of line character. The window that "View as Text" uses is automatically closed when you display the next message or shrink/close the folder (if you want to retain it, you have to use the middle button to display the message containing the attachment).

Changes to Lafite since July 19, 1988.

The browser middle-button title-bar menu has three new commands (in addition to the Find commands): Go to # lets you jump to a message by number; Describe Folder prints some informative information about the folder; and Enable Move To Menu attaches a menu to your browser window to let you move messages quickly to other folders. Setting LAFITE.AUTO.MOVE.MENU true lets you get this accelerated menu automatically.

Middle-button on an "active" browser's icon now only offers the "Get Mail" menu while the button is held down; if you don't select "Get Mail", it simply expands the window. This means that middle-button on that icon is more like every other icon.

The Browse subcommand "Forget Folders" lets you select lots of folders at once, somewhat more gracefully than the old way.

There is a new Browse subcommand Rename Folder.

Host abbreviations: if you have mail files on multiple directories, there is now an abbreviation facility. See documentation of LAFITE.HOST.ABBREVS.

The bug wherein Lafite forgot LAFITEDEFAULTHOST&DIR (and other "Personal" variables) over logout is (I hope) fixed.

Bug reports include the sysout name in addition to its date.

## Changes to Lafite in the version of November, 1989.

**New Looks Subcommands.** On the Middle-button menu in a Lafite message display window, the Looks command has, in addition to **Default** and **Fixed Width**, three new subcommands:

**Lowercase** Converts message body to lowercase. This is useful for messages from those bozos who haven't discovered their Shift Lock yet, or are typing on Model 33 Teletypes.

**Spread Paragraphs** Inserts 10-point leading between paragraphs. This is useful for messages that come from Tioga, which doesn't put a blank line between paragraphs in the plain text body.

**VP Line Breaks** This turns the character #o35, which at least some mail senders in the XNS world seem to use as an end-of-line character, into carriage return. Maybe some Viewpoint expert could fill me in.

**Saved Forms.** The **Save Form** button on a message sending window is now sanctioned to save (checkpoint) messages in progress, as most of you seem to want to do much more than create private mail forms. In light of this policy shift, the button has been renamed simply **Save**, and the message window stays open afterwards (close it as you wish). When a previously saved message is ultimately delivered, you are asked whether you want to keep the saved form or delete it. There is also a menu item **Delete Message Form** underneath middle-button **Browse** for deleting a saved message explicitly. You can think of **Save** as a fancy version of TEdit's Put, which stores the message in your mail directory and remembers it (in the Saved Form menu) for a later Get.

**Reply To.** There is an additional button labeled **Reply To** on message sending windows. It inserts a "Reply-to: *yourname*" field in the message being composed. The name is selected for pending delete, so if you want replies to go to someone else, you can just start typing the new name.

**Rename Folder** now has a better prompt. It also takes no action if you just pressed carriage return, thus returning the old name.

**Bad toc.** Fixed error message when you read a malformed toc (previously it broke). Several people have run into this problem by copying toc files between Sun and NS file servers on Sun Medley, which erroneously called the toc files type TEXT.

**Scavenger.** The programmatic entry to the mail scavenger has been renamed LAFITE.SCAVENGE, to avoid confusing people who knew the old argument list. Ordinarily, you should never have to call the scavenger by hand, as the **Browse** command will invoke it for you when needed.

I assume you are willing to deal with a completely unpolished interface, so I'll tell you what the current truth is about the Lafite on <LispCore>Library> and in the latest <LispCore>Next>Full.sysout. What follows might be viewed as the start of an implementor's manual. It is not, of course, anything I'd be willing to publish as a programmer's manual, but perhaps your experience will help us define what one might wish the interface really to be. I'm quite willing to add small things as you see the need, though there is little hope for any major projects in the near term. So let me know how this matches your needs.

My other reluctance, of course, is that internals, as these all are, are subject to change without notice, so I'd appreciate feedback on what parts you would like to see solidified into a supported interface.

There are almost no direct functional interfaces to what you want, but with some record definitions I think you can get most of it. You thus need to LOADCOMP(LAFITE). In general, you should probably avoid replacing record fields, as there are usually interactions; if you need to replace a field for which there is no programmatic interface, let me know. Fetching fields should be fine. The important records are two datatypes, MAILFOLDER and LAFITEMSG.

MAILFOLDER holds assorted facts and state about a single mail folder. The window property MAILFOLDER hangs off each Lafite browser window. Here are the most interesting fields:

FULLFOLDERNAME, VERSIONLESSFOLDERNAME, SHORTFOLDERNAME -- various forms of the folder's name.

FOLDERSTREAM -- a stream on the file behind the folder, if it's open.

MESSAGEDESCRIPTORS -- an array of LAFITEMSG objects, corresponding to messages 1 thru #OFMESSAGES.

Thus (ELT (fetch (MAILFOLDER MESSAGEDESCRIPTORS) of folder) n) returns the nth LAFITEMSG in folder. This field is only valid while a browser is open on the folder.

FOLDERLOCK -- a monitor lock that should be acquired if you want to perform any operations on the folder.

FIRSTSELECTEDMESSAGE, LASTSELECTEDMESSAGE -- number of the first and last messages currently selected (equal if only one selected).

BROWSERWINDOW, BROWSERMENU, BROWSERMENUWINDOW, BROWSERPROMPTWINDOW -- pointers to parts of the browser window for this folder. NIL if no browser open on folder.

(\LAFITE.GETMAILFOLDER folderName) -- this function returns a MAILFOLDER object for folderName, which should be either a full file name or a full name sans version number. If the result has a non-NIL BROWSERWINDOW field, there is a browser currently open on the folder, else not.

(\LAFITE.OPEN.FOLDER folder access recog) -- returns a stream on the file behind folder. access and recog are as with OPENSTREAM. This is the standard way to get the stream (rather than the FOLDERSTREAM field), unless you're sure the file's open for the right access.

Each message in a browser is described by a LAFITEMSG record. Its most interesting fields:

PARSED? -- true if message has been parsed. Set by LAFITE.PARSE.MESSAGE.FOR.TOC, which also distributes the standard parse into the fields FROM, SUBJECT, DATE.

DELETED? -- true if message is marked deleted.

SEEN? -- true if message is marked seen (NIL => "?" in browser).

MARKCHAR -- character code of message's mark.

MARKSCHANGED? -- true if MARKCHAR has changed since last update.

SELECTED? -- true if message is currently selected.

MSGFROMMEP -- true if message is from the logged in user.

# -- ordinal position of message in folder, from 1.

BEGIN -- byte position in file of start of message, which is always the internal header that starts "\*start\*".

MESSAGELENGTH -- length of message, including internal header.

STAMPLENGTH -- number of bytes between the BEGIN position and actual start of message.  
 START -- access field: byte position of the start of the actual message, viz., BEGIN+STAMPLENGTH.  
 END -- access field: byte position of the end of the message, viz., BEGIN+MESSAGELENGTH.  
 DATE -- date of message, actually just the 6 character string that appears in toc (so you can't give it to IDATE).  
 FROM, SUBJECT, TO -- From, Subject, To fields of the message. The TO field is usually only parsed out if MSGFROMMEP has been observed to be true.

Now, how to use all these:

The value of the variable LAFITE.AFTER.GETMAIL.FN, if non-NIL, is a function to call immediately after Lafite retrieves mail from a server. It is called with two arguments, the MAILFOLDER, and a list of LAFITEMSG objects describing the new messages (which have already been appended to MAILFOLDER:MESSAGEDESCRIPTORS, securely stored in the file, but not yet displayed in the browser window). The messages have not yet been parsed, I believe. The function is called while in control of MAILFOLDER's monitorlock.

Parsing messages. The function LAFITE.PARSE.MESSAGE.FOR.TOC sets all the fields in a LAFITEMSG that the toc needs, but you probably want more fields than that, so will need to do your own parsing. Lafite has a simple-minded but reasonably efficient parser that looks for header fields in a message (Field name followed by colon) and does something with them. It requires that you build a "parse table" describing what you want it to do. This is a list of lists, one for each field you want to parse. Each sublist is in the form ("FieldString" ResultFn . ResultArgs). Then "compile" the parse table by calling (LAFITE.MAKE.PARSE.TABLE table). The resulting compiled table can then be passed to this function:

(LAFITE.PARSE.HEADER stream parseTable start end onceOnly) -- Stream is the stream where the message lives -- the folder's stream for message in a file, or a tedit stream for a message you're sending. parseTable is the compiled parse table. Start and End are byte pointers in stream (normally the START and END fields of a LAFITEMSG). Parsing automatically stops at the end of the header. onceOnly is true if you want parsing to end as soon as any one field has been successfully parsed.

The parse proceeds as follows: the parser starts at byte position start and scans each line in the header, seeing if the field name matches (case-insensitively) a field in the parse table. If a match is found (the characters match up thru the colon), then the parser skips over any white space, then calls the ResultFn indicated in the parse table with two arguments, the stream (now positioned at the first non-blank character after the colon) and the ResultArgs from the table (this lets you multiplex one parsing function for several purposes). The function's job is to parse the rest of the field, leaving the stream positioned after the terminating <eol>, and to set freely the variable PARSERESULT as it wishes. PARSERESULT is what LAFITE.PARSE.HEADER ultimately returns.

You may not need to follow that too closely. Lafite's standard use of the parser is to make PARSERESULT an alist; the following functions are interesting as resultFn's, or to help them. You can look at the variable LA.FULLPARSEFIELDS and LA.TOCFIELDS for examples. The former is used for a "full parse", such as the Answer command needs; the latter parse three fields out for the toc.

(LAFITE.READ.TO.EOL stream) -- Reads the rest of the field, returning one long string. You should probably use this as the basis for all result fns, as it takes care of continuation lines and makes sure the stream is left properly at the end of the field.

(LAFITE.READ.LINE.FOR.TOC stream Args) -- Reads the field as one big string and stashes it on PARSERESULT indexed by (CAR Args). If all of your resultFn's are this one, then your parse result ends up something like ((field1 "contents1")(field2 "contents2")...). The string is limited in length to 255 characters (a toc restriction); if longer, it is truncated.

(LAFITE.READ.NAME.FIELD stream Args) -- This is almost the same, but no string length restriction, and it recognizes that some fields may have multiple occurrences (bad form, but allowable). If a field occurs twice, its alist entry has more than one string in its cdr: ((field1 "contents1a" "contents1b") ...).

(\GV.PARSERECIPIENTS field registry internalFlg editWindow) -- This function takes raw strings from the output of LAFITE.PARSE.HEADER and produces a list of individual recipient names. Field is a string or list of strings, such as LAFITE.READ.NAME.FIELD might produce during the header parse. registry is the grapevine registry default for names without registries (defaults to DEFAULTREGISTRY). If internalFlg is true, the names returned are in internal form (name . reg); if NIL, the names are strings of the form "name.reg". editWindow is for error messages (true when sending mail).

(LA.SETDIFFERENCE x y) -- like LDIFFERENCE, but treats x and y specifically as sets of strings, case-insensitively.

(LA.REMOVEDUPLICATES x) -- removes duplicates from x, treating x as a set of strings, case-insensitively.

Message manipulation. Most of these update the browser window appropriately.

(SELECTMESSAGE msgDescriptor mailFolder) -- "selects" msgDescriptor, a LAFITEMSG object, in mailFolder. Does not deselect anything.

(UNSELECTALLMESSAGES mailFolder) -- unselects all messages.

(MARKMESSAGE msgDescriptor mailFolder mark) -- changes msgDescriptor's mark character to mark (a charcode).

(SEENMESSAGE msgDescriptor mailFolder) -- mark msgDescriptor "seen".

(DELETEMESSAGE msgDescriptor mailFolder) -- mark msgDescriptor "deleted".

(UNDELETEMESSAGE msgDescriptor mailFolder) -- undelete msgDescriptor.

Most of the browser commands consist of a function \LAFITE.nameOfCommand, some of which spawn processes named \LAFITE.nameOfCommand.PROC. Here's the one for MoveTo:

(\LAFITE.MOVETO.PROC window mailFolder destinationFolder item menu shortOutput oldFileP) -- moves all selected messages from mailFolder to destinationFolder. Greys out item of menu while it's working (the BROWSERMENU field of mailFolder gets you the menu). When finished, if exactly one message was moved, and it was the one currently displayed, does the "display after delete" action. shortOutput is the folder's name (in short form, but it really doesn't matter). oldFileP is true if the caller believes, but has not yet verified, that destinationFolder describes an existing folder (this is true if the folder was on the folder menu). window is mailFolder's BROWSERWINDOW.

As I said, "unpolished". You probably really want the not yet defined subfunction of that:  
(\LAFITE.MOVE.MESSAGES mailFolder msgDescriptors destinationFolder).

---

---

## LAFITE UPDATE

---

---

This document describes changes to Lafite since the October, 1986 release.

### **Lafite Modes**

---

Lafite is now willing to operate in more than one mode "simultaneously". The mail watch background task checks all of your mailboxes in all the modes that are active, and **GetMail** retrieves mail from all mailboxes. The variable `LAFITE.USE.ALL.MODES` controls whether Lafite runs in more than one mode at once. The interesting values are:

- NIL This is the old way, where you must change modes manually. All commands operate only in the currently active mode.
- : POLL The mail watch task checks all modes, but **GetMail** only retrieves in the currently active mode.
- T Mail watch checks and **GetMail** retrieves in all modes.

The initial value is T. The flag has no effect on the **SendMail** command, which continues to bring up a message composition window in the current mode. It also does not affect the **Answer** command, which answers in the mode that the message was retrieved in. The retrieval mode is remembered in the table of contents file; in the absence of this knowledge, e.g., if the toc is deleted or the message was retrieved with an older Lafite, the **Answer** command uses simple rules that nearly always unambiguously distinguish GV from NS messages. Be aware that the value :POLL can be confusing to such hacks as automatic mail retrievers, since the status window can report new mail, yet **GetMail** won't retrieve it.

Ordinarily, Lafite only attempts to authenticate you in each mode when it starts up, and after logout. If, for example, the NS authentication service is down when you start up Lafite, or your NS password is incorrect, it does not repeatedly try to authenticate you, but merely reports the condition once and behaves as if the failing mode does not exist. If conditions later change, you can force another attempt by using the mode switching commands to explicitly switch to the mode. If your password is incorrect, you can change it using the NS Login subcommand of Quit (see below).

**GetMail** only attempts to retrieve mail from the servers that reported new mail in the most recent poll. This means that if one of your mail servers is not responding, you no longer have to wait during **GetMail** for it to time out. If you click **GetMail** at a time when the status window reports "No New Mail", then **GetMail** will attempt to contact all of your mailbox servers, just as it did in previous releases.



## Turning Lafite on and off

You can now turn Lafite on or switch modes from the background menu.



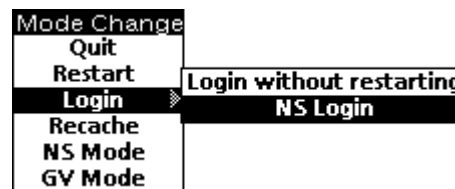
**Mail** This brings up a standard message sending window in the current mode, whether Lafite is on or not. If Lafite is not on, it will attempt to authenticate you in the current mode.

**Send Mail** This is the same as the main **Mail** command, except that you get to choose which kind of message form, just as middle-button on the Lafite status window's **Send Mail** command allows.

**Turn Lafite On** This is the same as calling (LAFITE :ON). It has no effect if Lafite is already on.

**Set Lafite Mode** This command allows you to change Lafite's mode. It offers a menu of the mode choices found also as part of the middle-button **Quit** command on the Lafite status window.

The middle-button **Quit** command on the Lafite status window has two new subcommands, **Recache** and **Login**. Here, for completeness, is a description of all the commands:



**Quit** This is the same as the main command, which is the same as calling (LAFITE :OFF). It closes all folders, asking about updating them if any need it, then turns off Lafite.

**Restart** This turns Lafite off, then immediately back on; same as (LAFITE :RESTART).

**Login** Changes the logged-in user for Lafite. This command first turns Lafite off, so that any updating required by closing out the previous user's folders happens under the previous user's login. It then prompts for user name and password, exactly as if you had called (LOGIN) yourself, and then for a new LAFITEDEFAULTHOST&DIR. Finally, it turns Lafite back on.

**Login without restarting** Performs a login without changing the state of Lafite. This is useful if you had previously logged in with an incorrect password, or if you have multiple user identities all sharing the same set of folders.

**NS Login** Prompts for a new login just for NS mail. You may need to use this if your global user name does not correspond to your NS user name or an alias of it in the default domain, or if you have different passwords for Grapevine and NS. There is actually

one of these commands for each mode that understands an independent login, but NS is the only current instantiation.

**Recache** In the case of several of the public interface variables, Lafite computes other structures based on them, and does not automatically recompute them when you change the public variable. The recache command forces Lafite to recompute these "caches" of the variables' values. A recache also occurs whenever Lafite is (re)started. The command is implemented by the function `LAFITE.COMPUTE.CACHED.VARS`, which you can advise if you have other caches that Lafite doesn't know about. The variables currently affected by this command (i.e., changes in whose values are not otherwise noticed) are:

```
LAFITE.DONT.DISPLAY.HEADERS
LAFITE.DONT.FORWARD.HEADERS
LAFITE.DONT.HARDCOPY.HEADERS
LAFITE.EXTRA.DISPLAY.COMMANDS
LAFITE.LOOKS.SUBCOMMANDS
LAFITE.GV.FROM.FIELD.
```

**NS Mode**

**GV Mode** Switch to the specified mode. There is one of these for each mode.

## Screen Appearance

---

There are several new variables and one function for controlling the appearance of your Lafite screen.

`LAFITE.BROWSER.LAYOUTS` [Variable]

A list of browser "layouts" specifying where Lafite should place browsers, their display windows and their icons. Each layout is of the form (*BrowserRegion IconPosition DisplayRegion*). *IconPosition* and *DisplayRegion* are optional. The region variables are standard window system regions of the form (*left bottom width height*); *IconPosition* is a position of the form (*x . y*). When the **Browse** command creates a new browser window, it chooses the first element of `LAFITE.BROWSER.LAYOUTS` that is not already in use. If all are in use, it prompts for a region.

`LAFITE.DISPLAY.SIZE` [Variable]

Specifies the default size of display windows, other than those already specified in a browser layout. If the value of this variable is a size (*width . height*), then Lafite prompts for a region with a box this size. If the value is `NIL`, Lafite prompts for a general region, requiring you to drag from one corner to the opposite. The initial value is `(500 . 300)`.

`LAFITE.EDITOR.LAYOUTS` [Variable]

A list of editor "layouts" specifying where Lafite should place message composition editors and their icons. As message editors have only a single window, each layout consists of only two elements (*EditorRegion IconPosition*). When one of the message-sending commands creates a new editor window, it chooses the first element of `LAFITE.EDITOR.LAYOUTS` that is not already in use. If all are in use, it prompts for a region.

LAFITE.EDITOR.SIZE

[Variable]

Specifies the default size of message composition editors, other than those already specified in LAFITE.EDITOR.LAYOUTS. If the value of this variable is a size (*width . height*), then Lafite prompts for a region with a box this size. If the value is NIL, Lafite prompts for a general region, requiring you to drag from one corner to the opposite. The initial value is (470 . 300).

For backward compatibility, if LAFITE.BROWSER.LAYOUTS is NIL, it is considered to contain the single element (LIST LAFITEBROWSERREGION NIL LAFITEDISPLAYREGION). If LAFITE.EDITOR.LAYOUTS is NIL, it is considered to contain the single element (LIST LAFITEEDITORREGION). The variables LAFITEBROWSERREGION, LAFITEDISPLAYREGION, and LAFITEEDITORREGION are now obsolete and may be removed in a future release.

(LAFITE.BROWSE.FOLDER *foldername layout options* —)

[Function]

Programmatic interface to the **Browse** command. Browses the mail folder named *foldername* and returns a mailfolder object. If a browser already exists for *foldername* it is returned; otherwise, a new browser is created. The normal Lafite defaulting rules apply to *foldername*, so, for example, "active" defaults to the file "active.mail" on the directory specified by LAFITEDEFAULTHOST&DIR. The argument *layout* is a list of up to three elements describing the layout of the browser; it takes the same form as elements of LAFITE.BROWSER.LAYOUTS. *options* is a list of zero or more keywords from among the following:

- :ACTIVE The browser is considered an "active" mail browser. Currently the only import of this setting is to enable GetMail on expansion (see below).
- :GETMAIL After loading the mail folder, retrieve new mail, if any, into this folder.
- :SHRINK After loading the mail folder, shrink the browser window. This option is ignored if :GETMAIL was specified.
- :FORGET If *foldername* is new to Lafite, don't remember it; i.e., do not add it to the menu of mail folders.
- :CONFIRM If *foldername* does not exist, require confirmation before creating it.

LAFITE.BROWSER.ICON.PREFERENCE

[Variable]

Specifies where a browser's icon should be positioned if it has not already been specified in a browser layout as described above. The possible values are

- NIL Place the icon at the bottom left corner of the browser window. This is the default.
- :ASK Prompt for a position.
- a function* Call this function, passing the browser window as its sole argument. The function should return a position, or NIL to make Lafite prompt for a position.

LAFITE.FOLDER.MENU.FONT

[Variable]

The font used in the menu of folders displayed by the **Browse** and **MoveTo** commands. Initially NIL, which means use the default system MENUFONT.

LAFITE.FOLDER.ICON

[Variable]

An icon specification for the icon used by Lafite browsers. It is of the form expected by TITLEDICONW, i.e., (*IconBitmap MaskBitmap TitleRegion*). This variable replaces the undocumented variables MSGFOLDERTEMPLATE, MSGFOLDERICON, and MSGFOLDERMASK.

LAFITE.MSG.ICON

[Variable]

An icon specification for the icon used by Lafite message editors. This variable replaces the undocumented variables MSGUNSENTREGION, MSGUNSENTICON, and MSGUNSENTMASK.

The function LAFITE itself has been extended to take advantage of the same options available to LAFITE.BROWSE.FOLDER, so that you can set up your entire default mail screen in a single call. Here is the new documentation:

(LAFITE *on/off* &optional *folder* &rest *options*)

[Function]

Turns Lafite on or off according to the first argument, which is one of the keywords :ON, :OFF, or :RESTART. For backward compatibility, the same symbols are accepted in the Interlisp package. :ON turns Lafite on, and has no effect if it is already on. :OFF turns it off, and has no effect if it is already off. :RESTART is equivalent to (LAFITE :OFF) followed by (LAFITE :ON *folder . options*).

The argument *folder* specifies a mail folder to browse. If it is omitted, i.e., for a simple (LAFITE :ON), it defaults to the value of DEFAULTMAILFOLDERNAME. NIL specifies no folder. So that you can specify *options* and still browse the default folder, *folder* = T also defaults to DEFAULTMAILFOLDERNAME.

The remaining arguments are optional keywords specifying what to do with the folder being browsed. The possible values are :ACTIVE, :GETMAIL, :SHRINK, :FORGET, and :CONFIRM, which are handled as described under LAFITE.BROWSE.FOLDER. For example, (LAFITE :ON "active" :CONFIRM :SHRINK) turns Lafite on, browses the folder "active", requiring confirmation if the file does not exist, and then shrinks the browser.

If *folder* is a list, it is interpreted as a list of specifications for files to browse and their placement. Each element is of the form (*foldername BrowserRegion IconPosition DisplayRegion . options*), which specifies invoking (LAFITE.BROWSE.FOLDER *foldername* (LIST *BrowserRegion IconPosition DisplayRegion*) *options*). If there were any *options* specified in the call to LAFITE itself, they are appended to the *options* specified in each element of *folder*. See LAFITE.BROWSE.FOLDER for details.

## Message Display

---

The —End of message— indication is back in operation for those of you who missed it in Medley.

Middle-button on the **Display** command now always prompts for its own display region (a box if LAFITE.DISPLAY.SIZE is non-NIL), and this window survives Updates; i.e., it stays open until you explicitly close it.

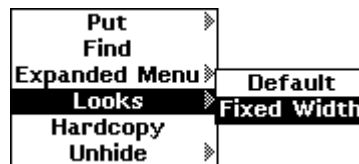
It is now possible to tell Lafite you do not care to see certain fields in the headers of messages, such as the numerous fields inserted by Arpanet mail transport.

|                              |            |
|------------------------------|------------|
| LAFITE.DONT.DISPLAY.HEADERS  | [Variable] |
| LAFITE.DONT.FORWARD.HEADERS  | [Variable] |
| LAFITE.DONT.HARDCOPY.HEADERS | [Variable] |

The value of each of these variables is a list of strings specifying the names of the fields you wish omitted by the **Display**, **Forward**, and **Hardcopy** commands, respectively. The alphabetic case of the strings is unimportant. When fields are omitted from a displayed message, an asterisk appears in the titlebar of the message. You can see the omitted fields by using the **Unhide** command from the titlebar menu (see below). There is no corresponding command to see the omitted headers in a forwarded message or, of course, in a hardcopied message.

The value ("Return-Path" "Redistributed" "Received" "Message-Id") is useful for filtering most of the rubbish from Arpanet headers.

Left- or Middle-button in the titlebar of a displayed message gives a modified TEdit menu of operations.



Those specific to Lafite are as follows:

**Looks** This command is used to change the appearance of the message. If you have selected a region of the message (longer than a single character), the command affects only that region; otherwise it applies to the whole message. This command affects only the display of this message this time; it has no effect on the message living in the mail folder.

The main **Looks** command brings up the same set of three menus as the normal TEdit Looks command, allowing you to choose any font and size. The subcommand **Fixed Width** changes the font to a fixed-width font, useful for viewing text formatted by folks in the fontless world. In addition, when changing to a fixed-width font, Lafite also sets the tab size to be the width of eight characters, something TEdit unfortunately doesn't do automatically. The subcommand **Default** sets the font back to Lafite's normal display font (the value of LAFITEDISPLAYFONT).

**Hardcopy** This command sends this single message to the default printer in whatever appearance it currently has. The hardcopy is sent immediately, independent of the setting of LAFITEHARDCOPYBATCHFLG.

**Unhide** If some fields of the message header have been omitted because they matched the strings in `LAFITE.DONT.DISPLAY.HEADERS`, this command reveals them. The subcommand **Hide** can be used to hide them again.

This menu can be further tailored using the following variables:

`LAFITE.EXTRA.DISPLAY.COMMANDS` [Variable]

This is the list of menu items added to the three possibly interesting TEdit commands offered in a message display window. Each item should be set up to return a function of one argument, which is applied to the message's text stream when the item is selected.

`LAFITE.LOOKS.SUBCOMMANDS` [Variable]

This is the list of menu subitems under the **Looks** command. It has the same format as `LAFITE.EXTRA.DISPLAY.COMMANDS`. At run time, Lafite inserts this list into the **Looks** item of `LAFITE.EXTRA.DISPLAY.COMMANDS`. The following function may be useful in implementing new Looks commands:

`(LAFITE.SET.LOOKS textstream newlooks paraloops)` [Function]

Changes the character looks of the current selection in *textstream*, or the whole stream if no more than one character is selected, to be *newlooks*, which can be a font descriptor or any argument acceptable to `TEDIT.LOOKS`. If *paraloops* is specified, it is a set of paragraph looks as expected by `TEDIT.PARALOOKS`. If *newlooks* is a fixed-width font and *paraloops* is `NIL`, this function also sets the default tab width of the selected text to be eight times the character width.

`LAFITEFIXEDWIDTHFONT` [Variable]

The font used by the **Fixed Width** command. It is initially the value of `DEFAULTFONT`, unless that font is variable-width, in which case it is set to Gacha 10.

`(LAFITE.HARDCOPY.TAB.WIDTH)` [Function]

When the **Fixed Width** command is used on a message, the tab width set for display is unlikely to be appropriate for hardcopy. If you subsequently issue the Hardcopy command on that window, Lafite has to guess what the tab width should be. Currently it does so by calling this function, whose initial definition computes a tab width based on `LAFITEFIXEDWIDTHFONT` coerced to Interpress. If you defaultly hardcopy to some other kind of printer, you may want to change this.

## Browser Commands ---

Middle-button on the **Update** command bypasses the normal Update options menu. Its behavior is controlled by the following variable:

`LAFITE.MIDDLE.UPDATE` [Variable]

A list of keywords specifying the normal way by which you like to update browsers. The choices are:

- :UPDATE Write out changes, but do not expunge deleted messages.
- :EXPUNGE Expunge deleted messages. If there are no deleted messages, this is equivalent to :UPDATE. Both :UPDATE and :EXPUNGE will also process any pending hardcopy (when LAFITEHARDCOPYBATCHFLG is T).
- :SHRINK After updating the folder, shrink the browser.
- :CLOSE After updating the folder, close the browser.
- :CONFIRM Require confirmation (via mouse) of the operation.

The keywords :UPDATE and :EXPUNGE are mutually exclusive, as are :SHRINK and :CLOSE. The default value is (:EXPUNGE :SHRINK :CONFIRM), which means that middle-button on **Update** prompts with a message like "Click LEFT to confirm Expunge and Shrink". The message specifies the operation implied by the setting of the variable, rather than always being "Expunge"; e.g., it might read "Click LEFT to confirm Hardcopy, Update table of contents and Shrink". As a special case, when the folder needs no updating at all and LAFITE.MIDDLE.UPDATE specifies :SHRINK, Lafite shrinks the browser immediately without asking for confirmation.

The menu of Update options offered by the Update, Shrink, and Close commands now more accurately reflects what needs to be done. In particular, there is no Expunge option if you have not deleted any messages.

The widest labels on the browser menu (**Undelete** and **Hardcopy**) have been shortened. With Lafite's default menu font (Helvetica 10 Bold), this allows browsers to be 54 points narrower than before, so that two browsers can fit side by side on the screen. (I personally prefer Modern 12 Bold (10 for users of NSDISPLAYSIZES), which yields a browser another 18 points narrower.) You can (awkwardly) change the labels by editing LAFITEBROWSERMENUITEMS.

LAFITEHARDCOPY.MIN.TOC now works even when LAFITEHARDCOPYBATCHFLG = T. (If more than LAFITEHARDCOPY.MIN.TOC messages are being hardcopied, the output is preceded by a table of contents.)

If you have ever performed **GetMail** on a browser (or browsed it with the :ACTIVE option), then the behavior of middle-button on the browser's icon behaves differently: instead of just expanding the window, it pops up a menu offering to get mail.



Choosing **GetMail** is equivalent to expanding the window, then choosing the regular **GetMail** command, but of course requires less interaction on your part.

Lafite browsers now support copy selection. Selecting with the mouse in a browser window while holding down the Copy or Shift keys selects the text of the field you are pointing at. You can select the Date, From, or Subject field. Selecting to the left of the Date field selects the whole message, producing a "summary line" such as "#185 14 Jun From: Carstairs.pa -- Meeting rescheduled". This use of Shift for copy selection means that only the CTRL key can be used to remove messages from the selection, an operation previously supported by either CTRL or Shift.

The title field of the browser has been rearranged so that the most interesting information appears first, and is thus less likely to fall off the right edge of the window. For example, the browser for {FS}<Carstairs>Mail>Active.mail might have a title reading "Browsing Active (Move To: Memos) on {FS}<Carstairs>Mail>".

LAFITEMOVETOCONFIRMFLG

[Variable]

This variable now only affects what happens when you choose the destination of a move from the menu (or using the middle-button accelerator), rather than typing it in (having selected "Other"). To summarize:

- NIL Do not require confirmation.
- LEFT Confirm only left-button MoveTo.
- MIDDLE Confirm only middle-button MoveTo.
- ALWAYS Require confirmation in all cases. This is the default.

Typing in a destination from the keyboard is enough evidence that you didn't slip on a mouse button, so confirmation is only required then if the destination file does not yet exist.

## Mail Sending

---

You can now control whether Lafite bothers to ask you about whether to send a message formatted or not:

LAFITE.SEND.FORMATTED

[Variable]

This variable is consulted when you ask to deliver a message that has what TEdit considers non-trivial formatting—font shifts, paragraph formatting, image objects, or NS characters. The choices are:

- NIL Send it unformatted.
- :ASK Prompt with a menu.
- T Send it formatted without asking.

The value can also be a list of elements (*formattingType answer*), specifying that when (TEDIT.FORMATTEDFILEP *message*) is *formattingType* the decision is *answer*. The initial value is

```
((NSCHARS :ASK)
 (CHARLOOKS :ASK)
 (PARALOOKS :ASK)
 (IMAGEOBJ T))
```

meaning to send messages with image objects formatted without asking (sending them unformatted is always a bad idea), but ask for all other kinds of formatted messages.

You can now send unformatted messages containing NS characters. However, note that the Koto version of TEdit did not understand NS characters in unformatted text, so you should not do this if you expect that some of your recipients have not yet upgraded to Lyric. If you never send



messages to Koto Lisp users (or don't care about this shortcoming), you might want to set the NSCHARS component of LAFITE.SEND.FORMATTED to NIL, i.e., have Lafite never send a message formatted merely because it contains NS characters.

Programs that compose message forms, such as those invoked from LAFITESPECIALFORMS, can specify that the message is to be delivered formatted without asking by giving the message (a textstream) the TEXTPROP property LAFITEFORMAT, value TEDIT.

Here are a couple more message customization variables:

LAFITE.SIGNATURE

[Variable]

A string with which to "sign" your messages. This string is appended to the standard message forms—those produced by the **SendMail**, **Answer**, and **Forward** commands—always starting it on a new line. There should be a carriage return at the beginning of the string if you want a blank line to set off the signature. The initial value is NIL, meaning no signature.

LAFITE.GV.FROM.FIELD

[Variable]

A string to use as the contents of the "From:" field of any Grapevine message you send that does not already have a From field. It must be a valid Grapevine address specification that parses down to your Grapevine identity, including registry. The usual syntax is "Human name <MailName.reg>", e.g., "Johann Amadeus Slonimsky III <Slo.pa>".

These two variables are examples of Lafite "personal" variables, in that they are closely bound to the logged-in user. The other is LAFITEDEFAULTHOST&DIR. If you change the logged in user (e.g., by calling (LOGIN)), Lafite resets all these variables to NIL, but remembers their values in case you log in again as yourself. This is designed to reduce confusion if you have more than one person using Lafite on the same machine. The values for each user are remembered as the value of the variable LAFITE.USER.INFO, an association list of user name and values, the latter being in property list format. Ordinarily you need not be aware of this variable, but you might, for example, want to initialize it with information about many users of a public machine.

---

## Grapevine

Lafite now does a more thorough job of parsing Grapevine messages according to the RFC 822 specification and objecting to malformed addresses. It also preserves full addresses when composing replies, rather than discarding all the commentary.

Lafite now supports "private distribution list" syntax in Grapevine messages. A private distribution list (dl) is an indirection mechanism: it allows you to name a possibly large group of addressees without explicitly enumerating them in the header of the message. Unlike a public dl, whose membership is maintained in the Grapevine database, a private dl is simply a file in which you have stored the real names. The names in the file must be in standard Grapevine syntax, just as you would type in the header of a message, but with two exceptions: (1) Names may be separated with one or more carriage returns, rather than commas, to enhance readability; and (2) The names must be fully qualified, i.e., the Grapevine registry must be included. The latter requirement eliminates confusion in the case where the current sender is in a different registry than the maintainer of the dl. The names can make use of all the standard Grapevine syntax, which allows you to include comments in the file. For example, you could have a line reading

<Carstairs.wbst> "Works with bj on the ACME project"

The file may be a TEdit file; Lafite strips out the formatting before attempting to parse the addresses in it.

The syntax for a private dl in a message is

To: *dlfilename*::

i.e., the name of the file containing the real names, followed immediately by the two characters colon, semi-colon. If the file name contains any characters whose syntax is significant to Grapevine messages—colon, semi-colon, parentheses, brackets—the file name must be enclosed in double quotes. For example,

To: "{FS9:PARC:Xerox}<Carstairs>Mail>Friends"::

The file name need not be complete; Lafite fills in defaults to locate the file according to the following two variables:

LAFITEDL.EXT

[Variable]

The default extension of private distribution list files. If a private dl is specified without an explicit extension, Lafite uses this extension when looking for the file. The value is initially "DL".

LAFITEDLDIRECTORIES

[Variable]

List of directories on which private distribution list files may be found. When you use a private dl in a message, Lafite searches LAFITEDEFAULTHOST&DIR, then this list, for the file containing the addressees. The value is initially NIL.

\*GV-SHOW-POSTMARK\*

[Variable]

When Lafite retrieves Grapevine mail, there is additional information about the transport of each message that Lafite normally discards as uninteresting, as it is for the majority of messages. When the variable \*GV-SHOW-POSTMARK\* is true, Lafite preserves this information in an extra line in the header of the message. The line reports the authenticated name of the sender, the time at which the message entered the Grapevine system, and the host from which it entered; for example,

GV-Info: Carstairs.pa at 4-Jun-88 23:38:00 from 204#16#

Note that the variable only affects the retrieval of mail, not its display. If the variable is NIL at the time of retrieval, the information is lost forever.

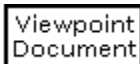
---

## NS Mail

The NS mail retrieval code correctly interprets messages that are really returned messages. It now turns such messages into a description of a failed delivery, with a mail server as the apparent sender, instead of just putting a cryptic "TransportProblem" header in the message and leaving the header looking as though the message was from you.

The NS mail watcher no longer maintains a "session" on the mail server. This reduces server load a bit, and means that there is no confusion if another client (either Lafite or Viewpoint) tries to retrieve the mail of a user who happens to be running Lafite on another machine at the same time.

NS Mail now handles "attachments" better. This change was actually made in a special release of NS mail in September, 1987, but is included here for completeness, and to explain the extensions that handle reference objects. Attachments are no longer left in your mailbox to be read later with, for example, Viewpoint. Instead, Lafite retrieves the entire attachment and encapsulates it into an image object that is enclosed as part of the text message, immediately following the header. A typical attachment appears in a mail message as:

Attachment: 

If you click inside the object with any mouse button, you are offered a menu of things you can do with the attachment. The choices vary according to the type of attachment:

- View as text** This brings up a window in which is displayed the raw content of the attachment as ascii bytes. Runs of non-ascii bytes are replaced by nulls to reduce the amount of garbage. Some attachments are utter gibberish, but some, such as Viewpoint documents and Interpress masters, contain sections that are plain text. With this command, you may be able to decide whether you care to do anything further with the attachment. (Sorry, there is no Viewpoint to TEdit converter, nor are there plans for one.)
- Put to file** This prompts you for a file name, and creates a file to contain the attachment. The file must be on an NS file server for this command to be very useful; otherwise, information will be lost. Once the file is so stored, you can retrieve it from Viewpoint and manipulate it just as if you had originally retrieved it as mail in Viewpoint. If you are running in Lyric, you must have the module NSRANDOM loaded for this command to work.
- Send to Printer** This command is only available for attachments that are in the form of an Interpress master. The command prompts you for a printer (which must be one that accepts Interpress, of course), and sends the attachment to it for printing.
- Expand folder** This command is only available for attachments that are in the form of a "folder". A folder is a mechanism for collecting several objects into a single one. The **Expand folder** command splits the attachment up into its component objects, each of which can be manipulated in the same way as a top-level attachment. For example, if the folder contains an Interpress master, you can print it.

If you use the **Put to file** command on a folder, the name component of the file name you type will be treated as the name of a new subdirectory, and the components of the folder will appear as files in that subdirectory. For other types of attachments, **Put to file** (usually) produces an ordinary (non-directory) file.

If the attachment is a Viewpoint "Reference" object, the **View as text** and **Send to Printer** commands apply to the file the reference mentions, not the attachment itself. The **Put to file** command is renamed **Store reference**, to make it clear it applies to the reference object itself,

rather than the file being referred to (which there is no point in copying, since Viewpoint can deal with the reference object itself just fine). References to folders also offer the command **FileBrowse**, which brings up a FileBrowser (if you have it loaded) browsing the directory referred to.

Messages containing attachments are otherwise just like formatted messages—you can move them to other folders, and you can forward them (assuming the mail is received by another Lafite recipient and did not have to pass through an information-losing mail gateway).

There is currently no mechanism for creating your own attachments to send to other users.

## Miscellaneous

---

The middle-button **Browse** command has a new subcommand **Browse & Forget**. This command prompts for a file name and browses it with the `:FORGET` option, i.e., it does not add it to the folder menu. Note that this command does not remove an existing folder from the menu, it merely refrains from adding a new one.

Lafite's table of contents code now handles messages containing NS strings in the Subject field. This fixes the bug where browsing such a file would claim that the toc was inconsistent with the folder, throw away the toc file and reparse from scratch. Also, the bug where Lafite reported "Folder is Empty" when discarding a malformed toc has been fixed.

Lafite now treats folder names as strings, and does its best to (a) ignore alphabetic case within folder names and (b) preserve the case that you type when creating a folder. This means, among other things, that you can now have mixed case in your `LAFITEDEFAULTHOST&DIR`, and that you can have folder names that are entirely numeric, such as "1186".

Lafite also keeps in its folder menu what the server considers the "canonical" name of each file to be. This means that if your folders include a file not on your default directory with a host not specified by its canonical name, then the next time you browse this folder, Lafite will think it is a new folder, leaving you with a menu containing both the old and the new names. Use the **Forget Folders** subcommand of **Browse** to dispose of such duplicates.

The information about your folders and message forms, formerly maintained in a file named `LAFITE.PROFILE` on your `LAFITEDEFAULTHOST&DIR`, is now kept in a file named `LAFITE.INFO` in a more extensible format. When you first run the new Lafite, if you have no `LAFITE.INFO` file, it will read `LAFITE.PROFILE` instead and convert to the new format.

Lafite does a better job now of noticing whether a mail file has been changed out from under you. It checks every time it opens the file, not just after logout. If the mail file has been changed but you have unsaved changes to the browser, Lafite will offer to save your changes if the mail file has merely been appended to. If, however, it has been expunged, so that the messages all live in different locations in the file, this is not possible, and the changes in your browser must be discarded.

When you return from logout and Lafite can't locate the file behind one or more of your browsers, it no longer closes the browser. The next time you try to do anything with the browser, such as display a message, Lafite will again try to locate the file. Thus, if the file server holding

the file is down, you need only wait for it to come back up before using the browser, rather than having to rebrowse the folder.

The after-logout code now runs mostly in a separate process, which allows the call to LOGOUT to return faster.

Lafite files are now of type LAFITE, rather than TEXT, since they typically contain non-text content, such as NS characters and TEdit formatting. On any kind of server but an NS file server, type LAFITE is usually coerced to BINARY.

## **Changes to Programmer Interface**

---

Well, actually, there is no documented programmer's interface. Numerous internals have changed; this section lists but a few.

With the new handling of modes, nearly everything surrounding the use of \LAFITEUSERDATA and the record LAFITEOPS (and hence the variable LAFITEMODELST) has changed. POLLNEWMAIL has been restructured; PRINTLAFITESTATUS behaves differently. There is a new hook LAFITENEWMAILFN. When this variable is non-NIL, its value is called, with no arguments, when the Lafite status window notes new mail.

The MAILFOLDER record has changed slightly. The use of its FOLDERDISPLAYWINDOWS field is different. The new field FOLDERDISPLAYREGIONS replaces BROWSERSELECTIONREGION.

The whole family of functions around \LAFITE.BROWSE has substantially changed.

Several functions have been deleted or renamed including LA.REMOVEDUPLICATES, LA.SETDIFFERENCE, PROFILEFILENAME, \LAFITE.MERGE.PROFILES (now \LAFITE.MERGE.NAMELISTS), CHANGEFLAGINFOLDER, \LAFITE.GETMAILFOLDER.

## **Partial List of Fixed or Obsolete ARs**

---

- 267 Easy font changing
- 307 Filter selected fields out of displayed msg?
- 552 Make it easier to change logged in user
- 1239 Recover from "Lafite is confused..."
- 1249 Want automatic GetMail on icon expansion
- 1273 Prompt for GV login if BadPassword error => Declined: use Login
- 1389 Answer command should preserve entire address
- 1644 Private dl's
- 2143 Want detachable display windows that survive Update
- 2421 Preserve case when creating files
- 2433 Use GETBOXREGION to place window?
- 2597 Check that GV headers conform to RFC822
- 3168 Want user-specifiable browser/ menu layout
- 3174 List of regions for browser, display, etc
- 3269 Update, GetMail etc should check creationdate/ eof and rebrowse if needed
- 4420 LAFITEHARDCOPY.MIN.TOC doesn't work with LAFITEHARDCOPYBATCHFLG
- 4628 addresses lacking a closing double quote break with illegal arg 65536

5109 Don't loop if bad NS password  
5110 Blank name (e.g. in Reply-to) completes to \*:domain:org  
5111 Want name parser that doesn't add default registries  
5116 Parser loops forever if header address has unmatched paren or quote  
5117 Want way of dealing with NS Mail attachments without running Star  
5121 Close/Shrink menu includes "Expunge" even if no deleted messages  
5122 Simultaneously watch for both NS and GV mail  
5125 Case sensitivity in LAFITEDEFAULTHOST&DIR screws up MoveTo  
5129 Want browser title rearranged to avoid clipping important info  
5134 Message from self not noted if user not authenticated when folder browsed  
5137 Want to be able to keep display window open after browser shrunk  
5300 Answer NS message in GV mode does not give To field  
5541 Default status window off screen on small 1186 screen  
5638 Prune headers on forwarded messages  
6372 LAFITENEWMAILTUNE not on daybreak  
6260 Blank To/cc line parsed incorrectly  
6547 Can't put Arpa address in From field  
6918 Send unformatted dies in COPYBYTES illegal arg  
7244 Numeric folder names die in string-equal  
8643 (il:lafite 'il:off) breaks in SHADEITEM, menu = nil  
8679 Changing modes with Lafite off doesn't reset \LAFITEUSERDATA?  
8995 When toc discarded, says "Folder empty"

---

---

## LAFITE UPDATE

---

---

Last Edited: November 16, 1989

This document describes changes to Lafite since the October, 1986 release. Comments and suggestions are welcome on the new features described herein.

### Installation

---

To install Lafite in a sysout where Lafite has never been loaded, simply load LAFITE.LCOM from the directory appropriate to your version of Lisp. However, if an older version of Lafite is already loaded in your sysout, you must instead Quit out of the old Lafite, and explicitly load *all* of Lafite's component files. If the old version of Lafite in your sysout is dated June, 1988 or later, you can do this by calling (LOAD-LAFITE "*directory*"), after which you may still need to load MAILSCAVENGE.LCOM if the old version was in your sysout; otherwise, load *all* of the following LCOMs from the appropriate directory:

|              |              |
|--------------|--------------|
| LAFITEBROWSE | NSMAIL       |
| LAFITEMAIL   | MAILSCAVENGE |
| LAFITESEND   | LAFITEFIND   |
| MAILCLIENT   | LAFITE       |

Of the files MAILCLIENT, NSMAIL, MAILSCAVENGE, LAFITEFIND, you need only load those that are already present in your sysout from the old Lafite (all are present in the standard Parc sysout, so all must be reloaded). Note that LAFITE.LCOM must be loaded last.

### Compatibility

---

This new Lafite is incompatible with most modules that thought they knew something about the internals of Lafite. Specific cases (this is mostly old news by now):

**Rooms.** Rooms has been updated to know about the new Lafite.

**Lens.** Lens has been updated to know about the new Lafite.

**LafitePrivateDL.** This LispUsers module is incompatible with the new Lafite, and also obsoleted by it (see the section "Grapevine" below).

**LafiteHighlight.** This probably still works, but you may prefer to use header filters in the new Lafite instead (see "Message Display" below).

**LafiteTimedDelete.** This probably still works.

**Vanilla Init.** Stan says he now has this working properly with the new Lafite.

**DateSort.** This module is obsoleted by the new Lafite.

**Lafite-Indent.** This is actually a TEdit macro package and is independent of Lafite.

## Lafite Modes

---

Lafite is now willing to operate in more than one mode "simultaneously". The mail watch background task checks all of your mailboxes in all the modes that are active, and **GetMail** retrieves mail from all mailboxes. The variable `LAFITE.USE.ALL.MODES` controls whether Lafite runs in more than one mode at once. The interesting values are:

- NIL This is the old way, where you must change modes manually. All commands operate only in the currently active mode.
- :POLL The mail watch task checks all modes, but **GetMail** only retrieves in the currently active mode.
- T Mail watch checks and **GetMail** retrieves in all modes.

The initial value is T. The flag has no effect on the **SendMail** command, which continues to bring up a message composition window in the current mode. It also does not affect the **Answer** command, which answers in the mode that the message was retrieved in. The retrieval mode is remembered in the table of contents file; in the absence of this knowledge, e.g., if the toc is deleted or the message was retrieved with an older Lafite, the **Answer** command uses simple rules that nearly always unambiguously distinguish GV from NS messages. Be aware that the value :POLL can be confusing to such hacks as automatic mail retrievers, since the status window can report new mail, yet **GetMail** won't retrieve it.

Ordinarily, Lafite only attempts to authenticate you in the non-primary modes when it starts up, and after logout. If, for example, you are in GV mode, and the NS authentication service is down when you start up Lafite, or your NS password is incorrect, it does not repeatedly try to authenticate you, but merely reports the condition once and behaves as if the failing mode does not exist. If conditions later change, you can force another attempt by using the mode switching commands to explicitly switch to the mode. If your password is incorrect, you can change it using the NS Login subcommand of Quit (see below).

**GetMail** only attempts to retrieve mail from the servers that reported new mail in the most recent poll. This means that if one of your mail servers is not responding, you no longer have to wait during **GetMail** for it to time out. If you click **GetMail** at a time when the status window reports "No New Mail", then **GetMail** will attempt to contact all of your mailbox servers, just as it did in previous releases.

## Turning Lafite on and off

---

You can now turn Lafite on or switch modes from the background menu.

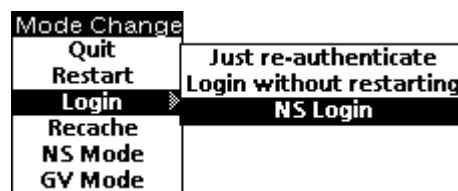


- Mail** This brings up a standard message sending window in the current mode, whether Lafite is on or not. If Lafite is not on, it will attempt to authenticate you in the current mode.



- Send Mail** This is the same as the main **Mail** command, except that you get to choose which kind of message form, just as middle-button on the Lafite status window's **Send Mail** command allows.
- Turn Lafite On** This is the same as calling (LAFITE :ON). It has no effect if Lafite is already on.
- Set Lafite Mode** This command allows you to change Lafite's mode. It offers a menu of the mode choices found also as part of the middle-button **Quit** command on the Lafite status window.

The middle-button **Quit** command on the Lafite status window has two new subcommands, **Recache** and **Login**. Here, for completeness, is a description of all the commands:



- Quit** This is the same as the main command, which is the same as calling (LAFITE :OFF). It closes all folders, asking about updating them if any need it, then turns off Lafite.
- Restart** This turns Lafite off, then immediately back on; same as (LAFITE :RESTART).
- Login** Changes the logged-in user for Lafite. This command first turns Lafite off, so that any updating required by closing out the previous user's folders happens under the previous user's login. It then prompts for user name and password, exactly as if you had called (LOGIN) yourself, and then for a new LAFITEDEFAULTHOST&DIR. Finally, it turns Lafite back on.
- Just re-authenticate** Re-authenticates the currently logged-in user without prompting for a new login. This may be useful if Lafite's original attempt at authenticating the user failed or got inconsistent information due to network difficulties.
- Login without restarting** Performs a login without changing the state of Lafite. This is useful if you had previously logged in with an incorrect password, or if you have multiple user identities all sharing the same set of folders.
- NS Login** Prompts for a new login just for NS mail. You may need to use this if your global user name does not correspond to your NS user name or an alias of it in the default domain, or if you have different passwords for Grapevine and NS. There is actually one of these commands for each mode that understands an independent login, but NS is the only current instantiation.
- Recache** In the case of several of the public interface variables, Lafite computes other structures based on them, and does not automatically recompute them when you change the public variable. The recache command forces Lafite to recompute these "caches" of the variables' values. A recache also occurs whenever Lafite is (re)started. The command is implemented by the function LAFITE.COMPUTE.CACHED.VARS, which you can advise if you have other caches that Lafite doesn't know about. The variables

currently affected by this command (i.e., changes in whose values are not otherwise noticed) are:

```
LAFITE.DONT.DISPLAY.HEADERS
LAFITE.DONT.FORWARD.HEADERS
LAFITE.DONT.HARDCOPY.HEADERS
LAFITE.EXTRA.DISPLAY.COMMANDS
LAFITE.LOOKS.SUBCOMMANDS
LAFITE.GV.FROM.FIELD
LAFITE.HOST.ABBREVS.
```

### NS Mode

**GV Mode** Switch to the specified mode. There is one of these for each mode.

## Screen Appearance ---

There are several new variables and one function for controlling the appearance of your Lafite screen.

LAFITE.BROWSER.LAYOUTS [Variable]

A list of browser "layouts" specifying where Lafite should place browsers, their display windows and their icons. Each layout is of the form (*BrowserRegion IconPosition DisplayRegion*). *IconPosition* and *DisplayRegion* are optional. The region variables are standard window system regions of the form (*left bottom width height*); *IconPosition* is a position of the form (*x . y*). When the **Browse** command creates a new browser window, it chooses the first element of LAFITE.BROWSER.LAYOUTS that is not already in use. If all are in use, it prompts for a region.

LAFITE.DISPLAY.SIZE [Variable]

Specifies the default size of display windows, other than those already specified in a browser layout. If the value of this variable is a size (*width . height*), then Lafite prompts for a region with a box this size. If the value is NIL, Lafite prompts for a general region, requiring you to drag from one corner to the opposite. The initial value is (500 . 300).

LAFITE.EDITOR.LAYOUTS [Variable]

A list of editor "layouts" specifying where Lafite should place message composition editors and their icons. As message editors have only a single window, each layout consists of only two elements (*EditorRegion IconPosition*). When one of the message-sending commands creates a new editor window, it chooses the first element of LAFITE.EDITOR.LAYOUTS that is not already in use. If all are in use, it prompts for a region.

LAFITE.EDITOR.SIZE [Variable]

Specifies the default size of message composition editors, other than those already specified in LAFITE.EDITOR.LAYOUTS. If the value of this variable is a size (*width . height*), then Lafite prompts for a region with a box this size. If the value is NIL, Lafite prompts for a general region, requiring you to drag from one corner to the opposite. The initial value is (470 . 300).

For backward compatibility, if `LAFITE.BROWSER.LAYOUTS` is `NIL`, it is considered to contain the single element `(LIST LAFITEBROWSERREGION NIL LAFITEDISPLAYREGION)`. If `LAFITE.EDITOR.LAYOUTS` is `NIL`, it is considered to contain the single element `(LIST LAFITEEDITORREGION)`. The variables `LAFITEBROWSERREGION`, `LAFITEDISPLAYREGION`, and `LAFITEEDITORREGION` are now obsolete and may be removed in a future release.

`(LAFITE.BROWSE.FOLDER foldername layout options —)`

[Function]

Programmatic interface to the **Browse** command. Browses the mail folder named *foldername* and returns a mailfolder object, or `NIL` on failure. If a browser already exists for *foldername* it is returned; otherwise, a new browser is created. The normal Lafite defaulting rules apply to *foldername*, so, for example, "active" defaults to the file "active.mail" on the directory specified by `LAFITEDEFAULTHOST&DIR`. The argument *layout* is a list of up to three elements describing the layout of the browser; it takes the same form as elements of `LAFITE.BROWSER.LAYOUTS`. *options* is a list of zero or more keywords from among the following:

- :ACTIVE The browser is considered an "active" mail browser. Currently the only import of this setting is to enable GetMail on expansion (see below).
- :GETMAIL After loading the mail folder, retrieve new mail, if any, into this folder.
- :SHRINK After loading the mail folder, shrink the browser window. This option is ignored if `:GETMAIL` was specified.
- :FORGET If *foldername* is new to Lafite, don't remember it; i.e., do not add it to the menu of mail folders.
- :CONFIRM If *foldername* does not exist, require confirmation before creating it.
- :OLD If *foldername* does not exist, do not create it, but return `NIL` instead.

`LAFITE.BROWSER.ICON.PREFERENCE`

[Variable]

Specifies where a browser's icon should be positioned if it has not already been specified in a browser layout as described above. The possible values are

- `NIL` Place the icon at the bottom left corner of the browser window. This is the default.
- :ASK Prompt for a position.
- a function* Call this function, passing the browser window as its sole argument. The function should return a position, or `NIL` to make Lafite prompt for a position.

`LAFITE.FOLDER.MENU.FONT`

[Variable]

The font used in the menu of folders displayed by the **Browse** and **MoveTo** commands. Initially `NIL`, which means use the default system `MENUFONT`.

`LAFITE.FOLDER.ICON`

[Variable]

An icon specification for the icon used by Lafite browsers. It is of the form expected by `TITLEDICONW`, i.e., *(IconBitmap MaskBitmap TitleRegion)*. This variable replaces the undocumented variables `MSGFOLDERTEMPLATE`, `MSGFOLDERICON`, and `MSGFOLDERMASK`.

LAFITE.MSG.ICON

[Variable]

An icon specification for the icon used by Lafite message editors. This variable replaces the undocumented variables MSGUNSENTREGION, MSGUNSENTICON, and MSGUNSENTMASK.

The function LAFITE itself has been extended to take advantage of the same options available to LAFITE.BROWSE.FOLDER, so that you can set up your entire default mail screen in a single call. Here is the new documentation:

(LAFITE *on/off* &optional *folder* &rest *options*)

[Function]

Turns Lafite on or off according to the first argument, which is one of the keywords :ON, :OFF, or :RESTART. For backward compatibility, the same symbols are accepted in the Interlisp package. :ON turns Lafite on, and has no effect if it is already on. :OFF turns it off, and has no effect if it is already off. :RESTART is equivalent to (LAFITE :OFF) followed by (LAFITE :ON *folder* . *options*).

The argument *folder* specifies a mail folder to browse. If it is omitted, i.e., for a simple (LAFITE :ON), it defaults to the value of DEFAULTMAILFOLDERNAME. NIL specifies no folder. So that you can specify *options* and still browse the default folder, *folder* = T also defaults to DEFAULTMAILFOLDERNAME.

The remaining arguments are optional keywords specifying what to do with the folder being browsed. The possible values are :ACTIVE, :GETMAIL, :SHRINK, :FORGET, and :CONFIRM, which are handled as described under LAFITE.BROWSE.FOLDER. For example, (LAFITE :ON "active" :CONFIRM :SHRINK) turns Lafite on, browses the folder "active", requiring confirmation if the file does not exist, and then shrinks the browser.

If *folder* is a list, it is interpreted as a list of specifications for files to browse and their placement. Each element is of the form (*foldername BrowserRegion IconPosition DisplayRegion* . *options*), which specifies invoking (LAFITE.BROWSE.FOLDER *foldername* (LIST *BrowserRegion IconPosition DisplayRegion*) *options*). If there were any *options* specified in the call to LAFITE itself, they are appended to the *options* specified in each element of *folder*. See LAFITE.BROWSE.FOLDER for details.

## Message Display

---

The —*End of message*— indication is back in operation for those of you who missed it in early versions of Medley.

Middle-button on the **Display** command now always prompts for its own display region (a box if LAFITE.DISPLAY.SIZE is non-NIL), and this window survives Updates; i.e., it stays open until you explicitly close it.

It is now possible to tell Lafite you do not care to see certain fields in the headers of messages, such as the numerous fields inserted by Arpanet mail transport.

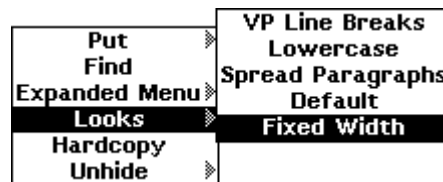
LAFITE.DONT.DISPLAY.HEADERS [Variable]  
 LAFITE.DONT.FORWARD.HEADERS [Variable]  
 LAFITE.DONT.HARDCOPY.HEADERS [Variable]

The value of each of these variables is a list of strings specifying the names of the fields you wish omitted by the **Display**, **Forward**, and **Hardcopy** commands, respectively. The alphabetic case of the strings is unimportant. When fields are omitted from a displayed message, an asterisk appears in the titlebar of the message. You can see the omitted fields by using the **Unhide** command from the titlebar menu (see below). There is no corresponding command to see the omitted headers in a forwarded message or, of course, in a hardcopied message.

The value ("Return-Path" "Redistributed" "Received" "Message-Id" "Errors-To") is useful for filtering most of the rubbish from Arpanet headers.

If you include the symbol GV in the list, Lafite filters the ugly "GVGV..." supplemental header from NS messages that arrived via the GV gateway. This supplement includes any headers that didn't translate into NS land, such as comments in addresses ("Fred Carstairs <fc@foo.bar.com>") and all those "Received:" lines in arpa mail. By filtering this entire section you could occasionally miss an interesting header hiding in there, but you can still use **Unhide**, of course.

Left- or Middle-button in the titlebar of a displayed message gives a modified TEdit menu of operations.



Those specific to Lafite are as follows:

**Looks** This command is used to change the appearance of the message. If you have selected a region of the message (longer than a single character), the command affects only that region; otherwise it applies to the whole message. This command affects only the display of this message this time; it has no effect on the message living in the mail folder.

The main **Looks** command brings up the same set of three menus as the normal TEdit Looks command, allowing you to choose any font and size. The subcommands apply some built-in looks or other transformations:

**Fixed Width** Changes the font to a fixed-width font, useful for viewing text formatted by folks in the fontless world. In addition, when changing to a fixed-width font, Lafite also sets the tab size to be the width of eight characters, something TEdit unfortunately doesn't do automatically.

**Default** Sets the font back to Lafite's normal display font (the value of LAFITEDISPLAYFONT).

**Spread Paragraphs** Inserts 10-point leading between paragraphs. This is useful for messages that come from Tioga, which doesn't put a blank line between paragraphs in the plain text body.

**Lowercase** Converts message body to lowercase. This is useful for messages from those bozos who haven't discovered their Shift Lock yet, or are typing on Model 33 Teletypes. This and the next command change only the body of the message, not the header.

**VP Line Breaks** This turns the character #035, which at least some mail senders in the XNS world seem to use as an end-of-line character, into carriage return.

**Hardcopy** This command sends this single message to the default printer in whatever appearance it currently has. The hardcopy is sent immediately, independent of the setting of LAFITEHARDCOPYBATCHFLG.

**Unhide** If some fields of the message header have been omitted because they matched the strings in LAFITE.DONT.DISPLAY.HEADERS, this command reveals them. The subcommand **Hide** can be used to hide them again.

This menu can be further tailored using the following variables:

LAFITE.EXTRA.DISPLAY.COMMANDS [Variable]

This is the list of menu items added to the three possibly interesting TEdit commands offered in a message display window. Each item should be set up to return a function of one argument, which is applied to the message's text stream when the item is selected.

LAFITE.LOOKS.SUBCOMMANDS [Variable]

This is the list of menu subitems under the **Looks** command. It has the same format as LAFITE.EXTRA.DISPLAY.COMMANDS. At run time, Lafite inserts this list into the **Looks** item of LAFITE.EXTRA.DISPLAY.COMMANDS. The following function may be useful in implementing new Looks commands:

(LAFITE.SET.LOOKS *textstream newlooks paralooks*) [Function]

Changes the character looks of the current selection in *textstream*, or the whole stream if no more than one character is selected, to be *newlooks*, which can be a font descriptor or any argument acceptable to TEDIT.LOOKS. If *paralooks* is specified, it is a set of paragraph looks as expected by TEDIT.PARALOOKS. If *newlooks* is a fixed-width font and *paralooks* is NIL, this function also sets the default tab width of the selected text to be eight times the character width.

LAFITEFIXEDWIDTHFONT [Variable]

The font used by the **Fixed Width** command. It is initially the value of DEFAULTFONT, unless that font is variable-width, in which case it is set to Gacha 10.

If you have in older versions of Lafite set LAFITEDISPLAYFONT to a fixed-width font just so you can easily read messages from outside users, you might consider setting LAFITEDISPLAYFONT

back to a more pleasing font and using the **Fixed Width** command in the few cases where you need it.

(LAFITE.HARDCOPY.TAB.WIDTH)

[Function]

When the **Fixed Width** command is used on a message, the tab width set for display is unlikely to be appropriate for hardcopy. If you subsequently issue the Hardcopy command on that window, Lafite has to guess what the tab width should be. Currently it does so by calling this function, whose initial definition computes a tab width based on LAFITEFIXEDWIDTHFONT coerced to Interpress. If you defaultly hardcopy to some other kind of printer, you may want to change this.

## Browser Commands

---

Middle-button on the **Update** command bypasses the normal Update options menu. Its behavior is controlled by the following variable:

LAFITE.MIDDLE.UPDATE

[Variable]

A list of keywords specifying the normal way by which you like to update browsers. The choices are:

- :UPDATE Write out changes, but do not expunge deleted messages.
- :EXPUNGE Expunge deleted messages. If there are no deleted messages, this is equivalent to :UPDATE. Both :UPDATE and :EXPUNGE will also process any pending hardcopy (when LAFITEHARDCOPYBATCHFLG is T).
- :SHRINK After updating the folder, shrink the browser.
- :CLOSE After updating the folder, close the browser.
- :CONFIRM Require confirmation (via mouse) of the operation.

The keywords :UPDATE and :EXPUNGE are mutually exclusive, as are :SHRINK and :CLOSE. The default value is (:EXPUNGE :SHRINK :CONFIRM), which means that middle-button on **Update** prompts with a message like "Click LEFT to confirm Expunge and Shrink". The message specifies the operation implied by the setting of the variable, rather than always being "Expunge"; e.g., it might read "Click LEFT to confirm Hardcopy, Update table of contents and Shrink". As a special case, when the folder needs no updating at all and LAFITE.MIDDLE.UPDATE specifies :SHRINK, Lafite shrinks the browser immediately without asking for confirmation.

The menu of Update options offered by the Update, Shrink, and Close commands now more accurately reflects what needs to be done. In particular, there is no Expunge option if you have not deleted any messages.

The widest labels on the browser menu (**Undelete** and **Hardcopy**) have been shortened. With Lafite's default menu font (Helvetica 10 Bold), this allows browsers to be 54 points narrower than before, so that two browsers can fit side by side on the screen. (I personally prefer Modern 12 Bold (10 for users of NSDISPLAYSIZES), which yields a browser another 18 points narrower.) You can (awkwardly) change the labels by editing LAFITEBROWSERMENUITEMS.

LAFITEHARDCOPY.MIN.TOC now works even when LAFITEHARDCOPYBATCHFLG = T. (If more than LAFITEHARDCOPY.MIN.TOC messages are being hardcopied, the output is preceded by a table of contents.)

If you have ever performed **GetMail** on a browser (or browsed it with the :ACTIVE option), then the behavior of middle-button on the browser's icon behaves differently: if you hold down the button, it pops up a menu offering to get mail.

**Get Mail**

Choosing **GetMail** is equivalent to expanding the window, then choosing the regular **GetMail** command, but of course requires less interaction on your part. If you release the button without selecting **GetMail**, the browser is expanded, just as if there had been no menu interaction at all.

Lafite browsers now support copy selection. Selecting with the mouse in a browser window while holding down the Copy or Shift keys selects the text of the field you are pointing at. You can select the Date, From, or Subject field. Selecting to the left of the Date field selects the whole message, producing a "summary line" such as "#185 14 Jun From: Carstairs.pa -- Meeting rescheduled". This use of Shift for copy selection means that only the CTRL key can be used to remove messages from the selection, an operation previously supported by either CTRL or Shift.

The title field of the browser has been rearranged so that the most interesting information appears first, and is thus less likely to fall off the right edge of the window. For example, the browser for {FS}<Carstairs>Mail>Active.mail might have a title reading "Browsing Active (Move To: Memos) on {FS}<Carstairs>Mail>".

LAFITEMOVETOCONFIRMFLG

[Variable]

This variable now only affects what happens when you choose the destination of a move from the menu (or using the middle-button accelerator), rather than typing it in (having selected "Other"). To summarize:

- NIL    Do not require confirmation.
- LEFT   Confirm only left-button MoveTo.
- MIDDLE   Confirm only middle-button MoveTo.
- ALWAYS   Require confirmation in all cases. This is the default.

Typing in a destination from the keyboard is enough evidence that you didn't slip on a mouse button, so confirmation is only required then if the destination file does not yet exist.

LAFITE.AUTO.MOVE.MENU

[Variable]

Setting this variable non-NIL enables an automatic "accelerated Move To" menu. The menu is attached to your browser window and contains the names of all folders to which you have moved messages from this browser. Selecting a folder name with the mouse is equivalent to selecting the **MoveTo** command in the browser and choosing that folder name, with no confirmation required. The menu also contains the items "Display" and "Delete", which are simply copies of the normal browser commands. This option is convenient when walking through a browser, dispatching messages into other folders. Of course, to move a message into some other



folder not in the menu, use the regular **MoveTo** command; afterwards, the folder you choose will be added to the accelerated menu.

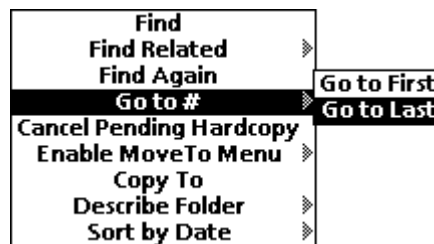
The menu does not appear until the first time you issue a **MoveTo** from this browser to a folder other than the default MoveTo (the one that middle-button chooses). It is attached by default to the right-hand side of the browser window, but you can set `LAFITE.AUTO.MOVE.MENU` to one of the symbols `LEFT`, `RIGHT`, `TOP` or `BOTTOM` to be explicit about where you want it.

You can get an accelerated Move To menu when `LAFITE.AUTO.MOVE.MENU` is `NIL` (or before you have moved to a second folder) by selecting **Enable Move To Menu** from the middle-button title-bar menu. In this case, you are asked to choose which folders you want in the menu.

You can separately close the menu without closing the browser window. To reopen the menu, use the subcommand **Restore Move To Menu** from the title-bar menu.

## Browser Title-Bar Menu

Lafite now automatically loads the formerly separate module LAFITEFIND. In addition to the three Find commands available from the menu you get when you hold down the middle button in a browser's title bar, there are several new commands. All are described here briefly for completeness.



**Find** Searches the browser for one or more messages matching some search criteria, which are obtained by a series of additional menus and type-in:

1. **Find {Next | Previous} {One | All}** chooses between searching forward from the current message or backward, and whether to stop at the first matching message, or to go ahead and find all matching messages at once.
2. **{From | Subject | Body | Mark | Related}** indicates what part of the message to look at. **From** and **Subject** specify the corresponding fields in the header of a message; **Body** specifies the entire message; **Mark** specifies the single-character mark to the left of the message number in the browser window; **Related** specifies using the **Find Related** command (below).

Actually, the **From** search specification is intended to match what you see in the browser window, which means you can also use it to search for messages you sent to another person; i.e., **From** searches the From field of every message, and in addition searches the To field of messages from you.

A **Body** search is substantially slower than the others, since it has to scan the mail file, rather than just the table of contents.

3. Finally, you are prompted (in all except **Related**) to type a string (or single character if **Mark**) to search for. Alphabetic case is irrelevant.

On a successful search, Lafite leaves as the browser's selection the single matching message (for **One**) or all matching messages (for **All**).

**Find Related** Searches for all the messages "related" to this one; specifically, for any message whose subject contains the current message's subject (minus an initial "Re:", if any) as a substring. In most cases, this finds exactly the messages that were sent using the **Answer** command in reply to this message or to a subsequent reply. Of course, if the current message's subject is empty or some short uninformative phrase (e.g., "bug"), the search may find more messages than you intended. The search is normally forward; you can search backward with a subcommand. At the end of this command, all related messages, including the current one, are selected, so you can use Display to cycle through them, MoveTo to put them all in some other folder, etc. If you want to find only the *first* related message, use the **Find** command and choose the **Related** "part".

**Find Again** Repeats the previous **Find** command. This is most useful when the previous search was a **Find Next One** or **Find Previous One**, in which case you can step through all matching messages one at a time using this command. The "previous search" is global, i.e., not confined to the browser in which it was issued, so you can, for example, search one browser, then use **Find Again** to perform the same search in a different browser.

**Go to #** Scrolls the browser to a message specified by number and selects it (and unselects any other message). You specify the number either via mouse with the numberpad reader or by typing digits from the keyboard. The subcommands **Go to First** and **Go to Last** allow you to jump directly to the first or last message in the browser.

**Cancel Pending Hardcopy** If you have LAFITEHARDCOPYBATCHFLG set to T and have issued the Hardcopy command for one or more messages but not yet issued an Update (which will actually perform the hardcopy formatting), this command will cancel the hardcopy.

**Enable Move To Menu** Brings up an accelerated MoveTo menu, as described in the previous section. The subcommand **Restore MoveTo Menu** simply restores an earlier MoveTo menu that you had closed, without asking what folders should be in it.

**Copy To** Copies the selected message(s) to the folder of your choice. This is the same as **MoveTo** from the main menu, but it does not delete the message or mark it moved, nor is there a middle-button accelerator.

**Describe Folder** Prints in the browser's prompt window some descriptive information about the folder: its full file name, the number of messages, the number of deleted messages (if any), and the number of disk (512-byte) pages it occupies. The subcommand **Inspect Folder** is intended for debugging; it brings up an inspector on the browser's MAILFOLDER object.

**Sort by Date** Sorts the browser's contents by date. The subcommand **Sort Selected Range** sorts only between the first and last selected messages. These commands are described further below.

## Sorting Mail

---

Lafite now has the ability to sort messages in a browser in order of their "Date:" fields, using the titlebar command **Sort by Date** or its subcommand **Sort Selected Range**. Messages lacking a Date field or having an unparseable Date field are sorted so as to remain next to the preceding message. The messages are sorted in the browser only, until you next **Update**, at which time the messages are written to the file in the sorted order. When messages have been rearranged, the **Update** option **Write Out Changes Only** is not available, as there is no file format for out of order messages; you must **Expunge**, even if there are no deleted messages.

You can also have Lafite automatically sort new mail:

LAFITE.SORT.NEW.MAIL

[Variable]

If the value is T, Lafite always sorts new mail; if :MULTIPLE, Lafite only sorts when the mail was retrieved from more than one server. Initial value is NIL.

Note that even when using a single mail server, mail can easily be out of order, due to the varying speeds at which messages pass through gateways, etc. Sorting assures that the messages will appear in the order in which they were posted. Sorting new mail seems fast enough that it is probably always worth doing; hence, the :MULTIPLE option may be dropped at some future time (send in your votes now).

Of course, to do the sorting, Lafite must now actually parse the Date field of messages. This also allows Lafite to display the date in the browser in a canonical form, rather than one that varies with the whim of the sender's software. Lisp's date parser (IDATE) was substantially beefed up to handle a wider variety of dates. It now claims to handle all dates legal in RFC822 syntax (except the silly single-digit military time zones), plus a fair variety of other formats found in mailers of recent years. The parser changes are in a module DATEPATCH loadable separately from Lafite.

Lafite considers dates older than 1970 spurious, so as to avoid being fooled by messages from machines that neglected to set the time.

With a new representation of dates, Lafite also has a new table of contents format. If you browse a folder with an old-style toc, Lafite will print "(older format)" when it reads it. Next time you do an Update, the toc will be written in the new format. Old versions of Lafite cannot read the toc files saved by this new version of Lafite, so if you try to browse a new folder with an old Lafite it will be forced to parse the file from scratch.

Lafite does not automatically parse dates on old messages, so until you issue a Sort command, old message dates will still be displayed as whatever string the old toc saved, not the new canonical form. In addition, when you *do* issue the Sort command, it must first go back and retrieve afresh all the date fields and parse them, an operation whose speed is dominated by the access time to your file system, much as parsing a folder from scratch.

## Folder Hierarchy

---

*This feature is based on a package written by Mike Dixon.*

You can now organize your folders into a hierarchy, so that the folder menus presented by **Browse** and **MoveTo** are easier to use. Folders are organized into *groups*. A folder can be in any number of groups. Groups can have subgroups. The top level of the folder menu is composed of some or all of the groups, plus all folders that are not in any group.

The **Edit Folder Hierarchy** command on middle-button Browse lets you define new groups and change existing ones.



### Edit Folder Hierarchy

**Edit a Group** The top-level command is the same as the subcommand **Edit a Group**. Lafite brings up a menu of all your defined groups. After you choose one, it offers you a choice of things to do to the group:



**Delete Group** does just that. Any folder in that group that is in no other group will reappear in the top-level folder menu. **Rename Group** lets you change the name of the group. **Change Members** prompts you with a menu of all folders, with the current group members preselected; you can add or remove folders. The folders in this menu are arranged with the ones not yet in any group clustered together to make it easy for you when you are first defining groups. **Create Subgroup** prompts you for the name of a new group, then which folders to put in it. The new group will appear as a member of the submenu presented by the supergroup, and its members in turn appear as a submenu to that. **Change Subgroups** lets you add or remove subgroups from the current group.

**Add New Group** Like **Create Subgroup**, but the new group appears at top level in the menu.

**Change Top-Level Groups** Prompts you to select which groups should appear in the top level of the folder menu. By default, groups created with **Add New Group** appear in the top level, while groups created by **Create Subgroup** do not. Regardless of your choice here, any group that is not currently a subgroup of another is included in the top level menu so that you don't lose it.

Note: the folder hierarchy is saved on your Lafite.info file, along with folder names and form names. Older versions of Lafite do not know about this structure, but those since June 1988 will at least preserve it. However, if you delete folders in an old version and then return to a new version of Lafite, the deleted folders may still appear in the hierarchy; you must manually delete

them either by editing the variable `LAFITE.FOLDER.STRUCTURE` or by editing the file `Lafite.info`.

## Mail Sending

---

There are two additional commands on the message composition window:

- Reply To** This command inserts a "Reply-to: *Your Name*" field in the header of the message. The name is selected for pending delete, so if you want replies to be sent to a different address, you can just start typing it. Lafite in Grapevine mode automatically prompts you to insert a Reply-to field at delivery time if one of the addressees has the syntax of a distribution list name (contains the character "^"). However, it can't figure this out for XNS or Arpa distribution lists, so you are strongly encouraged to insert the Reply-to field on your own with this command.
- Change Mode** This lets you change the mode of a message from GV to NS or vice versa. This can be useful if you clicked **SendMail** while in the wrong mode, or you want to Answer a message but do so from the other side of the fence. This command does not make any attempt to fix addresses already in the header to their other form; you have to do this yourself. The one exception is that if the cc or Reply-to field consists exactly of your mail name in the old mode, it substitutes your name in the new mode.

In addition, the old **Save Form** command has been renamed simply **Save**. This is in recognition of the fact that people seem much more often to want to save (checkpoint) messages in progress, than to create private mail forms. The message window now remains open after using the **Save** command (close it as you wish). When a previously saved message is ultimately delivered, you are asked whether you want to keep the saved form or delete it. There is also a menu item **Delete Message Form** underneath middle-button **Browse** for deleting a saved message explicitly. You can think of **Save** as a fancy version of TEdit's Put, which stores the message in your mail directory and remembers it (in the Saved Form menu) for a later Get.

You can now control whether Lafite bothers to ask you about whether to send a message formatted or not:

`LAFITE.SEND.FORMATTED`

[Variable]

This variable is consulted when you ask to deliver a message that has what TEdit considers non-trivial formatting—font shifts, paragraph formatting, image objects, or NS characters. The choices are:

- `NIL` Send it unformatted.
- `:ASK` Prompt with a menu.
- `T` Send it formatted without asking.

The value can also be a list of elements (*formattingType answer*), specifying that when (`TEDIT.FORMATTEDFILEP message`) is *formattingType* the decision is *answer*. The initial value is

```
((NSCHARS :ASK)
 (CHARLOOKS :ASK)
```

```
(PARALOOKS :ASK)
(IMAGEOBJ T))
```

meaning to send messages with image objects formatted without asking (sending them unformatted is always a bad idea), but ask for all other kinds of formatted messages.

You can now send unformatted messages containing NS characters. However, note that versions of TEdit before Medley did not fully understand NS characters in unformatted text, so you should not do this if you expect that some of your Lisp recipients have not yet upgraded to Medley. If you never send messages to Koto or Lyric Lisp users (or don't care about this shortcoming), you might want to set the NSCHARS component of LAFITE.SEND.FORMATTED to NIL, i.e., have Lafite never send a message formatted merely because it contains NS characters.

Programs that compose message forms, such as those invoked from LAFITESPECIALFORMS, can specify that the message is to be delivered formatted without asking by giving the message (a textstream) the TEXTPROP property LAFITEFORMAT, value TEDIT. If the message should turn out not to be formatted after all, it is sent as plain text, just as if the LAFITEFORMAT property were not present.

You can also select the mode of the message by giving the message the TEXTPROP property LAFITEMODE. The value should be the mode, e.g., GV or NS. With no such property, the message is sent in the current mode.

Here are some more message customization variables:

LAFITE.SIGNATURE [Variable]

A string with which to "sign" your messages. This string is appended to the standard message forms—those produced by the **SendMail**, **Answer**, and **Forward** commands—always starting it on a new line. There should be a carriage return at the beginning of the string if you want a blank line to set off the signature. The initial value is NIL, meaning no signature.

LAFITE.GV.FROM.FIELD [Variable]

A string to use as the contents of the "From:" field of any Grapevine message you send that does not already have a From field. It must be a valid Grapevine address specification that parses down to your Grapevine identity, including registry. The usual syntax is "Human name <MailName.reg>", e.g., "Johann Amadeus Slonimsky III <Slo.pa>".

Note that unlike LAFITE.SIGNATURE, which appears in the message form itself, there is no visible evidence of LAFITE.GV.FROM.FIELD during message composition. Lafite does not use its value until it is time to add the Date and From fields when it delivers the message.

These two variables are examples of Lafite "personal" variables, in that they are closely bound to the logged-in user. The other is LAFITEDEFAULTHOST&DIR. If you change the logged in user (e.g., by calling (LOGIN)), Lafite resets all these variables to NIL, but remembers their values in case you log in again as yourself. This is designed to reduce confusion if you have more than one person using Lafite on the same machine. The values for each user are remembered as the value of the variable LAFITE.USER.INFO, an association list of user name and values, the latter being in property list format. Ordinarily you need not be aware of this variable, but you might, for example, want to initialize it with information about many users of a public machine.

## Private DLs

---

Lafite now supports private distribution lists in outgoing messages. A private distribution list (dl) is an indirection mechanism: it allows you to name a possibly large group of addressees without explicitly enumerating them in the header of the message. Unlike a public dl, whose membership is maintained in the mail system's distributed database, a private dl is simply a file in which you have stored the real names. The names in the file must be in standard syntax for the protocol you are using (GV or NS), just as you would type in the header of a message, except that names may be separated with one or more carriage returns, rather than commas, to enhance readability. In addition, the Grapevine dl handler insists that the names be fully qualified, i.e., the Grapevine registry must be included. This is to eliminate confusion in the case where the current sender is in a different registry than the maintainer of the dl. Such a requirement would make NS dl's horrendously verbose, however, so we arbitrarily waive the restriction for them. Thus, be careful if you compose public NS dls.

The names in a Grapevine dl can make use of all the standard Grapevine syntax, which allows you to include comments in the file. For example, you could have a line reading

```
<Carstairs.wbst> "Works with bj on the ACME project"
```

Since NS header syntax does not support such elaborate syntax, comments are not permitted in NS dl files.

The file may be a TEdit file; Lafite strips out the formatting before attempting to parse the addresses in it.

The syntax for a private dl in a message is

```
To: dlfilename::;
```

i.e., the name of the file containing the real names, followed immediately by the two characters colon, semi-colon. In an NS message, this syntax is actually parsed as the fully qualified Clearinghouse name *dlfilename::defaultOrg*, and will appear that way to the recipient, but Lafite treats this kind of name specially. If the file name contains any characters whose syntax is significant—colon, semi-colon, parentheses, and brackets for Grapevine, colon for NS—the file name must be enclosed in double quotes. For example,

```
To: "{FS9:PARC:Xerox}<Carstairs>Mail>Friends";;
```

The file name need not be complete; Lafite fills in defaults to locate the file according to the following two variables:

LAFITEDL.EXT

[Variable]

The default extension of private distribution list files. If a private dl is specified without an explicit extension, Lafite uses this extension when looking for the file. The value is initially "DL".

LAFITEDLDIRECTORIES

[Variable]

List of directories on which private distribution list files may be found. When you use a private dl in a message, Lafite searches LAFITEDEFAULTHOST&DIR, then this list, for the file containing the addressees. The value is initially NIL.

## Grapevine

---

Lafite now does a more thorough job of parsing Grapevine messages according to the RFC 822 specification and objecting to malformed addresses. It also preserves full addresses when composing replies, rather than discarding all the commentary.

\*GV-SHOW-POSTMARK\*

[Variable]

When Lafite retrieves Grapevine mail, there is additional information about the transport of each message that Lafite normally discards as uninteresting, as it is for the majority of messages. When the variable \*GV-SHOW-POSTMARK\* is true, Lafite preserves this information in an extra line in the header of the message. The line reports the authenticated name of the sender, the time at which the message entered the Grapevine system, and the host from which it entered; for example,

GV-Info: Carstairs.pa at 4-Jun-88 23:38:00 from 204#16#

Note that the variable only affects the retrieval of mail, not its display. If the variable is NIL at the time of retrieval, the information is lost forever. A compromise is to set the variable true, but add "GV-Info" to the appropriate filter lists (LAFITE.DONT.DISPLAY.HEADERS, etc.), so that you see it only when you Unhide the header.

## NS Mail

---

The NS mail retrieval code correctly interprets messages that are really returned messages. It now turns such messages into a description of a failed delivery, with a mail server as the apparent sender, instead of just putting a cryptic "TransportProblem" header in the message and leaving the header looking as though the message was from you.

The NS mail watcher no longer maintains a "session" on the mail server. This reduces server load a bit, and means that there is no confusion if another client (either Lafite or Viewpoint) tries to retrieve the mail of a user who happens to be running Lafite on another machine at the same time.

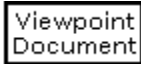
Formatted mail in NS mode is now retrieved more efficiently, by adjusting the formatting information directly, rather than performing higher-level TEdit operations that require doing a Get and Put. In addition to being faster, this avoids problems with messages that cannot be Put properly, such as those containing image objects unknown on the receiving side.

NS mail header fields are displayed in the order specified by the variable NSMAIL.HEADER.ORDER, independent of the order in which the mail system may have delivered them (the new Services 11 servers deliver some headers in an odd order).

Lafite now accepts messages sent in serialized format 3, which it turns out is identical to format 2.

NS Mail now handles "attachments" better. This change was actually made in a special release of NS mail in September, 1987, but is included here for completeness, and to explain the extensions that handle reference objects. Attachments are no longer left in your mailbox to be read later with, for example, Viewpoint. Instead, Lafite retrieves the entire attachment and encapsulates it into an image object that is enclosed as part of the text message, immediately following the header. A typical attachment appears in a mail message as:



Attachment: 

If you click inside the object with any mouse button, you are offered a menu of things you can do with the attachment. The choices vary according to the type of attachment:

- View as text** This brings up a window in which is displayed the raw content of the attachment as ascii bytes. Runs of non-ascii bytes are replaced by nulls to reduce the amount of garbage. Some attachments are utter gibberish, but some, such as Viewpoint documents and Interpress masters, contain sections that are plain text. With this command, you may be able to decide whether you care to do anything further with the attachment. (Sorry, there is no Viewpoint to TEdit converter, nor are there plans for one.)
- Put to file** This prompts you for a file name, and creates a file to contain the attachment. The file must be on an NS file server for this command to be very useful; otherwise, information will be lost. Once the file is so stored, you can retrieve it from Viewpoint and manipulate it just as if you had originally retrieved it as mail in Viewpoint. If you are running in Lyric, you must have the module NSRANDOM loaded for this command to work.
- Send to Printer** This command is only available for attachments that are in the form of an Interpress master. The command prompts you for a printer (which must be one that accepts Interpress, of course), and sends the attachment to it for printing.
- Expand folder** This command is only available for attachments that are in the form of a "folder". A folder is a mechanism for collecting several objects into a single one. The **Expand folder** command splits the attachment up into its component objects, each of which can be manipulated in the same way as a top-level attachment. For example, if the folder contains an Interpress master, you can print it.

If you use the **Put to file** command on a folder, the name component of the file name you type will be treated as the name of a new subdirectory, and the components of the folder will appear as files in that subdirectory. For other types of attachments, **Put to file** (usually) produces an ordinary (non-directory) file.

If the attachment is a Viewpoint "Reference" object, the **View as text** and **Send to Printer** commands apply to the file the reference mentions, not the attachment itself. The **Put to file** command is renamed **Store reference**, to make it clear it applies to the reference object itself, rather than the file being referred to (which there is no point in copying, since Viewpoint can deal with the reference object itself just fine). References to folders also offer the command **FileBrowse**, which brings up a FileBrowser (if you have it loaded) browsing the directory referred to.

Messages containing attachments are otherwise just like formatted messages—you can move them to other folders, and you can forward them (assuming the mail is received by another Lafite recipient and did not have to pass through an information-losing mail gateway).

Lafite now sends plain text messages as "text attachments" rather than "mail notes", unless \*NSMAIL-SEND-MAIL-NOTES\* is true. This avoids a nasty bug in Services 11.3 mail servers and should be completely transparent to users.

There is currently only a crude mechanism for creating your own attachments to send to other users, as follows: The attachment must be a file on an NS file server. To send an attachment, place a line "Attached-File: *XNS Filename*" in the header of your message. To send the file as a reference (this sends only a pointer to the file, not the file's content, so the recipient must have access to your server), use the line "Attached-Reference: *XNS Filename*". You can have only one such line in the header, and the body of the message must be small enough to send as a mail note (less than 8000 characters). Probably the only interesting kind of attachment to send currently is an Interpress master, which both Lafite and Viewpoint recipients will be able to print. To "forward" an attachment you received in NS mail, you can use **Put to File** on the attachment, then compose a message with "Attached-Reference" or "Attached-File" referring to the resulting file. Incompleteness: Most of the attributes of the file are lost when you send the file itself as an attachment, rather than a reference. You cannot send directories as attachments.

---

## Scavenger

Lafite's mail scavenger has been completely rewritten. It now handles most simple cases of bad message lengths by altering the file in place, rather than laboriously copying the entire file to a scratch location. The diagnostic output is more informative. The scavenger is also integrated with the browser, so that if you browse a malformed folder and the browser reports "unable to parse", you are immediately offered the option of scavenging.

You can still scavenge manually by calling `LAFITE.SCAVENGE`, but ordinarily you should never need to do this.

`(LAFITE.SCAVENGE FOLDERNAME ERRORMSGSTREAM FORGET?)`

[Function]

*FOLDERNAME* is the name of the folder to scavenge, either a full file name, or abbreviated as it might appear in the folder menu. *ERRORMSGSTREAM* is an output stream to which to direct diagnostic information; it defaults to the terminal. If *FORGET?* is true, the folder is not added to the folder menu, just as if you had used the **Browse & Forget** command.

The old scavenger entry point `MAILSCAVENGE.IN.PLACE` is obsolete and has been removed.

---

## Miscellaneous

The middle-button **Browse** command has a new subcommand **Browse & Forget**. This command prompts for a file name and browses it with the `:FORGET` option, i.e., it does not add it to the folder menu. Note that this command does not remove an existing folder from the menu, it merely refrains from adding a new one. Also, since it is not in the menu, it will be inconvenient to move mail into it—you have to select "Other Folder" and type the name each time. The command is intended for those times you want to browse some other user's mail folder that you do not plan to look at again.

Browse has another new subcommand **Rename Folder**, which you can use to change the name of a folder. This renames the file(s) supporting the folder and fixes various menus.

Lafite's table of contents code now handles messages containing NS strings in the Subject field. This fixes the bug where browsing such a file would claim that the toc was inconsistent with the folder, throw away the toc file and reparse from scratch. Also, the bug where Lafite reported "Folder is Empty" when discarding a malformed toc has been fixed.

The Lafite message file format has been very slightly extended to permit you to receive messages of up to 99,999,999 bytes (about 100MB!) in length. This means it no longer is forced to split apart messages longer than 99,999 bytes, which did very bad things to large Viewpoint attachments and sometimes caused loss of mail server connection. It also means that Lafite files are no longer exactly compatible with Laurel or Hardy format, in case anyone cares. As mitigation, however, messages that fit in the old format (99,999 bytes or less) are converted back to the old format whenever they are moved (either by **MoveTo** or **Expunge**), so you can arrange for a Hardy-compatible file by moving all messages into another folder (assuming your messages are all short enough).

Lafite now treats folder names as strings, and does its best to (a) ignore alphabetic case within folder names and (b) preserve the case that you type when creating a folder. This means, among other things, that you can now have mixed case in your LAFITEDEFAULTHOST&DIR, and that you can have folder names that are entirely numeric, such as "1186".

Lafite also keeps in its folder menu what the server considers the "canonical" name of each file to be. This means that if your folders include a file not on your default directory with a host not specified by its canonical name, then the next time you browse this folder, Lafite will think it is a new folder, leaving you with a menu containing both the old and the new names. Use the **Forget Folders** subcommand of **Browse** to dispose of such duplicates.

The information about your folders and message forms, formerly maintained in a file named LAFITE.PROFILE on your LAFITEDEFAULTHOST&DIR, is now kept in a file named LAFITE.INFO in a more extensible format. When you first run the new Lafite, if you have no LAFITE.INFO file, it will read LAFITE.PROFILE instead and convert to the new format.

Lafite does a better job now of noticing whether a mail file has been changed out from under you. It checks every time it opens the file, not just after logout. If the mail file has been changed but you have unsaved changes to the browser, Lafite will offer to save your changes if the mail file has merely been appended to. If, however, it has been expunged, so that the messages all live in different locations in the file, this is not possible, and the changes in your browser must be discarded.

When you return from logout and Lafite can't locate the file behind one or more of your browsers, it no longer closes the browser. The next time you try to do anything with the browser, such as display a message, Lafite will again try to locate the file. Thus, if the file server holding the file is down, you need only wait for it to come back up before using the browser, rather than having to rebrowse the folder.

The after-logout code now runs mostly in a separate process, which allows the call to LOGOUT to return faster.

Lafite files are now of type LAFITE, rather than TEXT, since they typically contain non-text content, such as NS characters and TEdit formatting. On any kind of server but an NS file server, type LAFITE is usually coerced to BINARY. You should be careful when copying old mail files (those that might have been written as type TEXT) to a server with a different end of line convention (e.g., between XNS and Unix).

LAFITE.HOST.ABBREVS

[Variable]

A list specifying abbreviations for commonly-used host and directory combinations. This is useful if you regularly work with mail files on more than one directory. Each element of LAFITE.HOST.ABBREVS is a list (*abbreviation fullname*), where *abbreviation* is a string ending in colon (and not containing any other filename punctuation) specifying the abbreviation you want to use, and *fullname* is a string specifying the full host and directory name for which you wish the abbreviation to stand. You can use the abbreviation any place that Lafite prompts for a folder name, and Lafite will expand it accordingly. Conversely, Lafite will replace *fullname* with *abbreviation* when it puts folder names into menus, title bars and such. *abbreviation* can also be a list of abbreviations for the same directory, in which case Lafite will use the first element of the list when constructing abbreviated names, useful if you want the names to sort differently in the folders menu, or make the abbreviation stand apart from the rest of the name.

For example, if LAFITE.HOST.ABBREVS is

```
((("~Pooh~:" "Pooh:") "{pooh/n}<pooh>carstairs>mail>")
 ("Jo:" "{FS8:Parc:Xerox}<Josephine>Mail>"))
```

then if you type the name "Pooh:Active", Lafite will treat it as if you typed "{pooh/n}<pooh>carstairs>mail>Active", but will turn that full name into "~Pooh~:Active" when putting it in the folders menu. Lafite only substitutes for entire directories, so, for example, it would not abbreviate "{FS8:Parc:Xerox}<Josephine>Mail>Old>August" to "Jo:Old>August".

LAFITE.HOST.ABBREVS is one of the variables that Lafite caches, so it will not notice changes to it until you either restart Lafite or use the **Recache** command. And because host abbreviations take the syntactic form of a device field, you cannot use an abbreviation to stand for a directory that includes a device, e.g., {vax}dp1:<Fred>.

## Changes to Programmer Interface

---

Well, actually, there is no documented programmer's interface. Numerous internals have changed; this section lists but a few.

With the new handling of modes, nearly everything surrounding the use of \LAFITEUSERDATA and the record LAFITEOPS (and hence the variable LAFITEMODELST) has changed. POLLNEWMAIL has been restructured; PRINTLAFITESTATUS behaves differently. There is a new hook LAFITENEWMAILFN. When this variable is non-NIL, its value is called, with no arguments, when the Lafite status window notes new mail.

The MAILFOLDER record has changed slightly. The use of its FOLDERDISPLAYWINDOWS field is different. The new field FOLDERDISPLAYREGIONS replaces BROWSERSELECTIONREGION.

The whole family of functions around \LAFITE.BROWSE has substantially changed.

Several functions have been deleted or renamed including LA.REMOVEDUPLICATES, LA.SETDIFFERENCE, PROFILEFILENAME, \LAFITE.MERGE.PROFILES (now \LAFITE.MERGE.NAMELISTS), CHANGEFLAGINFOLDER, \LAFITE.GETMAILFOLDER.

MAKEXXXSUPPORTFORM, when given an address in a-list form, chooses the first address in the list that is for a supported mode. Thus, you can arrange to have Lisp Reports always go to an NS address (assuming NS mail is loaded), by making (NS . "NS Support Address") be the first element of LISPSUPPORT. Special forms that call MAKEXXXSUPPORTFORM can also encourage a particular mode this way without having to switch the mode themselves.

### **Partial List of Fixed or Obsolete ARs**

---

- 267 Easy font changing
- 307 Filter selected fields out of displayed msg?
- 552 Make it easier to change logged in user
- 1239 Recover from "Lafite is confused..."
- 1249 Want automatic GetMail on icon expansion
- 1273 Prompt for GV login if BadPassword error => Declined: use Login
- 1389 Answer command should preserve entire address
- 1644 Private dl's
- 2143 Want detachable display windows that survive Update
- 2421 Preserve case when creating files
- 2433 Use GETBOXREGION to place window?
- 2597 Check that GV headers conform to RFC822
- 3168 Want user-specifiable browser / menu layout
- 3174 List of regions for browser, display, etc
- 3269 Update, GetMail etc should check creationdate / eof and rebrowse if needed
- 4420 LAFITEHARDCOPY . MIN . TOC doesn't work with LAFITEHARDCOPYBATCHFLG
- 4628 addresses lacking a closing double quote break with illegal arg 65536
- 5109 Don't loop if bad NS password
- 5110 Blank name (e.g. in Reply-to) completes to \*:domain:org
- 5111 Want name parser that doesn't add default registries
- 5116 Parser loops forever if header address has unmatched paren or quote
- 5117 Want way of dealing with NS Mail attachments without running Star
- 5121 Close / Shrink menu includes "Expunge" even if no deleted messages
- 5122 Simultaneously watch for both NS and GV mail
- 5125 Case sensitivity in LAFITEDEFAULTHOST&DIR screws up MoveTo
- 5129 Want browser title rearranged to avoid clipping important info
- 5134 Message from self not noted if user not authenticated when folder browsed
- 5137 Want to be able to keep display window open after browser shrunk
- 5300 Answer NS message in GV mode does not give To field
- 5541 Default status window off screen on small 1186 screen
- 5638 Prune headers on forwarded messages
- 6372 LAFITENEWMAILTUNE not on daybreak
- 6260 Blank To / cc line parsed incorrectly
- 6547 Can't put Arpa address in From field
- 6918 Send unformatted dies in COPYBYTES illegal arg
- 7244 Numeric folder names die in string-equal
- 8643 (il:lafite 'il:off) breaks in SHADEITEM, menu = nil
- 8679 Changing modes with Lafite off doesn't reset \LAFITEUSERDATA?
- 8995 When toc discarded, says "Folder empty"

There is a new version of Lafite. For Medley/LispCore users, it is on {Eris}<LispCore>Internal>Library>. There is also a version compiled for Lyric, but substantially less tested, on {Phylum}<LispLibrary>Lyric>New>.

For many users, the highlight of the new version is that GV and NS mode run simultaneously. Details of this and some of the other features and bug fixes may be found in the 15-page file Lafite-Jun-88.IP (same directories).

If you want to load the new version on top of the old, first Quit out of the old Lafite, then load *all* of the following LCOMs from the appropriate directory:

```
LAFITEBROWSE
LAFITEMAIL
LAFITESEND
MAILCLIENT
NSMAIL
LAFITEFIND
LAFITE
```

Of the three files MAILCLIENT, NSMAIL, LAFITEFIND, you need only load those that are already present in your sysout from the old Lafite; I believe all are present in the standard Parc sysout, so all must be reloaded. Note that LAFITE.LCOM must be loaded last. Lyric users: you must have NSRANDOM loaded (it is in the Parc sysout) for full NS Mail functionality.

Rooms users should additionally load {Pogo:AINorth}<Rooms>Medley>Users>NEW-LAFITE-WINDOW-TYPES.DFASL and re-dump their suite. Henceforth NEW-LAFITE-WINDOW-TYPES will be autoloading when their suite is loaded, provided {Pogo:AINorth}<Rooms>Medley>Users> is on DIRECTORIES. There apparently isn't a version compiled for Lyric just yet.

This version of Lafite is incompatible with Lens; Ramana will send a message when there is news on that front. It is also unlikely to be compatible with any other program that thinks it knows about Lafite internals, as many have changed.

If the brave pioneers report no outrageous bugs in the next week or two, these will get moved to the standard release places.

## **Lafite To Do**

Last Edited Sep 4, 1987, vanMelle.pa. Last ar = 8995.

### **Browse/SubBrowse commands**

Delete Folder Command ?

Want Rename Folder command [ar 5140]

Browse empty folder => divide by zero [ar 8166]

### **Browser**

Parse dates to provide uniform date syntax in browser window [ar 2412]

Sort [ar 365]

Redistribute vs. Forward (implies smarter Answer) [ar 1693]

Deferred MoveTo for speed [ar 392]

Parse EOL = CRLF files correctly [ar 730]

Imageobj in message header -> "24-bit extended NS encoding not supported" [ar 4635]

Want icon to be labeled to be more overlappable [ar 1242]

Default the Subject field to first line of message [ar 1828]

Want messages directly addressed to recipient to be highlighted in browser [ar 2441]

Want sub-browsers to allow scanning selected subset of messages [ar 2401]

Want "Get Mail" near "Display" [ar 4263]

### **Browser Marks**

Want to use DEL instead of ESC to abort mark setting [ar 478]

Want to abort mark entry by clicking [ar 1071]

Want multiple flags per message [ar 54]

### **Browser Display**

Scroll by integral number of lines

Want Display after Delete to happen in more cases [ar 2405]

Browser Title include number of messages [ar 2415]

Toc "From:" not show <addr> or defaultregistry

Updating of mark does not draw delete line thru mark [ar 1072]

Want to control when browser chooses to scroll up [ar 907]

Want "End of Messages" indicator in browser [ar 765]

### **Browser Searching**

Search commands [ar 571]

Want selection ops using "In-reply-to" field to jump to other msgs [ar 481]

### **Message Display**

Don't tie up mouse if it will take a while [ar 1387]

Want to be able to abort Display when it's taking too long [ar 2498]

### **Hardcopy**

Hardcopy options (#copies, #sides, font, printer, etc) [ar 941, 952, 2403]

Want Hardcopy forms like Laurel [ar 2439]

Want to be able to cancel pending Hardcopy [ar 2495]

Want batch hardcopy to run in a separate process [ar 5145]

\*\*Don't hardcopy too much at once.

### **Update Command**

Have LOGOUT perform Update [ar 2414]

Maybe maintain "First deleted message" and/or "First changed message" to speed expunge and update.

Accelerator for Expunge bypassing Changes/Expunge menu? [ar 761]

Intercept FILE SYSTEM RESOURCES EXCEEDED during expunge [ar 2337]

Don't leave Lafite in bad state if Expunge aborted [ar 2346]

Want faster update [ar 2427]

\*\*Don't Update shouldn't try to close other files (server down) [14 jan 87]

\*\*When shrink/close with Expunge, etc, shrink immediately, gray the icon. When finished, ungray/close the icon.

### **Retrieving Mail**

Try harder to open connection to distant site [ar 2465]

GetMail breaks with Bad.State.For.Bin when mail has gotten archived [ar 5142]  
 Break on messages longer than 99999 bytes [ar 2345]  
 Choke on very long (9 piece) messages with bad.state.for.bin [ar 4604]  
 Programmatic GetMail [ar 2416]  
 Intercept errors during GetMail (disk full, bad.state.for.bout) [ar 2418, 2348]  
 Programmatic filtering of incoming mail [ar 1385]  
 Want to make it harder to GetMail to wrong folder [ar 3164]  
 Should inhibit Delete et al during GetMail [ar 2445]  
 9305 crash after getmail clicked during shrink/expunge [ar 4027]

## **Grapevine**

Suppress/interpret "Server full" abort messages [ar 429]  
 Say "server full" instead of AllDown when appropriate [ar 479]  
 Appropriate errorset protection against grapevine; bad.state.for.bout [ar 173]  
 Notice messages examined by Ernestine? [ar 187]  
 Notice if messages deleted (discrepancy with what is printed) [ar 1198]  
 BAD . STATE . FOR . BOUT under GV . ADDTOITEM while sending message [ar 2517]  
 Break when retrieving archived mail fails [ar 5795]

## **NS Mail**

Don't break if can't find Clearinghouse [ar 3049]  
 CLOSEF of closed stream when NS mail server not responding [ar 3160]  
 Delivery failed--(SERVICE . ERROR MediumFull) [ar 3223]  
 Delivery dies if mode switched at same time [ar 5131]  
 Space in From field not allowed in Arpanet mail [ar 5139]  
 \*\*Break in ns.openmailbox the first time it is used--(difference n nil) [15apr87]

## **Message Sending**

Require templates be filled in? [ar 2420]  
 Want alternative to Lafite Outbox for getting at old messages [ar 3169]  
 Better programmatic interface?  
 Want settable cc: self [ar 951]  
 Want cc to file instead of self [ar 1240]  
 Want blind cc [ar 5133]  
 Want nicknames [ar 1251]  
 Want faster Abort [ar 1733]  
 Unexplained failures—finishes instantly, or "something went wrong" [ar 1946, 2053]  
 If break under COERCETEXTOBJ, Date/From not discarded [ar 2706]  
 Don't detach all windows—let TEdit do it.  
 Want composition window to shrink during delivery [ar 5138]  
 Don't top delivery window so much [ar 7944]  
 Outbox confused: new msg in first slot, others cleared [ar 8075]  
 \*\*\SENDMESSAGE . RESTARTABLE takes CAR of string [Pavel 6jul86]  
 \*\*Reshaping delivery window during delivery gets ARG NOT MENU [24 Jul 86]  
 \*\*Abort button has a scroll bar! [27 May 86]  
 \*\*When delivering, shrink delivery window

## **Answer**

Answer: want command to yank answered msg into buffer [ar 5123]  
 Answer command for multiple selection [ar 2406]

## **Quit/SubQuit**

Want to be able to Quit Lafite without breaks in odd situations [ar 2521]

## **Mail File Maintenance**

Partition mail folder menu by directory and subdirectory [ar 1130]  
 Require confirmation when forgetting folders, forms [ar 2480]  
 Don't try to access {dsk} files on machine different from where they first lived [ar 2442]

## **Maintain**

Want wildcarded facility for finding names in MAINTAIN [ar 3556]  
 Want Add Forwarding command [ar 5120]



Should make it harder to add invalid names to DLs [ar 5143]

Want command to locate all dl's containing user [ar 5149]

## Mail Scavenge

MailFileScavenger [ar 623]

Offer to invoke scavenger on parse failure [ar 4213]

If parse fails, provide partial use?

Win if file ends in extra nulls

MP9318 when the final rename causes "File System Resources Exceeded" [ar 3836]

## Documentation

LAFITEFIND

## Other

Confusion from Idle logins [ar 4411]

Better recovery from file servers down [ar 4250]

Want status window not to resize [ar 5130]

Want MailFolder icon that stacks better [ar 2404]

Conflict between Lafite and BIG [ar 2718]

Shouldn't break under \LAFITEDEFAULTHOST&DIR when server down [ar 3166]

Want Lafite to close file if it's idle for long enough [ar 3162]

Want better error messages when profile vars (e.g. font) set badly [ar 2520]

Better recovery from lost file server after logout [ar 1388]

One-time font change for display or hardcopy

use { TEMP }

Edit Message [ar 2428]

Want MCI interface [ar 3104]

Command to put backtrace into message window [ar 614]

Protect initialization better from Hardreset [ar 2083]

Want programmers' interface to Mail and Lafite [ar 2462]

Add background menu command to turn Lafite on [ar 2497]

Closing a folder when file disappears should flush pending commands [ar 2456]

Browser got --- --- [0 chars], messages maybe lost [ar 3253]

Add font variables to FONTVARS [ar 6697]

Profile confusion when you move your default dir [ar 7244]

\*\*Extend info in profiles [MikeDixon, 19Jan87]

\*\*Reshaping new LafiteStatusWindow gets divide by zero in RESHAPEALLWINDOWS [10 Jun 86, 7jul86]

\*\*Race: Move to new folder vs. Browse of that folder [1 Aug 86]

\*\*Race: Move vs Update's choice of expunging or not [7 sep 86]

\*\*Notice Files doesn't understand fs aliases [29 sep 86]

\*\*Want number of messages on browser icon [12 Nov 86] ({qv}<briggs>lisp>lafitefoldericon)

\*\*Cleanup Folders message not scrolled to be visible [3 Sep 87]

\*\*LA . LONGFILENAME should handle pathnames [Rao, 19 Feb 88]

## Not mine

Display should be faster [ar 60]

Viewpoint mail notes with NS characters read wrong [ar 4667]

Documentation: need load MAILCLIENT [ar 8282]

## Browse/SubBrowse commands

- File name prompt should allow spaces in name [ar 4270] 6/86
- Don't add new name to known folders if disconfirmed [ar 5113] 6/86
- Want to enumerate all mail files in a directory [ar 5118] 6/86
- Browse Laurel File broken [] 6/86

## Browser Display

- Want all browser prompt messages to go thru common routine [ar 3161] 6/86

## Browser Searching

- Find Previous All when current selection satisfies search says "found in 1 messages" if no other instances found [ar 2413] 6/86
- "Find Related is unreliable" [ar 3190] 6/86
- Want to be able to Find messages from self [ar 5115] 6/86
- Search for Mark [] 6/86

## Message Display

- When "No more messages", at least topw the old msg [ar 5170] 6/86

## Hardcopy

- Error if close browser before hardcopy finishes [ar 2784] 6/86

## Update Command

- Don't write scratch file on {DSK} if there isn't space [ar 5124] 6/86
- Want Expunge command not to move selection pointer [ar 5141] 6/86

## Retrieving Mail

- Graphical feedback during GetMail [ar 1466] 6/86

## Grapevine

- SHOULDNT in MS . RETRIEVEOPERATION [ar 2461] 6/86
- Parse recipient names with respect to actual sender's registry, not DEFAULTREGISTRY [ar 5144] 6/86

## Message Sending

- Separate formatting info [ar 426]
- SendMail with Lafite off breaks under ATTACHWINDOW [ar 2985]
- SendMail with Lafite off and mode not set fails [ar 3157] ✓
- Error if have Image Obj in private form [ar 3163] ✓
- LAFITE . SENDMESSAGE breaks attempting to print to nonexistent prompt window [ar 3165] ✓
- Illegal arg if you send as plain text a message containing imageobj's [ar 1244]
- Document LafiteSupport et al variables, point elsewhere if appropriate [ar 2619] ✓
- NIL displayed in outbox when no subject [ar 4180] 6/86
- Better icon for unsent mail; include Subject [ar 5112] 6/86
- Remove BEFOREEXIT prop if delivery aborted [ar 5146] 6/86
- Formatted messages should not say "Format: TEdit" [ar 5171] 6/86

## Browser

- close, button "hardcopy only" doesn't close [ar 1838] ✓
- Changing mark character should imply Update needed [ar 2193] ✓

## Quit/SubQuit

- Quit should not flush unsent mail [ar 5135] 6/86

## Other

- LAFITEMODE dies with arg not menu NOBIND if Lafite not on [ar 2890, ✓ 3003] ✓
- ↑E out of NS Lafite password prompt gives 9305 [ar 3158] ✓
- Status window too narrow for some "not responding" messages [ar 3159] ✓
- Quit dies with udf ##CLOSE## [ar 2831] ✓
- Want protection against LAFITEMAILWATCH process being killed [ar 2339] ✓
- Multiple LAFITEMAILWATCH processes remain when more than one Lafite user uses the same machine [ar 3310]
- Want TEDITPROPS arg to \SENDMESSAGE [ar 2387] ✓
- Saved forms should retain formatting info [ar 2407]
- Save Form should not make duplicate names in menu [ar 2482]

"m" character marks should be moved [ar 2422]  
 Message with image objects should automatically be sent formatted [ar 2426]  
 Udf CLOSEMAILFOLDER under Browser Laurel File [ar 2429]  
 "unrecognized courier error message - 0" while sending NS mail [ar 2449]  
 Glitches in prompt for filename in Save Form [ar 2425]  
 Lafite ignores profile if file server unresponsive; later smashes it [ar 5119] 6/86  
 Default Status Window position inappropriate for 15" display [ar 5147] 6/86

## **Fixed in 6/88 Lafite**

### **Browse/SubBrowse commands**

Case sensitivity in LAFITEDEFAULTHOST&DIR screws up MoveTo [ar 5125]  
 When toc discarded, says "Folder empty" [ar 8995]

### **Browser**

Want user-specifiable browser/menu layout [ar 3168]  
 Want browser title rearranged to avoid clipping important info [ar 5129]  
 Message from self not noted if user not authenticated when folder browsed [ar 5134]  
 \*\*Want to be able to specify icon position.  
 \*\*Want browser title to be short if possible. [19jan87]  
 \*\*Want to be able to shift-select from browser [Shrager 4Mar87]

### **Browser Display**

Want detachable display windows that survive Update [ar 2143]

### **Message Display**

Easy font changing [ar 267]  
 Filter selected fields out of displayed msg? [ar 307]  
 Use GETBOXREGION to place window? [ar 2433]  
 Want to be able to keep display window open after browser shrunk [ar 5137]

### **Hardcopy**

LAFITEHARDCOPY . MIN . TOC doesn't work with LAFITEHARDCOPYBATCHFLG [ar 4420]

### **Update Command**

Update, GetMail etc should check creationdate/eof and rebrowse if needed [ar 3269]  
 Close/Shrink menu includes "Expunge" even if no deleted messages [ar 5121]

### **Retrieving Mail**

Want automatic GetMail on icon expansion [ar 1249]  
 LAFITENEWMAILTUNE not on daybreak [ar 6372]

### **Grapevine**

Prompt for login if BadPassword error [ar 1273] => Declined: use Login  
 Answer NS message in GV mode does not give To field [ar 5300]  
 Check that headers conform to RFC822 [ar 2597]

### **NS Mail**

Don't loop if bad password [ar 5109]  
 Want way of dealing with NS Mail attachments without running Star [ar 5117]  
 Simultaneously watch for both NS and GV mail [ar 5122]  
 \*\*Don't hold onto session [19aug86]  
 Blank name (e.g. in Reply-to) completes to \*:domain:org [ar 5110]  
 Blank To/cc line parsed incorrectly [ar 6260]

### **Message Sending**

Cleanup mess of LAFITE [CURRENT] EDITORWINDOWS, MESSAGEFILE prop  
 Private dl's [ar 1644] (see {QV}<Briggs>Lisp>LafitePrivateDl)  
 addresses lacking a closing double quote break with illegal arg 65536 [ar 4628]  
 Parser loops forever if header address has unmatched paren or quote [ar 5116]  
 Prune headers on forwarded messages [ar 5638]  
 Can't put Arpa address in From field [ar 6547]  
 Send unformatted dies in COPYBYTES illegal arg [ar 6918]

Signature (For From, for end of msg).

\*\*Want to answer the "formatted" question automatically [12 apr 86]

## Answer

Answer command should preserve entire address [ar 1389]

Want name parser that doesn't add default registries [ar 5111]

## Quit/SubQuit

Make it easier to change logged in user [ar 552]

(il:lafite 'il:off) breaks in SHADEITEM, menu = nil [ar 8643]

## Mail File Maintenance

Preserve case when creating files [ar 2421]

Can't have numeric folder names

## Other

Recover from "Lafite is confused..." [ar 1239]

List of regions for browser, display, etc [ar 3174]

Default status window off screen on small 1186 screen [ar 5541]

Numeric folder names die in string-equal [ar 7244]

Changing modes with Lafite off doesn't reset \LAFITEUSERDATA? [ar 8679]

Parse NS chars better.

\*\*Proclaim profile variables special [Pavel, 24 jan 87]

\*\*checklafitemailfolders gets non-numeric arg NIL in IDIFFERENCE [3apr87]

\*\*Want programmatic interface to browsing, complete with window/icon positioning and shrinking  
[Gregor 29apr87]

\*\*Prompt for icon position on mail folders

\*\*After logout, don't close missing folders

---

---

## UNIXMAIL

---

---

By: Bob Bane (Bane.mv@envos.Xerox.com)

### INTRODUCTION

UNIXMAIL is a new mail sending and receiving mode for Lafite. It sends mail via Unix hosts using the SMTP mail transfer protocol and can receive mail either by reading a Unix mail spool file or by calling the Berkeley mail program.

### INSTALLATION

Turn Lafite off, load the file UNIXMAIL, make sure UNIXMAIL is configured appropriately (check the settings of the variables below, and make sure any other modules UNIXMAIL may need are loaded), then restart Lafite. If you are running Lafite on a machine that is isolated from the Xerox mail environment, you will probably want to set the variable LAFITE.USE.ALL.MODES to NIL and call (LAFITEMODE 'UNIX) before you turn Lafite back on; this makes Lafite send and receive mail in Unix mode only.

### CONFIGURING

See **SENDING MAIL** and **RECEIVING MAIL** below for the exact meanings of the variables you will be asked to set. **NOTE:** these variables are checked by Lafite at authentication time only. To change UNIXMAIL's behavior without stopping Lafite, change the variables and then make Lafite re-authenticate (middle-button on the **Quit** Lafite menu, then select the **Just re-authenticate** sub-menu entry under **Login**).

D-machines:

UNIXMAIL.SEND.MODE must be set to SOCKET and UNIXMAIL.SEND.HOST must be set to the name of a TCP host that will accept SMTP connections. UNIXMAIL.RECEIVE.MODE must be set to SPOOL and UNIXMAIL.SPOOL.FILE must be set to the pathname of your Unix mail spool file.

Unix-based emulators:

The default values of UNIXMAIL.SEND.MODE and UNIXMAIL.RECEIVE.MODE (PROCESS and SPOOL, respectively) will work if you normally send and receive mail from the machine where Medley is running. If your machine is on a non-trivial mail network, your machine may be sending and retrieving your mail from a remote machine without your knowledge; check all the variables below carefully if you have problems.

### OTHER MODULES YOU MAY NEED

UNIXMAIL may need other library modules to work. The modules needed vary depending on what hardware you are using:

D-machines:

TCP is mandatory for Unix sending and may be used for Unix receiving, NFS is optional for Unix receiving

Unix-based emulators:

one of TCPOPS or UNIXCOMM is mandatory for sending

### SENDING MAIL

UNIXMAIL can send mail in one of two ways, depending on the setting of UNIXMAIL.SEND.MODE:

UNIXMAIL.SEND.MODE [Variable]

If its value is the atom `PROCESS`, `UNIXMAIL` will send mail by doing SMTP with a Unix process-stream, normally running `/usr/etc/mconnect`. This option only works with Medley running on one of the Unix-based emulators.

If its value is the atom `SOCKET`, `UNIXMAIL` will send mail by doing SMTP with a TCP host. For this to work, an appropriate version of TCP must be loaded: either the TCP library module for D-machines or the TCPOPS library module for emulators that support it.

`UNIXMAIL.SEND.MODE` defaults to `PROCESS`.

Each of these send modes can be configured as well:

`UNIXMAIL.SEND.PROCESS` [Variable]

When `UNIXMAIL.SEND.MODE` is `PROCESS`, the value of this variable is the program run to create the SMTP process-stream. Initially the string `"/usr/etc/mconnect"`; if your machine automatically forwards your mail to another host, you may have to put that host's name into the command, e.g. `"/usr/etc/mconnect fred"`

`UNIXMAIL.SEND.HOST` [Variable]

When `UNIXMAIL.SEND.MODE` is `SOCKET`, the value of this variable is the name of the host `UNIXMAIL` will attempt to contact via TCP to open an SMTP stream over socket 25. Initially `NIL`; on a Unix-based emulator this means to try the machine Medley is running on. This variable must be set when running on a D-machine.

## RECEIVING MAIL

`UNIXMAIL` can receive mail in one of two ways, depending on the setting of `UNIXMAIL.RECEIVE.MODE`:

`UNIXMAIL.RECEIVE.MODE` [Variable]

If its value is the atom `SPOOL`, `UNIXMAIL` will receive mail by reading a Unix mail spool file.

If its value is the atom `MAILER`, `UNIXMAIL` will receive mail by running a Berkeley mailer as a Unix process-stream, normally `/usr/ucb/mail`. This option only works on Medley running one of the Unix-based emulators, and is a bit slower than `SPOOL` mode; it is primarily useful when you wish to occasionally switch between Lafite and the Berkeley mailer.

`UNIXMAIL.RECEIVE.MODE` defaults to `SPOOL`.

Each of these receive modes can be configured as well:

`UNIXMAIL.RECEIVE.PROCESS` [Variable]

When `UNIXMAIL.RECEIVE.MODE` is `MAILER`, the value of this variable is the program run to create the SMTP process-stream. Initially the string `"/usr/ucb/mail -N"`; the `-N` means to not print any banner or read any initialization file on starting the mailer.

`UNIXMAIL.DONT.RECEIVE.STATUS` [Variable]

When `UNIXMAIL.RECEIVE.MODE` is `MAILER`, the value of this variable is a set of message status letters; `UNIXMAIL` will leave behind any message whose status is included. Initially `"`, which means to read all messages regardless of status; another useful value would be `"O"` which means leave old messages behind.

`UNIXMAIL.SPOOL.FILE` [Variable]

When `UNIXMAIL.RECEIVE.MODE` is `SPOOL`, the value of this variable is the file `UNIXMAIL` will receive mail from. Any time this file exists, Lafite will say you have new mail; when Lafite gets mail from this file, it will read all messages in the file and then set its size to zero. Initially `NIL`; on a Unix-based emulator this means to try the file `"{UNIX}/usr/spool/mail/username"`, where `username` is the value of `(UNIX-USERNAME)`. To access a Unix mail spool file from a D-machine, it will probably

be necessary to load and configure either the TCP or NFS modules and then set UNIXMAIL.SPOOL.FILE appropriately.

*The great art of living easy and happy in society is to study proper behaviour, and even with our most intimate friends to observe politeness; otherwise we will insensibly treat each other with a degree of rudeness, and each will find himself despised in some measure by the other.*

Boswell, *London Journal* (1762)

---

## Foreward

---

This appendix is an edited version of "Message System Mores," chapter 6 of the Xerox *Laurel Manual*, by Douglas K. Brotz, and the essay "Message System Mores" that Brotz published in *ACM Transactions in Office Information Systems*, Vol 1, No. 2. The material that appeared only in the ACM journal is copyright 1983 by the Association for Computing Machinery, Inc.

Douglas Brotz was a member of the team at the Palo Alto Research Center (PARC) that designed Laurel, which is an electronic message system similar to Lafite that was written for the Xerox Alto. Through his involvement with Laurel, Brotz discovered patterns of electronic message system behavior that may apply to Lafite users. He also developed some rules for appropriate message system behavior, i.e., message system etiquette. Because many Laurel and Lafite users have found this essay helpful, we have edited it for inclusion here, making the references appropriate for Lafite and deleting information that appears elsewhere in the Lafite manual.

---

## Introduction

---

This is an essay on manners, in particular, message system manners. Electronic message systems provide a new mode of communication that, while offering convenience, speed, and reliable delivery, also opens channels that may be abused. At the Xerox Palo Alto Research Center, we have designed and implemented an electronic message system that has quickly spread throughout the Xerox Corporation. Through its use, we have discovered many patterns of message system user behavior that appear to apply to electronic message systems in general rather than to the particular system that we built. The focus of this essay is not on the features of our system, but on observations of user behavior in the electronic mail environment in an effort to spread understanding of this new medium and to instruct users in proper behavior.

The contents of this essay may be divided into roughly two kinds, objective observations of message system social phenomena and



definitely biased suggestions of standards. The opinions expressed herein are solely those of the author. These opinions are not based on scientific studies or samples, but rather on certain intuitive feelings that have evolved through a close association with our system since its inception.

---

## Communication Patterns

---

Part of the evolution of a society is the structure within which its members communicate. Face-to-face communication, both spoken and through gestures, has been with us for a very long time. Written communication and telephone communication have been employed for a substantially lesser amount of time. Nevertheless, these modes of communication have been around long enough to have developed certain standards of conduct and a framework in which reasonable communication can take place.

The electronic message medium has existed for a much shorter period of time, perhaps 20 or so years. (I am purposely ignoring telegraphic communication, which has very different characteristics due to its long delays and high cost.) Electronic message systems on personal computers have been available for even less time, probably less than 10 years. In this time, standards of electronic communication have not yet had time to mature, so we are still groping toward a workable electronic-messaging society.

In any of the mature communication media, each society places limits on what is considered acceptable behavior. Vulgar language or gestures are generally frowned upon in face-to-face communication, except in smaller sub-societies in which this mode of behavior is necessary for group membership. Shouting at close range is similarly considered to be in bad taste. Methods of dealing with such behavior in face-to-face communication run from mild rejection of the speaker to complete avoidance of that speaker in the future. As the number of human societies is large, and each has had much experience with this means of communication, the means employed for dealing with such situations are quite varied. Within each group, however, the methods used can be quite effective in stifling unwanted behaviors.

I will list several kinds of situations that arise in the electronic message medium and means for dealing with them. Where possible, I will draw parallels to other more traditional modes of communication to illustrate acceptable manners. In addition, I will try to point out the ways in which communicating via electronic mail is different from the traditional communication media, and how this modifies the problems to be dealt with.

---

## The Wrong Number

---

We all have dialed wrong numbers and received calls from people who have dialed wrong numbers. The protocol for handling such situations is simple, and arises naturally as a result of the way in which standard phone calls are initiated. A typical wrong number dialog may be as follows:

Callee: Hello.

Caller: Hello. May I speak to John?

Callee: There is no one at this number by that name. I believe you have the wrong number.  
 Caller: Oh. Isn't this 555-1234?  
 Callee: No, it isn't. (And sometimes . . . ) This is 555-4321!  
 Caller: Thank you. I'm sorry to have bothered you.

In postal communication, receiving misaddressed mail or mail for a former resident who has moved is akin to the telephone's wrong number. The post office's suggested remedy is for the recipient to line out the address and remail the letter. The post office will then attempt to forward the letter to the correct address, deliver it to the proper address, or return the letter to the sender.

Note that in both of these situations, it was not necessary to begin the actual conversation or open the letter. Enough information is exchanged at the outset to determine if the parties in the communication are the correct ones. This is usually not true when communicating via electronic mail.

In electronic message systems, it is seldom the case that a message sent to a particular name is actually delivered to a recipient with a different name. A different situation is (unfortunately) common when a recipient has a popular name. The problem is that several people may have the same last name, and some electronic message systems do not have convenient facilities for mapping a person's actual name into that person's message system name. Thus, a person named Doe may receive mail for ADoe, BDoe, etc. Here, the original error is committed by the sender, who did not consider that ADoe's message system name was actually ADoe, but just assumed that it was Doe.

The parallel to this situation in the telephone medium is actually a bit more elaborate than the dialog given above. It is more like:

Callee: Hello.  
 Caller: Hello. Is Johnny there?  
 Callee: Hold on, I'll get him.  
 John: Hello?  
 Caller: Hey Johnny, let's boogie on down to the hoedown.  
 John: Who is this?  
 Caller: Come on buddih, this is good old Bodine!  
 John: I don't know any Bodine.  
 Caller: Oh. Ain't this 555-1234?

and so on. Notice that in this case a partial name match has occurred, and it is only later in the conversation that one of the parties discovers that something is awry. In the electronic mail case, it is nearly always the case that the message must be at least partially read to determine that it has reached an incorrect recipient.

This situation can be (and has been) handled in several inappropriate ways. First (and worst), the incorrect recipient can just ignore the message. No one gains through such inaction. Second, the incorrect recipient may send a response to the sender of the form "Stop sending me this trash!" This is a bit more helpful, but not quite the best that can be done. Third, the incorrect recipient may send the correct recipient a message of the form "Tell your senders what your name is!" This is not even as good as the previous response, as a message system user cannot know all possible senders.

Proper consideration by all involved can alleviate the "wrong number" syndrome considerably. Senders of messages should know their recipients. When sending a message, if you are not sure of a person's message system name, look it up. At Xerox PARC, the phone list has everyone's message system name correctly listed. Perhaps other organizations should do the same, and eventually a message-system-wide "white pages" may be published. Such lists help, but not if the senders don't use them.

A message addressed to an individual tends to be more important than a message addressed to a distribution list, in that a reply from an individual is expected more than replies from anonymous members of distribution lists. The names contained in distribution lists are usually correct, so there is generally no misdelivery problem. However, senders type the names of individual addressees for important messages directly. Thus, when there is a misaddressed message, it is generally an important one.

When you realize that a message is not for you, use the Forward command to send it back to the sender along with your polite comment that the message has reached a "wrong number." Forwarding the message back is important, as the sender may not have a copy of that message any more. Once you have determined that you have received a "wrong number" message, *stop reading it*. A message sent through the message system may have personal material, and it is none of your business to peruse the entire message. (Many users who typically dispose of their received mail at a rapid clip take great delight in reading every last character of a misaddressed message—indeed, they consider it their solemn duty to do so.) It is for this reason that I do not suggest forwarding the message to the proper recipient. Determining who is the proper recipient is the job of the sender. It is presumptuous to believe that you know who the proper recipient is; you may actually forward the message to yet another incorrect recipient. Besides, determining the correct recipient may require reading more of the message than you ought to read. (If you think you know the message system name of the correct recipient by the time you realize that you are not the correct recipient, then you might include that name in your short covering note back to the sender. However, the mistaken sender should not expect correct identification of the intended recipient, just as he or she would not expect it in the telephone or postal mail systems.)

Some further points to consider are these. The "wrong number" mishaps generally happen to people who have common names and whose system names are exactly their last names. The honor of having one's system name be exactly one's last name is generally historical ("I was the first Doe hired here, therefore I'm entitled to be Doe.PA forever!") A reasonable solution would be that our system administrators ensure that no user has a name that is a suffix of another's, so that when ADoe arrives, then Doe has his or her message system name changed to BDoe, or whatever. In this way, the existing message system facilities will catch messages sent to Doe and return them as having been sent to a nonexistent name, at which point the sender can look up the correct message system name.

---

## Rudeness and Vulgarity

---

The electronic mail medium joins several disparate properties of other communication media in an interesting way. The display of mail on a personal computer is a rather personal experience. Certain feelings of privacy and ownership pervade a personal computer user's relationship with his or her machine. Thus, the process of reading one's own electronic mail includes many of the personal aspects of face-to-face communication.

On the other hand, sending electronic mail is much more impersonal. The recipient is not present, and nearly none of the social strictures that govern one's face-to-face communication are present. The sender is also able to speak his or her piece completely, without any intervening exchanges with the recipients that might moderate the entire business. This situation is enhanced

when the recipients are not named directly, but are addressed indirectly through an impersonal distribution list. This imbalance in attitudes between sender and recipient has wide-ranging consequences.

One obvious consequence of this imbalance is that opinions expressed and the language used to express them in messages can be wildly inappropriate to the customs and expectations of the recipients of such a message. A reader may justifiably feel slapped in the face by a message he or she considers to be in extremely bad taste.

An interesting feature of the most annoying messages is that they tend to come from some "other" part of our messaging community. Julian Orr at PARC has observed that the most serious exchanges of antagonistic messages in our network have occurred shortly after some previously isolated message communities have joined. When the two societies meet and exchange messages, for some period the tone of the "other" community's messages has offended members of "our" community. After an adjustment period, the two communities come to some understanding and establish norms for their intercommunication. This understanding typically involves identification of subjects whose discussion will cease to cross community boundaries.

The development of the junk mail lists in our Palo Alto and El Segundo, California registries illustrates this point. Both communities established Junk<sup>^</sup> distribution lists for people interested in any for sale ads, announcements, random comments, etc. A slow link between these two locations was replaced by a faster one, and the volume of message traffic between the two communities increased dramatically. The two lists, Junk<sup>^</sup>.PA and Junk<sup>^</sup>.ES, were combined into an AllJunk<sup>^</sup> list. However, even though the stated purpose of both Junk<sup>^</sup> lists was "anything goes," many PA registrants felt the ES junk mail was beyond the bounds of good taste. In other words, "their junk is much worse than our junk." A majority of the original members of these lists withdrew from the Junk<sup>^</sup> lists, and many splinter lists developed, ranging from Whimsy<sup>^</sup>.PA ("lighthearted mail for PA") to @CrankMail.DL (a widely used private list in El Segundo). This is a history of two communities that seem to have similar characteristics. The implications for new message networks linking quite disparate communities are that more serious problems are likely to develop.

When rebuked for inappropriate behavior, errant senders have been known to say "I didn't intend it that way!" This is not good enough. The damage has already been done. The only remedy is for senders to think about what they are saying and to whom they are saying it. The message system to date has been fairly unrestricted. Only as long as the society of message system users practices self-restraint will such a freewheeling communication medium be tolerated. There are several means of applying institutional censorship to the message system traffic, means that we hope will never need to be implemented.

---

## Message System Costs

---

Many of the problems associated with improper use of the message system are exacerbated (caused?) by the lack of charging for message system usage. In nearly all other modes of communication, "sending a message" implies a certain cost (or risk), which rises with the number of recipients that are being reached. Free speech is, in this sense, not free at all. Certainly in a free society, one can say what one pleases, but not without

paying for the means to say it. Let me illustrate this with some examples.

In nearly every communication medium, costs for the use of that medium are borne by the sender of messages. Postal mail requires the sender to pay for a stamp for each copy of a message that is sent. Telephone service is charged to the originator of calls, and each call (in general) goes to only one recipient. Broadcasting messages via radio or television requires a large investment on the part of the sender. The costs of printing handbills or posters are likewise borne by their authors. Public speeches, if they are to reach a large audience, require use of sound systems, etc., that are paid for by the speaker.

It may be argued that recipients do pay some of the costs for using some of these systems. However, these costs (the price of a radio receiver, basic telephone service, etc.) are generally constant; they do not increase as received message usage increases. A receiver's cost for electronic mail is similar in this respect in that the cost of a workstation on which the message system runs is borne by the receiver.

Some other modes of communication do require explicit payment by the receiver. Commercial films, books, magazines, and records fall into this category. However, publication of these materials does involve a substantial financial risk. Material that is not likely to be well received is seldom published, and when it is, large costs are often incurred by the publisher.

Electronic mail as it is usually implemented has a very different cost structure. The cost for a sender is minimal. It essentially consists of the time it takes to compose and send a message. If time is considered the major cost factor, then it is the recipients who pay dearly for the messages they receive. When the amount of time each recipient spends on a message sent to a large distribution list (even if a quick scan of part of the message followed by a Delete), is summed over all recipients, this is easily much more than the time consumed by the sender of that message.

While we would like to keep the free structure of a message system, where any user can send any message to any other users, this freedom must be used with some care. When electronic message systems become widespread, they will undoubtedly change their cost structures to match those of the more traditional communication systems.

---

## Unsolicited Mail

---

The existence of large public distribution lists in our message system makes it easy for a sender to reach a very wide audience. Each distribution list has a distinct purpose, e.g., lists of people interested in particular topics, lists of employees in certain organizations, lists of members of particular projects, etc. Some lists are used primarily to keep track of all users of the message system. These include such lists as AllPA^.PA, AllES^.ES, etc., which contain the names of all individuals in those particular registries. There are also some lists maintained on a purely geographical basis, e.g., PaloAlto^.PA, which lists all message system users in Palo Alto, California. This is not necessarily the same as AllPA^.PA, which includes people in the PA registry, but who may not actually work in Palo Alto.

The audiences addressed by these lists should not be considered captive audiences for all users of the message system. At Xerox PARC, the purpose of any distribution list may be discovered by

any user by running the Maintain program. Although all lists are (currently) available for use by any message system user, many lists, e.g., AllN^.N where N is a registry name, should not be used by anyone who doesn't have a very good reason for doing so.

Many distribution lists exist for the enjoyment of their members who wish to receive items of interest to them. One should feel free to send an announcement of an upcoming musical event in Northern California, for instance, to Music^.PA. Such a message is quite inappropriate to send to AllPA^.PA, PaloAlto^.PA, etc. There are lists of message system users who have agreed to suffer through any and all messages. These lists (Junk^.PA, various @CrankMail.DL files, etc.) are the only lists to which ridiculous messages may be sent without incurring the justifiable wrath of message system users.

A message system user should understand when a message is appropriate to send to all people in his or her work group. Social values are different in different locations, and the members of each group should understand what they are. It has been observed that messages that are sent to audiences wider than the sender's immediate group are the ones that cause the most trouble.

Unfortunately, unsolicited messages have continued to be sent to inappropriate lists. Examples of inappropriate messages for standard organizational or geographic lists are:

"Does anyone know how to get my Alto fixed?"

"This is to let everyone in the message system world know that my phone number has changed."

"I want everyone to know that I really like my roofing contractor."

I'm sure that each user of the message system can recall some other similar gem. The following sections explore some of the consequences of unsolicited mail.

---

## The Chain Reaction

---

To add insult to injury, after some piece of particularly ridiculous mail has been broadcast to an inappropriate audience, it invariably follows that some recipients cannot control their urge to make even bigger spectacles of themselves by sending their two cents' worth to everyone who received the original nonsense. While the original event is thought by many message system users to be annoying, the latter is considered to be downright stupid. Remember that once you push the Deliver button and watch the last chance to cancel fade away from your screen, there is no way to erase your comments from the collective memory of your peers.

I would like to list some of the typical responses that have been sent not just to the original perpetrator, but to the entire list of victims.

"Your message is inappropriate to send to all these good people."

"If you don't like junk, then get off Junk^."

"How do I get off Junk^?"

and, my favorite,

"Do you realize that if all of us replied to all of us (as I am doing right now) that the number of messages that would be sent would exceed the number of atoms in the known universe . . . ."

It is my opinion that bombarding only the original sender of a ridiculous message with equally nonsensical replies is poetic justice.

Use the Reply-To field to counteract the chain reaction phenomenon. However, there are situations in which replying to the entire list of original recipients is appropriate. In these cases, send the message without a Reply-To field, so that recipients who use Answer will get forms with all recipient names and lists included as recipients.

One final note on this topic. Although Lafite provides these mechanisms to help break chain reactions, the ultimate responsibility for messages sent lies with their senders. Always check the list of recipients in any message you are about to send.

---

## Off-the-Record Responses

---

There are many situations in which a user submits a question to a wide audience, say to a distribution list of people interested in such questions, and indicates that he or she will collect responses and later make them public. This is a most reasonable thing to do, and it helps to reduce the chain reaction effect. Be sure to include a Reply-To: >>Self<< field when performing such services for your audience.

A note of caution is in order here. Messages should be considered private unless otherwise indicated. If your intention is to publish the responses, then by all means make that intention clear in the same message that poses the original question. If your message did not make that intention clear, and you decide that you would like to publish the responses, then follow up on each response asking whether you may do so.

If the intention to publish responses is clearly indicated in the original message, then publication of any response is fine, as long as that response does not explicitly mention that it should be considered private.

---

## Masquerading

---

On occasion, people have received messages from fictitious senders, or even worse, from someone masquerading as another real message system user. This is a most serious breach of message system etiquette, and should be considered so by all message system users.

A fictitious From field is legitimate when a valid Sender field is included. For instance, messages that are properly signed with an organization's name, say "The Lafite Group," may be sent by explicitly typing a "From: The Lafite Group" line in the message header. Lafite will notice that a From field is already there, and it will include a Sender: *User Name* line in the delivered message instead of its usual From: *User Name* line. Any time you receive a message that has a strange From field, you may check the Sender field for the actual sender.

By a "masquerader" I mean someone who subverts the normal mechanisms embedded in the standard message system programs to send messages of dubious value, without having his or her name

appear in such messages. This action is possible not only in electronic message systems, but in other more traditional communication media as well. Masquerading as another may be a criminal act when committed using traditional communication media, with penalties specified in laws that prohibit libel, slander, and fraud. Other situations, such as telephone “breathers,” are similarly outlawed.

Masquerading is the most serious social problem that we deal with in our message system. It occurs very rarely (three times to my knowledge), but when it does, our message system administrators make every effort to discover the perpetrator. The consequences may be serious, and the discovered perpetrators have all apologized publicly. I quote without reference from one such apology made by a masquerader who saw the error of his ways: “It is clear that any abuse of the message system, however lighthearted the intent, has repercussions far beyond what one might expect.”

At this time, I do not know of any court cases involving libel, slander, etc., in an electronic mail context. Such cases are sure to arise when electronic mail does become more widespread. Masquerading in the message system is not cute or clever. Don't do it.

---

## **Wizards Versus Naive Users**

---

This section is addressed mainly to the wizards who should know better. The population of message system users covers a broad range, from those who have knowledge of the most arcane details of a system to those who just barely understand the basics of using that system. When you send a message to a wide audience, be considerate of the naive users, who may be confused by technical jargon.

This admonition extends to those who are using a new, restricted program. It does not help a recipient to hear “Oh you're using that old program. Well, I guess you're stuck.” Just don't mention such things to users who cannot take advantage of them.

---

## **The Moral of This Tale**

---

The moral of all this is simple: Be considerate. As we strive toward this goal, everyone's use of the message system will become even more of a joy than it already is.



The main part of this manual tells you how to use Lafite at client companies. However, Lafite may be used somewhat differently at the Xerox site where you work. For example, you may use a message transport system known as Grapevine instead of the NS system; or you may use both systems. Or your site may use both Lafite and Laurel (a mail system similar to Lafite that was written for the Xerox Alto). This appendix gives you the specialized information you may need to use Lafite within Xerox.

Lafite use at Grapevine sites is described first. This section includes the documentation for Maintain, the program used to maintain Grapevine public distribution lists. It also tells you how to send Lafite messages to ARPANET recipients. Next described is Lafite use at sites having both the Grapevine and NS systems. Then there is a short description of some Lafite-related packages currently available only within Xerox. The final section describes the differences between Laurel and Lafite.

Note: this appendix was not indexed due to the difficulty of creating and maintaining separate customer and internal indexes.

---

## The Grapevine Mail System

---

For about five years most Xerox sites have used the Grapevine electronic message transport system; it was for this system that Lafite was originally developed. At the present writing, Xerox is in the process of converting from Grapevine to the NS system. Some sites use only NS mail, some Grapevine, and some both. For the most part, you use Lafite with the Grapevine system in exactly the same way as with the NS system. The four major differences are the mail protocols, the forms of recipient addresses, the ability to send mail to ARPANET recipients, and the way in which public distribution lists are maintained.

---

## Grapevine Protocols

---

The Grapevine system uses its own set of mail protocols, which you can obtain by loading the Library package file MAILCLIENT.DCOM. Simply type (LOAD '{FILESERVER}<DIRECTORY>MAILCLIENT.DCOM in the executive window.

---

## Grapevine Recipient Addresses

---

A Grapevine recipient address has two parts separated by a period, in the form *Recipient.Registry*. The first part is the identifier for the recipient, usually that person's last name. If there are two or more people with the same last name in the same registry, each is usually identified by his or her first initial plus the last name.

The second part of a Grapevine address is the registry name. A registry is no more than a device for grouping related names. The registry name helps the mail system determine which mail server has the in-box for the recipient. Most registry names correspond roughly to "campuses" of activity within Xerox, which should make them easier to remember.

Lafite allows you to omit the registry name for recipients who are in your registry. You can find out what your local registry is by typing DEFAULTREGISTRY at the prompt in your executive window. DEFAULTREGISTRY is the global variable that names your local Grapevine registry; it is used if your log-in name does not include a registry. This variable is normally set in your initialization file.

---

## Grapevine Registries

---

At present, the following registries exist:

|              |                                                                                                                                                          |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| ARPA         | A pseudoregistry used to forward mail to people on the ARPANET and points beyond.                                                                        |
| AUTO         | A registry for software-driven functions requiring automatic access and Grapevine authentication.                                                        |
| CHOLLA       | A registry for PARC/ICL, Palo Alto, California (no individuals in this registry).                                                                        |
| DC           | A registry for the District of Columbia.                                                                                                                 |
| DLOS         | A registry for Dallas, Texas.                                                                                                                            |
| EOSA and EOS | A registry for associates or affiliates.                                                                                                                 |
| ES           | A registry for El Segundo, California.                                                                                                                   |
| FX           | A registry for Fuji Xerox, Japan.                                                                                                                        |
| GUEST        | A registry for Palo Alto, California.                                                                                                                    |
| HENR         | A registry for Henrietta, New York.                                                                                                                      |
| LB           | A registry for Leesburg, Virginia.                                                                                                                       |
| OGC          | A registry for the Office of the General Counsel.                                                                                                        |
| OSDA and OSD | A registry for associates or affiliates.                                                                                                                 |
| PA           | A registry for Palo Alto, California.                                                                                                                    |
| PASA         | A registry for Pasadena, California.                                                                                                                     |
| RX           | A registry for Rank Xerox, Great Britain.                                                                                                                |
| SIEMENS      | A registry for the Communications and Information Systems Group, Private and Special-Purpose Networks Division at Xerox XSG, OSD, Palo Alto, California. |
| STHQ         | A registry for the Xerox Corporate Headquarters, Stamford, Connecticut.                                                                                  |
| SV           | A registry for Sunnyvale, California.                                                                                                                    |
| WBST         | A registry for Webster, New York.                                                                                                                        |
| X            | A registry to be used by people in many geographic locations.                                                                                            |
| XRCC         | A registry for the Xerox Research Centre of Canada, Toronto, Canada.                                                                                     |

---

## Grapevine Distribution Lists

---

The Grapevine system supports over 1,500 distribution lists, some of which are work related and some of which are purely for fun. You can obtain a complete distribution list directory, giving the name of each list plus a short description of its purpose, by printing {Indigo}<Registrar>GV>DLMap.Txt.

Grapevine distribution list recipient names end with the character “^,” in the form AISBU^.PA.

The public distribution lists for each registry are stored on mail servers and are maintained by the administrators of that registry, by the owners of the lists, and in some cases by the members of the lists. To create a public distribution list you must contact an administrator for your registry, who will make sure that your proposed name does not conflict with any others, and who will create the list for you. You can get your name added to appropriate lists by contacting the owners of those lists or by using the Maintain program (see below). If you cannot make a desired change to a public list yourself, send a message to the owners of the list. In Grapevine registries, by convention, the owners of a list named *List^.Registry* are listed in the companion list *Owners-List^.Registry*.

**Delivery to Distribution Lists.** If a recipient list includes public distribution lists, you must confirm the delivery by including a Reply-To field in the message (see chapter 9). If you do not include a Reply-To field before selecting Deliver, Lafite will give you a special Reply-To Field menu, asking you to choose between four options. The options are to send the message as is, have recipients reply to yourself, have them reply to someone else, or to abort the message. This is intended to minimize unintentional deliveries to large distribution lists.

---

## ARPANET Messages

---

The ARPANET (Advanced Research Projects Agency Network) is a network linking ARPA research sites, including some Xerox sites. The Grapevine system is connected to the ARPANET, so you can correspond with individuals at ARPA sites outside Xerox.

As of this writing, ARPANET individual and distribution list addresses are in the form *Name@Host.ARPANET*. However, all ARPANET addresses are gradually being changed to domain names. The primary Xerox address, “Xerox.ARPANET,” will soon become “Xerox.COM”; external ARPANET addresses are also being changed. The name changes are not expected to greatly inconvenience Xerox ARPANET users. The current Xerox address will be supported for some time after the change, so you can receive mail from people who don’t yet have your new address. If you send mail to an obsolete address, the ARPANET mail gateway will insert the proper address in the message header, but it is a good idea to keep an eye out for address changes in the correspondence you receive and update your private list of addresses accordingly. You don’t have to update ARPANET addresses in any Grapevine distribution lists you own, because ARPANET names on Grapevine distribution lists are kept current by Xerox network administrators (who periodically run a program to update all the names). If you want to verify an ARPANET address, you can search for it in the ARPANET host table stored on {Indigo}<NetInfo>ARPANET>Hosts.Txt. The first name in a host entry list is the primary name and the one you should be using.

ARPANET messages sent via Grapevine undergo some special processing. Because many mail systems receiving the mail you send over the ARPANET cannot process the long lines contained in Lafite messages, carriage returns are automatically inserted to make each line less than 72 characters long if there are any lines of

more than 90 characters within the first 5,000 characters of the body of your message. If you have already inserted carriage returns to make your lines slightly longer than 90 characters and you don't want your message to look unaesthetic, include a "Line-Fold: No" line in your header.

**ARPANET Distribution Lists.** ARPANET distribution lists are not maintained at any one, central location. Instead, they are administered locally by interested sites; for example, SF-Lovers@Rutgers is maintained at Rutgers. When you send a message to an ARPANET distribution list address, it goes to the appropriate host, who sends it to all the members of the list. Some hosts anthologize messages for certain lists (such as SF-Lovers); that is, someone at the host site periodically collects messages from members and sends anthologies of them out to everyone on the list.

Almost every ARPANET distribution list has a Xerox-internal redistribution list, so that all Xerox members are actually members of the redistribution list. For example, the redistribution list for SF-Lovers@Rutgers is XeroxSFlovers^.PA. You use Maintain to add yourself to the redistribution list of an ARPANET list in the same way you would add yourself to a Xerox distribution list. If you want to add yourself to (or make another administration request regarding) an ARPANET list that does not have a Xerox redistribution list, send a message to *DistributionList-Request@Host*.

### Other Forms of Grapevine and ARPANET Addresses

---

Lafite recognizes three special forms of address that contain arbitrary text in addition to a valid Grapevine or ARPANET address. The first form is *Human Sensible Name* <Actual Address>. For example, if you wanted to include Cheryl Jones's first name in her address you could write it as Cheryl Jones <Jones.PA>. The area outside the angle brackets can contain any text except for certain prohibited characters: parentheses, angle brackets, commas, colons, semicolons, and quotation marks.

The second form of address is *Actual Address (Comments)*. You could, for instance, send an invitation to Jones.PA (and spouse). You can include any characters within the parentheses except additional parentheses.

The third form of address is "*Comments*" <Actual Address>. For example, you could send a message to "Special Attn: Sybalsky, Shih" TEditFriends^.PA. Any of the prohibited characters can be included within the quote marks except additional quote marks.

### Maintain

---

Maintain is a Lisp Library package used only within Xerox for maintaining Grapevine public distribution lists and changing your Grapevine password. To load Maintain, type (LOAD '{FILESERVER}<DIRECTORY>MAINTAIN.DCOM) in your executive window. To enter Maintain, type (MAINTAIN). Maintain will try to log you in automatically with your current user name and password. If that fails, then you may log in using the Login command in Maintain (see below).

### How to Issue Maintain Commands

---

Maintain has its own prompt sign, in the form GV:. To issue a command in Maintain, you must type after the prompt sign only the initial letters that uniquely identify that command; i.e., as soon as enough of a word (usually one letter) has been typed, Maintain will complete that word for you. For instance, when typing the Add

Friend command, you need only type A F. Throughout this section, the letters that Maintain fills in are given in square brackets.

Once you have issued a command for a specific distribution list or individual, Maintain will fill in that same name for the next command you type. If the name is appropriate for that command, press the space bar or carriage return to confirm it. If the name is not appropriate for the command, begin typing the correct distribution list or individual's name to replace it, ending by hitting the space bar or carriage return.

---

## The Login Command

To log into Maintain, type L[ogin] after Maintain's GV prompt. Now type your name and registry and press the carriage return. Maintain will prompt you for your password, each character of which will appear as an asterisk on the screen. Grapevine maintains its own copy of your password, and the password you give here must be the one Grapevine knows. If you give your password successfully, Maintain will inform you that you are logged in. Once you are logged in, you may proceed to issue other Maintain commands. If you cannot log in, because you have forgotten your password or for some other reason, then you will have to ask an administrator for your registry to issue a Change Password command.

Maintain automatically logs in the current user when it starts up, so you only need to issue the Login command if you wish to change your identity for some reason.

---

## The ? Command

To obtain a list of Maintain commands, type a question mark after the GV prompt. You will see that the commands are Add Friend, Add Member, Add Owner, Change Password, Change Remark, Login, Quit, Remove Friend, Remove Member, Remove Owner, Type Entry, Type Members, and ^YzEnter Lisp. Each of these commands is described below.

---

## The Type Entry Command

The Grapevine system has a *name data base* of entries giving information about groups and individuals. Groups represent distribution lists and some other groups that are unimportant to casual users, such as access control lists. Individuals represent human users and server computers. There are also "abstract users" that are syntactically individuals but actually name a group, such as LafiteSupport.PA.

To read an entry in the data base, type T[type] E[ntry] after the GV prompt. Maintain will ask you for the name of the group or individual whose entry you want to read, then print the entry. When it is finished, it will return you to the GV prompt.

An entry for a group has four components: the *remark*, the number of *members*, a list of *owners*, and a list of *friends*. The remark describes the purpose of the group; for example, the remark for the distribution list Books^.PA is "Resource for Readers (primarily reviews)."

The members are the individuals or other groups contained in the group. To obtain a list of members, use the Type Members command (described below).

The owners are the people empowered to add and remove owners, friends, and members for the group. They can also set and change the remark.

The friends are the people empowered to add and remove their own names to and from the group. If the list of friends is None, no one but the owners can add members to the list. If the list is an asterisk, anyone on the Grapevine system can add himself or herself to the list. If the list is of the form \*. *Registry*, anyone in that registry can add himself or herself to the list, and so on.

An entry for an individual has two components: connect site and mailbox site(s).

---

### The Type Members Command

---

To find out who the members of a distribution list are, type T[type] M[embers] after the GV prompt. Maintain will fill in the words "of group" followed by a colon. Type the name of the distribution list after the colon and press the carriage return. Maintain will then print the list of members. If it is a very long list, your executive window may fill up, stop printing the list, and become highlighted in reverse video, a sign for you to hit the space bar to get the listing going again.

Some members of a group may be groups themselves. To see the members of those groups, you can use extra Type Members commands. If you use Type Members for the special group Groups.*Registry*, you will obtain a list of all distribution lists for that registry.

When Maintain is through printing the members of a distribution list, it will repeat the GV prompt.

---

### The Add Member Command

---

You can add yourself (or another person if you are an owner) to a distribution list by typing A[dd] M[ember] [name] *YourName.Registry* after the GV prompt. Maintain will print "to group" after which you should fill in the name of the distribution list to which you want to add your name. If Maintain prints "done," you have successfully added yourself to the list. If it prints "NoChange," you were already a member of the list. If it prints "NotAllowed," you should send a message to the owners of the list, asking them to add your name.

---

### The Remove Member Command

---

You can remove yourself (or another person if you are an owner) from a distribution list by using the Remove Member command. This works in the same way as the Add Member command.

---

### The Add Owner Command

---

Use the Add Owner command to add an owner to a distribution list. This is not allowed unless you yourself are an owner. Otherwise, this works in the same way as the Add Member command.

---

### The Remove Owner Command

---

Use the Remove Owner command to remove an owner from a distribution list. This is not allowed unless you yourself are an owner. Otherwise, this works in the same way as the Remove Member command.

---

### The Add Friend Command

---

Use the Add Friend command to add a friend to a distribution list. You cannot do this unless you are an owner. Otherwise, this works in the same way as the Add Member command.

### **The Remove Friend Command**

---

Use the Remove Friend command to remove a friend from a distribution list. You cannot do this unless you are an owner. Otherwise, this works in the same way as the Remove Member command.

### **The Change Remark Command**

---

The remark is the description of the distribution list that Maintain prints when you use the Type Entry command. To change the remark, type C[hange] R[emark] after the GV prompt. Maintain will prompt you for the name of the group, then ask you to type the new remark followed by a carriage return. You cannot change the remark for a list unless you are an owner.

### **The Change Password Command**

---

You can change your Grapevine password using Maintain's Change Password command. Type C[hange] P[assword] after the GV prompt. Maintain will respond with "for individual." Now type in your name and registry. Maintain will prompt with "to be." Type in your new password, which will appear as a series of asterisks. Maintain will ask you to confirm your new password by typing it again. When you terminate this word with a carriage return or a space, Maintain will store it in the Grapevine data base. Be very careful when typing your new password, because if you make a mistake and can't reproduce the password later you will have to obtain the help of an administrator for your registry to correct it.

After returning to the executive, you will need to reinvoke the Login command (see above) to continue using Lafite to get and send mail.

### **The ^Y Command**

---

If you want to drop into Lisp temporarily, then return to Maintain, type control-Y at the GV prompt. Maintain will fill in "Enter Lisp" and return you to the regular executive window prompt. To go back to Maintain, type OK.

### **The Quit Command**

---

To leave Maintain, type Q[uit] after the GV prompt. Maintain will ask you to confirm this command with a carriage return.

---

## **If You're on Both Networks**

---

Because the Grapevine and NS systems are totally separate, you must have a registered name and password to send mail on each system you are using. You can have two different passwords, but it is advisable to use the same one because otherwise you will be prompted to log in again each time you switch systems. You can also call (LOGIN 'GV::) yourself to set your Grapevine log-in, and (LOGIN 'NS::) to set an NS log-in, apart from your default (Grapevine) log-in.

### **Lafite Modes**

---

When you use Lafite on the Grapevine system, Lafite is in Grapevine mode. When you use Lafite on the NS system, it is in NS mode. Mail sent to you on a particular system can be

accessed, answered, and forwarded only if Lafite is in that system's mode.

If you have loaded both NS and Grapevine protocol handlers (NS Mail and MailClient), Lafite needs to know which mode to operate in. The first time you start Lafite after loading a new Lafite or a full sysout, you must set the mode explicitly. Until you do this, the Lafite status window reads "Mode Not Set." After that, Lafite starts up in the same mode it was in when you quit Lafite. To change modes after you have initialized Lafite, use the middle mouse button to choose Quit from the status menu. Another menu containing several options will appear; one is GV Mode and another is NS Mode. Select the mode you want with the left mouse button. The center region of your status window reflects the mode you select. If you are in Grapevine mode, it says Lafite (GV). If you are in NS mode, it says Lafite (NS). You can also find out what mode you are in by typing (LAFITEMODE) in your executive window, or change modes by typing (LAFITEMODE 'GV) or (LAFITEMODE 'NS). If you always want to start Lafite in the same mode, you can set the global variable LAFITEMODEDEFAULT to the value GV or NS, in which case Lafite will initialize itself in that mode.

At sites with multiple Lafite modes, the values of the global variables LISPSUPPORT, LAFITESUPPORT, and TEDITSUPPORT are association lists in the form ((Mode1 "Address1") (Mode2 "Address2")---). Because the Lisp Report, Lafite Report, and TEdit Report forms are currently all sent to Grapevine addresses, they cannot be used at all in NS mode.

---

## From Grapevine to NS and Vice Versa

---

Without switching modes, you can send mail from a Grapevine to an NS address or vice versa through a *mail gateway*. The gateway provides pseudoregistries and pseudodomains for mail addressed to the other kind of network. (A list of the special registries and the domains they correspond to is stored on {Indigo}<Registrar>GV>GV-NS-Mapping.Txt.) A single message can contain both Grapevine and NS addresses.

When Lafite is in Grapevine mode, you can send mail to the NS address *Name:Domain:Organization* by addressing it to "*Name:Domain:Organization*".NS (the double quotes are needed to keep the colon and any spaces in the name from being misparsed). For convenience, the domains PARC, OSBU North, and OSBU South can also be addressed as the registry of the same name, with the spaces removed. Thus, to send mail to Jones:PARC:Xerox, send it to Jones.PARC; to send mail to Oscar:OSBU North:Xerox, send it to Oscar.OSBUNorth.

When Lafite is in NS mode you can send mail to the GV address *Name.Registry* by addressing it to *Name.Registry:GV*. For convenience, the registries PA and ES can be addressed directly as NS domains. Thus, to send mail to Jones.PA, send it to Jones:PA.

**Conversion of Grapevine Distribution Lists for NS Users.** Until recently, people who used only NS mail could not add themselves to the many Grapevine public distribution lists. To address this problem, a new naming structure is being applied to general-interest public distribution lists in the PA, ES, and X registries. (Most organization and project lists, or any lists that might be used for access control, will not be converted to the new naming structure.) The new structure allows both NS and Grapevine mail users to add themselves to, broadcast to, and receive messages from these lists. Grapevine users will have to change the way they add themselves to lists. NS users will have to change both the way they send messages to lists that currently reside in Grapevine and the way they add themselves to lists.



To enable the new distribution list structure, new NS domains have been established in a one-to-one correspondence with the ES, PA, and X Grapevine registries. "All Areas:Xerox" is the NS domain analogous to the X registry. "PA Area:Xerox" is the NS analog of the geographically based registry PA, and "ES Area:Xerox" is analogous to ES. After the restructuring is implemented for a distribution list in, for example, the X registry, Grapevine users will continue to broadcast to *List^X*, but will add themselves to *List-GV^X*. NS users will broadcast to *List:All Areas*, but will add themselves to *List-NS:All Areas*. Either form of broadcast will reach both the Grapevine members (*List-GV^X*) and the NS members (*List-NS:All Areas:Xerox*).

The conversion process will take several months. Distribution lists will be converted upon requests from their owners. If you own a distribution list and wish to request conversion, retrieve {USC:OSBU South:Xerox}<Public USC>DLRequest.Form, fill out the form, and send it to the appropriate address, according to the table below.

| Registry | Grapevine User              | NS User                           |
|----------|-----------------------------|-----------------------------------|
| ES lists | UserAdministration.ESArea   | UserAdministration:ES Area:Xerox  |
| PA lists | UserAdministration.PAArea   | UserAdministration:PA Area:Xerox  |
| X lists  | UserAdministration.AllAreas | UserAdministration:AllAreas:Xerox |

As soon as a distribution list of which you are a member is converted to the new naming structure, you will receive notice of the conversion. If you are a Grapevine member of *List^X*, you will be added to *List-GV^X* and removed from *List^X*. If you are a Grapevine user and want to add yourself to a distribution list of which you are not currently a member, use Maintain to attempt to add yourself to *List-GV^X*. If Maintain complains that this list does not exist, then add yourself to *List^X*. You should continue to address messages to *List^X*.

If you are primarily an NS user and are a member of Grapevine *List^X*, then when that distribution list is converted to the new structure, you will be moved to *List-GV^X*. Then you should ask your system administrator to add you to *List-NS:All Areas:Xerox*. To make sure you don't miss any mail, wait three days after you start receiving two copies of messages sent to *List^X*, then remove yourself from *List-GV^X* with Maintain. To add yourself to a new distribution list, ask your system administrator to add you to *List-NS:All Areas:Xerox*. If this list does not exist, then add yourself to *List^X* using Maintain. You should broadcast messages to *List:All Areas:Xerox*.

One key point to remember is to never send messages to a list with a -GV or -NS suffix. If you do, you will not reach both NS and Grapevine members. Another key point is, never broadcast to both *List^X* and *List:AllAreas*, or everybody will get two copies of the message.

A more detailed description of the conversion process is filed on {USC:OSBU South:Xerox}<Public USC>DLConversion.Doc and {Indigo}<Registrar>GV>DLConversion.Doc.

## Lafite-Related Lisp Users' Packages

As of this writing, Xerox employees have access to four Lafite-related Lisp Users' packages that have not yet been released to customers. The packages are Mailomat, Mail Sorter, Mail Cards, and Mail Reader. Mailomat enables you to automatically retrieve mail for one or multiple users for whom you have passwords. The

program is filed on {Eris}<Lisp>Koto>LispUsers>Mailomat.DCOM; the documentation is on {Eris}<Lisp>Koto>LispUsers>Mailomat.TEdit. Mail Sorter will automatically sort mail into folders according to a set of rules you have specified. The code and documentation are on {QV}<Pimp>MailSorter.DCOM and {QV}<Pimp>MailSorter.TEdit, respectively. Mail Cards integrates the Note Cards environment with the Lafite mail system; you can place Lafite messages in specialized note cards, which are then placed in a Note Cards network. The code is stored on {QV}<Pimp>MailCards.DCOM and {QV}<Pimp>MailCardsLafiteInterface.DCOM; the documentation, on {QV}<Pimp>MailCards.TEdit. Lastly, the Mail Reader uses a voice synthesizer to read your mail to you over the telephone. You do not need special code to use this package, because you enter all commands by pressing the buttons on a touch-tone telephone. The Mail Reader's documentation is stored on {QV}<Speech>Speech>MailReader>.MailReader.TEdit.

All the abovementioned file names are subject to change. If you can't find a specific package, ask a release master or a network administrator for a pointer.

---

## From Laurel to Lafite

---

Previous users of the Laurel mail program will find Lafite to have a similar user interface. Some of the ways in which it differs from Laurel are the following:

Laurel can only access mail folders on the local disk; Lafite can access folders on remote file servers. Thus, it is not necessary to transfer mail folders back and forth between your local disk and your file servers.

Laurel can only "browse" one mail folder at a time; Lafite can have several mail folders "opened" at the same time. Using the Interlisp window system, you can view the tables of contents of many mail folders and refer to messages in them independently.

You may have multiple windows displaying messages and multiple windows for sending messages and you may move text freely among them.

Messages can be selected in the table of contents by clicking *anywhere* within the summary line, unlike in Laurel, where you must select at the left end.

---

## The Browse Laurel File Command

---

Lafite can read mail files written by the Laurel and Hardy mail programs; the files it writes are in Laurel format.

To browse a Laurel file that was produced by the Laurel mail reader version 6.1 or later, use the Browse Laurel File command from the middle-button Browse menu. Laurel 6.1 files are almost the same as Lafite files, but contain some line-formatting information that is stripped out by this command. After you have applied this command once to a file, you can subsequently browse the file with the normal Browse command (unless you use Laurel on it again, of course). This command is not intended for repeated use, but simply to make mail files built by Laurel more pleasing to browse. This command has the side effect of destroying any TEdit-formatted messages, so it should be used with care.

# GLOSSARY OF TECHNICAL TERMS

---

This glossary covers hardware and software terms not specific to Lafite that are used in this manual.

**argument** Element of a Lisp function that specifies what that function operates on. For example, when you call (LOAD 'FILENAME), *FILENAME* specifies what file is to be loaded. Note the use of parentheses and the single quote mark—the quote causes *FILENAME* to be taken literally, rather than as the name of a variable to be evaluated.

**association list** A list of lists that associates a key (usually an atom) with a value. The first element of each sublist is the key; the rest of the sublist is a value.

**atom** Any continuous string of characters, numbers, letters, or combinations thereof (except some prohibited characters such as parentheses). An atom is the smallest structure in Lisp.

**back up** To duplicate files to use in case the originals are destroyed in a hardware or software failure.

**bit map** A representation of any graphical entity as a sequence of bits.

**break** A state entered by Lisp during error processing that allows you to recover from the error by typing commands in a break window. If you don't know what to do with a break, type ^ after the prompt to abort the operation, then start over.

**buffer** A temporary storage area in a computer.

**bug** Any error, malfunction, or problem with hardware or software that produces unexpected or unintended results.

**cache** To save or store intermediate results to avoid having to recompute them.

**caret** A blinking pointer indicating where keyboard characters will appear when typed.

**case sensitive** Sensitive to case, that is, upper- and lowercase letters have different meanings.

**Clearinghouse** A Xerox Network service that provides a directory function within an internetwork, allowing all other system components to locate requested resources and other registered objects. It is implemented as a distributed system.

**cursor** A small picture (usually an arrow) on the display that tracks the motion of the mouse.

**data base** A collection of data organized for rapid search and retrieval.

**DEdit** The Interlisp-D editor for programming code.

**default** An action taken (or value specified) unless another action is specified by the user.

**directory** A set of one or more files that are stored together in the same place on a device.

**executive window** The window the blinking caret is in when you first start up Lisp on the machine. Note that if this and any other window overlap, the executive window dominates (i.e., covers) the other while control is in the executive window. To transfer control to another window (that accepts type-in), click the mouse in it, which usually also brings the window to the top.

**extension** The second part of a file name, usually used to indicate the type of file. It is separated from the first part of the file name (which indicates what the file contains) by a period. For example, the extension for Lafite mail folders is Mail, so a typical file name might be Active.Mail.

**field** A part of a message header, preceded by a field name and a colon. Contains information identifying the sender, recipients, subject, or other information of interest either to the users or to the mail system.

**file server** A computer on the network that provides a file storage and retrieval service.

**format** To produce in a specified form; the layout of a document, etc.

**function** A Lisp procedure that carries out a series of steps to produce some result. A function has a name and zero or more arguments on which it does its work.

**global variable** A variable accessible globally to all loaded programs, and whose value is usually not changed, except explicitly by the user (e.g., in an init file). Global variables are often used to personalize some aspect of the behavior of a program.

**hard copy** The physical copy of an on-screen document.

**host** Any machine on a network. Often used to refer specifically to a machine that provides a network service, such as filing.

**icon** A pictorial representation, usually of a shrunken window. An icon can be moved about on the screen by clicking on it with the left mouse button, and can be expanded by clicking on it with the middle mouse button.

**intialization (init) file** A file that is loaded when an Interlisp sysout is first started, and which usually customizes your Lisp environment according to your tastes and the idiosyncracies of your site. The usual arrangement is to have a site initialization file, which supplies information common to all users at your site (e.g., the name of your printer or directory search paths), and a personal initialization file, which supplies information about how you personally like your environment set up (e.g., where on the screen you like your Lafite windows).

**keyword** A significant word from a title or document used as an index to content.

**mail server** A computer that stores and distributes mail.

**menu** A collection of text strings, buttons, or icons on a display screen generally used to present a set of possible actions.

**microcode** The microinstructions of a computer, especially a microprocessor.

**mode** A particular functioning arrangement or condition of a computer.

**network** An interconnection, by some communications medium, of several computers, allowing them to communicate and share resources. Sometimes also used to refer to the medium itself, e.g., a coaxial cable in the case of the Ethernet.

**package** An Interlisp-D tool, or program, designed to help you carry out some complex procedure or task. Lafite is a package that enables you to read and send mail. A package must be loaded into your environment before you can use it.

**parse** To analyze something into its constituent parts, either to expose the structure or extract information. Specifically used to refer to the action of locating the message boundaries in a mail folder, or extracting fields from a message header (e.g., to discover who the message is addressed to).

**position** As a Lisp data type, a pair of display coordinates, in the form (XPosition .YPosition). If you want to know the coordinates of a particular spot on the screen, type (GETPOSITION) and then click the mouse over the desired spot.

**protocols** Standards specifying how machines exchange information over a network.

**random access** Access to arbitrary parts of a file in no particular order (contrast with sequential access, which is accessing a file from the start sequentially through to the end).

**region** As a Lisp data type, a set of numbers describing a rectangular region of the display, in the form (*LEFTCOORDINATE BOTTOMCOORDINATE WIDTH HEIGHT*). If you want to know the region corresponding to any particular rectangular area on the screen, type (GETREGION) and then shape the rectangle as desired.

**scavenge** To try to fix messages that are not in the proper format.

**search path** List of directories to be searched for a file in the order given.

**stream** A programming abstraction that links your display to your message file.

**string** A sequence of characters. As a type of Lisp data, always entered in double quotes.

**sysout** A frozen version of an Interlisp-D environment. It contains all the information needed to initialize virtual memory when Interlisp-D is started.

**time out** To stop waiting for a user response, or to sever a network connection because of inactivity on the part of one of its participants.

**value** A word or a number that a variable is set to.

**virtual memory** Large working space on a local disk within which the machine runs Lisp.

# GLOSSARY OF TECHNICAL TERMS

---

This glossary covers hardware and software terms not specific to Lafite that are used in this manual.

**argument** Element of a Lisp function that specifies what that function operates on. For example, when you call (LOAD 'FILENAME), *FILENAME* specifies what file is to be loaded. Note the use of parentheses and the single quote mark—the quote causes *FILENAME* to be taken literally, rather than as the name of a variable to be evaluated.

**association list** A list of lists that associates a key (usually an atom) with a value. The first element of each sublist is the key; the rest of the sublist is a value.

**atom** Any continuous string of characters, numbers, letters, or combinations thereof (except some prohibited characters such as parentheses). An atom is the smallest structure in Lisp.

**back up** To duplicate files to use in case the originals are destroyed in a hardware or software failure.

**bit map** A representation of any graphical entity as a sequence of bits.

**break** A state entered by Lisp during error processing that allows you to recover from the error by typing commands in a break window. If you don't know what to do with a break, type ^ after the prompt to abort the operation, then start over.

**buffer** A temporary storage area in a computer.

**bug** Any error, malfunction, or problem with hardware or software that produces unexpected or unintended results.

**cache** To save or store intermediate results to avoid having to recompute them.

**caret** A blinking pointer indicating where keyboard characters will appear when typed.

**case sensitive** Sensitive to case, that is, upper- and lowercase letters have different meanings.

**Clearinghouse** A Xerox Network service that provides a directory function within an internetwork, allowing all other system components to locate requested resources and other registered objects. It is implemented as a distributed system.

**cursor** A small picture (usually an arrow) on the display that tracks the motion of the mouse.

**data base** A collection of data organized for rapid search and retrieval.

**DEdit** The Interlisp-D editor for programming code.

**default** An action taken (or value specified) unless another action is specified by the user.

**directory** A set of one or more files that are stored together in the same place on a device.

**executive window** The window the blinking caret is in when you first start up Lisp on the machine. Note that if this and any other window overlap, the executive window dominates (i.e., covers) the other while control is in the executive window. To transfer control to another window (that accepts type-in), click the mouse in it, which usually also brings the window to the top.

**extension** The second part of a file name, usually used to indicate the type of file. It is separated from the first part of the file name (which indicates what the file contains) by a period. For example, the extension for Lafite mail folders is Mail, so a typical file name might be Active.Mail.

**field** A part of a message header, preceded by a field name and a colon. Contains information identifying the sender, recipients, subject, or other information of interest either to the users or to the mail system.

**file server** A computer on the network that provides a file storage and retrieval service.

**format** To produce in a specified form; the layout of a document, etc.

**function** A Lisp procedure that carries out a series of steps to produce some result. A function has a name and zero or more arguments on which it does its work.

**global variable** A variable accessible globally to all loaded programs, and whose value is usually not changed, except explicitly by the user (e.g., in an init file). Global variables are often used to personalize some aspect of the behavior of a program.

**hard copy** The physical copy of an on-screen document.

**host** Any machine on a network. Often used to refer specifically to a machine that provides a network service, such as filing.

**icon** A pictorial representation, usually of a shrunken window. An icon can be moved about on the screen by clicking on it with the left mouse button, and can be expanded by clicking on it with the middle mouse button.

**intialization (init) file** A file that is loaded when an Interlisp sysout is first started, and which usually customizes your Lisp environment according to your tastes and the idiosyncracies of your site. The usual arrangement is to have a site initialization file, which supplies information common to all users at your site (e.g., the name of your printer or directory search paths), and a personal initialization file, which supplies information about how you personally like your environment set up (e.g., where on the screen you like your Lafite windows).

**keyword** A significant word from a title or document used as an index to content.

**mail server** A computer that stores and distributes mail.

**menu** A collection of text strings, buttons, or icons on a display screen generally used to present a set of possible actions.

**microcode** The microinstructions of a computer, especially a microprocessor.

**mode** A particular functioning arrangement or condition of a computer.

**network** An interconnection, by some communications medium, of several computers, allowing them to communicate and share resources. Sometimes also used to refer to the medium itself, e.g., a coaxial cable in the case of the Ethernet.

**package** An Interlisp-D tool, or program, designed to help you carry out some complex procedure or task. Lafite is a package that enables you to read and send mail. A package must be loaded into your environment before you can use it.

**parse** To analyze something into its constituent parts, either to expose the structure or extract information. Specifically used to refer to the action of locating the message boundaries in a mail folder, or extracting fields from a message header (e.g., to discover who the message is addressed to).

**position** As a Lisp data type, a pair of display coordinates, in the form (XPosition .YPosition). If you want to know the coordinates of a particular spot on the screen, type (GETPOSITION) and then click the mouse over the desired spot.

**protocols** Standards specifying how machines exchange information over a network.

**random access** Access to arbitrary parts of a file in no particular order (contrast with sequential access, which is accessing a file from the start sequentially through to the end).

**region** As a Lisp data type, a set of numbers describing a rectangular region of the display, in the form (*LEFTCOORDINATE BOTTOMCOORDINATE WIDTH HEIGHT*). If you want to know the region corresponding to any particular rectangular area on the screen, type (GETREGION) and then shape the rectangle as desired.

**scavenge** To try to fix messages that are not in the proper format.

**search path** List of directories to be searched for a file in the order given.

**stream** A programming abstraction that links your display to your message file.

**string** A sequence of characters. As a type of Lisp data, always entered in double quotes.

**sysout** A frozen version of an Interlisp-D environment. It contains all the information needed to initialize virtual memory when Interlisp-D is started.

**time out** To stop waiting for a user response, or to sever a network connection because of inactivity on the part of one of its participants.

**value** A word or a number that a variable is set to.

**virtual memory** Large working space on a local disk within which the machine runs Lisp.



**A**

Abort command, 45, 47, 87  
 Active Mail folder. *See* folders, Active Mail  
 addresses, recipient, 28–29, 71–73  
 Answer command, 38–39, 86  
 Answer form. *See* forms, Answer  
 authentication, of users, 3, 7, 51

**B**

bit maps, including in messages, 36–37, 46  
 Body command, 64  
 Browse command, 9, 22–23, 42, 85  
 Browse Laurel File command, 23, 85  
 browser window. *See* windows, browser

**C**

CC field, 26, 28–29  
 Close command, 16  
 commands  
     invoking, 8  
     Lafite, 85–87. *See also* individual  
     command names

**D**

DEFAULTMAILFOLDERNAME [variable], 56  
 Delete command, 15, 86  
 Deliver command, 45, 87  
 Display command, 11–12, 86  
 distribution lists, 28–31, 39, 45, 76–77  
 Don't Quit command, 49  
 Don't Update File command, 16, 49

**E**

electronic mail  
     cost structure, 75  
     need for etiquette, 70–71, 73–74, 79  
 Expunge Deleted Messages command, 16, 49, 53, 87

**F**

fields, header, 26–31, 38–42. *See also* individual  
     field names  
 files  
     including in messages, 33–34  
     loading, 5–6  
     mail. *See* folders  
     necessary, 5  
 Find Again command, 63, 65  
 Find command, 63–65  
 Find Next All command, 63–65  
 Find Next One command, 63–65  
 Find Previous All command, 64  
 Find Previous One command, 63–65  
 Find Related Backward command, 65  
 Find Related command, 63–65  
 folder menu, 9, 21–24  
 folders  
     Active Mail, 3, 7, 9, 17, 21, 52, 56  
     creating, 21–22  
     default, 23, 56  
     deleting, 23–24  
     naming, 21–22, 56  
     parsing, 52–53  
     repairing, 66–67  
     storing, 3, 51–52, 55  
     structure of, 3  
     updating, 16–17, 49  
 fonts, 36, 57  
 Forget Folder command, 23–24, 85  
 Forget Message Form command, 42–43, 85  
 forms  
     Answer, 38–39  
     creating, 41–42, 59–61  
     deleting, 42–43

- directory search path for, 55
- extension for names, 56
- Forward, 39
- kinds of, 25–26
- menu, 38, 42–43
- Standard, 26–27, 38, 86
- Report, 40–41, 56–57

## Forward

- command, 39, 86
- form. *See* forms, Forward

Frequency field, 40

## From

- command, 64–65
- field, 29–30

# G

Get command, 34–35

Get Mail command, 10, 87

# H

Hardcopy command

- Lafite, 19, 86
- window menu, 19, 36

headers. *See* message headers

# I

Impact field, 41

in-box, 3, 7, 10, 51, 58

Include command, 33–34

In-Reply-To field, 31

# J

Just Close command, 17

# L

Lafite

- capabilities of, 1, 3
- quitting, 49
- starting, 7

LAFITEBROWSERFONT [variable], 57

LAFITEBROWSERREGION [variable], 56

LAFITEBUFFERSIZE [variable], 59

LAFITEDEFAULTHOST&DIR [variable], 55

LAFITEDISPLAYAFTERDELETEFLG [variable], 59

LAFITEDISPLAYFONT [variable], 57

LAFITEDISPLAYREGION [variable], 56

LAFITEEDITORFONT [variable], 57

LAFITEEDITORREGION [variable], 56

LAFITEENDOFMESSAGEFONT [variable], 57

LAFITEENDOFMESSAGESTR [variable], 58

Lafite Find, 5–6, 63–65

LAFITEFLUSHMAILFLG [variable], 58

LAFITEFORMDIRECTORIES [variable], 55

LAFITEFORM.EXT [variable], 56

LAFITEFORMSMENU [variable], 60

LAFITEGETMAILTUNE [variable], 59

LAFITEHARDCOPYBATCHFLG [variable], 58

LAFITEHARDCOPYFONT [variable], 57

LAFITEHARDCOPY.MIN.TOC [variable], 58

LAFITEIFFFROMMETHENSEENFLG [variable], 58

LAFITEMAIL.EXT [variable], 56

LAFITEMENUFONT [variable], 57

LAFITEMOVETOCONFIRMFLG [variable], 59

LAFITENEWMAILTUNE [variable], 59

LAFITENEWPAGEFLG [variable], 58

(LAFITE 'OFF) [function], 49

(LAFITE 'ON 'MAILFOLDER ) [function], 7

LAFITEOUTBOXSIZE [variable], 59

Lafite Report

- command, 86

- form. *See* forms, Report

LAFITESPECIALFORMS [variable], 60

LAFITESTATUSWINDOWPOSITION [variable], 56

LAFITESUPPORT [variable], 57

LAFITETITLEFONT [variable], 57

LAFITETOC.EXT [variable], 56

Lisp Report

- command, 86

- form. *See* forms, Report

LISPSUPPORT [variable], 56

log-in procedures, 5

# M

(MAILSCAVENGE 'FILENAME 'ERRORMSGSTREAM

'TEMPDIR) [function], 66–67

(MAILSCAVENGE.IN.PLACE 'FILENAME) [function], 67

Mail Scavenger, 5–6, 53, 63, 66–67

MAILWATCHWAITTIME [variable], 58

(MAKEXXXSUPPORTFORM 'SYSTEMNAME  
'ADDRESS 'SYSTEMDATE) [function], 60–61

menus, 8, 34, 36, 63. *See also* individual  
command names

message

body, 25–26

forms. *See* forms

headers, 3, 25–31, 38–41, 63–64

marks, 10–11, 19, 21, 39, 58, 85

messages

answering, 38–39

deleting, 15

delivering, 45–47

displaying, 9–14, 59

expunging, 16

filing, 21–24. *See also* folders

forwarding, 39

misaddressed, 71–73

moving, 21–22, 59

printing, 19, 36, 57–58

and privacy, 71–72, 78

retrieving, 10, 35

saving, 34–35, 41–42

searching for, 63–65

selecting, 11–12

structure of, 3

undeleting, 15

writing, 25–43

Move To command, 21–22, 53, 86

## N

NS Mail

service, 3

protocols, 5–6

## O

Other Folder command, 21–24

out-box, 8, 46, 59

## P

protocols. *See* NS Mail

Put command, 34–35

## Q

Quit command, 49, 86

## R

Related command, 64–65

Reply-To field, 30–31

Report forms. *See* forms, Report

Restart command, 49, 86

## S

Saved Form command, 38, 42, 86

Save Form command, 41–42, 87

scroll bars. *See* windows, scrolling

selection pointer. *See* messages, selecting

Sender field, 29–30

Send Formatted Message command, 46

Send Mail command, 38, 42, 85

Send Plain Text command, 46–47

Shrink command, 16

Standard form. *See* forms, Standard

start-up procedures. *See* Lafite, starting

status window. *See* windows, status

Subject

command, 64

field, 26–27

## T

table of contents, Lafite, 3, 9–12, 15–17, 21–22, 24,  
52–53, 56, 58

TEdit, 1, 31–37, 53

TEdit Report

command, 38, 86

form. *See* forms, Report

TEDITSUPPORT [variable], 57, 60

TEDITSYSTEMDATE [variable], 60

text

copying, 33

- deleting, 33
- inserting, 31–32
- moving, 33
- selecting, 32–33

To field, 26, 28–29

## U

Undelete command, 15, 86

Update command, 16–17, 53, 86

Update Table of Contents command, 17

users

- authenticating. See authentication of users
- fictitious, 78–79

## V

variables, 55–60

## W

window menu, 36

windows

- browser, 9–10, 22, 56, 86–87
- closing, 16–17, 35, 46
- message composition, 25–26, 31–33, 35, 56, 87
- message display, 11–14, 56
- scrolling, 12–13
- shrinking, 16–17, 35
- status, 7–8, 46, 49, 56, 85–86
- thumbing, 13–14

Write Out Changes Only command, 16, 49, 53, 87

---

## Message Marks

---

? Means that a message has not yet been displayed.

*a* Means that a message has been answered.

*f* Means that a message has been forwarded.

*h* Means that a message has been hard-copied.

*m* Means that a message has been moved to another folder.

Note: Only one mark is present at a time; each mark replaces the previous one.

---

## Lafite Commands

---

---

### Status Window Commands

---

#### Browse

Pops up a menu of your mail folders. Selecting Browse with the middle button brings up another menu with the additional commands:

##### **Browse Laurel File**

Browses a file that was produced by Laurel.

##### **Forget Folder**

Removes a folder from the list of known mail folders.

##### **Forget Message Form**

Removes a message form from the list of known message forms.

#### Send Mail

Brings up a message composition window. If Send Mail is selected with the middle button, a menu is presented with the following choices:

##### **Lisp Report**

Provides a message template to report an Interlisp bug or make a suggestion.

##### **Lafite Report**

Provides a message template similar to the Lisp Report but sent to Lafite maintainers.

##### **TEdit Report**

Provides a message template similar to the Lisp Report but sent to TEdit maintainers.

**Saved Form**

Prompts for a form name.

**Standard Form**

Provides an empty message template.

**Quit**

Stops Lafite and closes all browser windows. Selecting the Quit command with the middle button brings up a menu of status-changing commands:

**Restart**

Turns Lafite off, then on again.

---

**Browser Window Commands**

---

**Display**

Displays the selected message.

**Delete**

Deletes the selected message.

**Undelete**

Undeletes the selected message.

**Answer**

Constructs an Answer form for the selected message.

**Forward**

Constructs a Forward form for the selected message.

**Hardcopy**

Prints the selected message.

**Move To**

Pops up a menu of known mail folders and moves the selected message to the chosen folder.

**Update**

Transmits the changes that you made to your mail folder to the physical mail file. Selecting Update with the middle mouse button brings up a menu with the commands:

**Write Out Changes Only**

Makes the browser and the mail file completely consistent with each other.

**Expunge Deleted Messages**

Removes all messages marked deleted, in addition to making the browser and the mail file completely consistent with each other.

**Get Mail**

Brings new mail into the folder.

## Message Composition Window Commands

---

### **Deliver**

Sends your message. During delivery, the menu atop the message window changes into a single item, **Abort**; if you click on this item, the delivery is aborted.

### **Save Form**

Asks you for a file name on which to save this message for later use as a message form.

---

## Message Marks

---

? Means that a message has not yet been displayed.

*a* Means that a message has been answered.

*f* Means that a message has been forwarded.

*h* Means that a message has been hard-copied.

*m* Means that a message has been moved to another folder.

Note: Only one mark is present at a time; each mark replaces the previous one.

---

## Lafite Commands

---

---

### Status Window Commands

---

#### Browse

Pops up a menu of your mail folders. Selecting Browse with the middle button brings up another menu with the additional commands:

##### **Browse Laurel File**

Browses a file that was produced by Laurel.

##### **Forget Folder**

Removes a folder from the list of known mail folders.

##### **Forget Message Form**

Removes a message form from the list of known message forms.

#### Send Mail

Brings up a message composition window. If Send Mail is selected with the middle button, a menu is presented with the following choices:

##### **Lisp Report**

Provides a message template to report an Interlisp bug or make a suggestion.

##### **Lafite Report**

Provides a message template similar to the Lisp Report but sent to Lafite maintainers.

##### **TEdit Report**

Provides a message template similar to the Lisp Report but sent to TEdit maintainers.



**Saved Form**

Prompts for a form name.

**Standard Form**

Provides an empty message template.

**Quit**

Stops Lafite and closes all browser windows. Selecting the Quit command with the middle button brings up a menu of status-changing commands:

**Restart**

Turns Lafite off, then on again.

---

**Browser Window Commands**

---

**Display**

Displays the selected message.

**Delete**

Deletes the selected message.

**Undelete**

Undeletes the selected message.

**Answer**

Constructs an Answer form for the selected message.

**Forward**

Constructs a Forward form for the selected message.

**Hardcopy**

Prints the selected message.

**Move To**

Pops up a menu of known mail folders and moves the selected message to the chosen folder.

**Update**

Transmits the changes that you made to your mail folder to the physical mail file. Selecting Update with the middle mouse button brings up a menu with the commands:

**Write Out Changes Only**

Makes the browser and the mail file completely consistent with each other.

**Expunge Deleted Messages**

Removes all messages marked deleted, in addition to making the browser and the mail file completely consistent with each other.

**Get Mail**

Brings new mail into the folder.

## Message Composition Window Commands

---

### **Deliver**

Sends your message. During delivery, the menu atop the message window changes into a single item, Abort; if you click on this item, the delivery is aborted.

### **Save Form**

Asks you for a file name on which to save this message for later use as a message form.

# 1. INTRODUCTION

---

---

## What Lafite Is

---

Lafite is an easy-to-use, display-oriented electronic message system. Like all electronic message systems, Lafite combines the advantages of the postal service and the telephone system. You can compose messages at leisure and in private, transmit them rapidly, and print copies of the messages you send and receive.

However, Lafite offers more than most electronic message systems. It uses the Interlisp-D window interface, which enables you to read and write many messages simultaneously while you are using other Interlisp systems. It is also fully integrated with the TEdit text editor, which allows you to write messages using all of TEdit's editing and formatting capabilities. In addition, Lafite enables you to create mail folders and file your messages in them and to create, save, and use special message forms.

---

## What This Manual Is About

---

This manual is designed to help you use Lafite. It assumes you understand the basic principles of using your Xerox Lisp workstation and the Interlisp-D environment. If you need information on operating your workstation, refer to its user's manual. The *Friendly Dandelion Primer* and the *Interlisp-D Reference Manual* describe the Interlisp environment.

If you are a new Lafite user, you will find it helpful to skim through the manual, then return to the beginning and practice using Lafite's features in the order in which they are described. If you are an experienced Lafite user, you should just skim through the manual, then turn to the appropriate section whenever you need help. If you are a technically knowledgeable or adventurous user, you may want to know how to customize the system. Information on modifying Lafite's behavior is provided in chapter 13, "Customizing Lafite."

You can find some useful background information in chapter 2, "Things to Know Before You Start." Chapter 3, "Obtaining Lafite," tells you how to log in and how to load Lafite and the other software you need to use it. In chapter 4, "Entering Lafite," you can find out how to access Lafite and how to use the highest-level Lafite window, the status window. You can learn how to read your mail in chapter 5, "Reading Messages," and how to update your mail files in chapter 6, "Deleting Messages and Updating the Mail Folder." Chapter 7, "Printing Messages," tells you how to obtain hard copies of your messages, and chapter 8, "Filing Messages," tells you how to organize them. You can find out how to compose and transmit Lafite mail in chapters 9 and 10, "Writing Messages" and "Sending Messages." Chapter 11, "Leaving Lafite," describes several ways you can exit the system.

If Lafite does not seem to be operating correctly, you can turn to chapter 12, "Troubleshooting Lafite Problems." Chapter 13, "Customizing Lafite," tells you how to change Lafite's global variables (we have tried to make it accessible to nonprogrammers who are willing to do a little extra reading). And chapter 14, "Using Lafite-Related Lisp Library Packages," contains user's guides to the Lafite Find and Mail Scavenger Lisp Library packages.

You will find several different kinds of reference material at the end of this manual. Appendix A, "Using Lafite Courteously," is a guide to message system etiquette. You can look up technical terms used in the manual in the glossary at the end of appendix A. After appendix A, you will find a quick-reference section giving brief summaries of Lafite's commands and a chart of message marks. At the very end, there is an index that shows you where to find all the important information given in this manual.

Lafite is a display-based, interactive program that manipulates a particular class of files called *mail files*. In essence, a mail file is just a set of messages, each of which is a piece of text formatted according to certain conventions. The details of these conventions are not of major interest to most users. Suffice it to say that messages have a *header* and a *body*. The header identifies the *sender* and *recipients* of the message as well as the *time* it was sent, a *subject*, and perhaps other information not directly related to the message's content. The body contains the content of the message. For each mail file, Lafite constructs and maintains a *table of contents* that summarizes the messages residing in the file. Mail files can be stored either on a file server that allows random access or on your own workstation's disk.

Lafite is an interface program that gives you access to the facilities provided by the Xerox Network Systems (NS) Mail Service. This system provides you with *in-boxes*, *mail drops*, and a *name data base*. An in-box is a place where messages sent to you are stored until you retrieve them. Each in-box resides on a *mail server*, which is a machine used for storing and distributing electronic mail. A mail drop is a service to which messages addressed to any user(s) of the system may be sent. The mail transport system sorts out the recipients and delivers messages to their individual in-boxes. The name data base holds the names of *registered users* of the message system along with named lists of such users.

Lafite retrieves inbound mail from your in-boxes on one or more mail servers. The messages are retrieved into a *mail folder*, which is usually called Active.Mail (henceforth referred to as the Active Mail folder). You can then open a *browser* onto the mail folder, which allows its contents to be read, replied to, forwarded, deleted, moved into other folders, printed, and so forth.

---

## What Programs You Need

---

Before you run Lafite, two Lisp Library programs must be loaded into your workstation's virtual memory. These are Lafite itself and NS Mail. NS Mail contains an implementation of the *protocols*, or information interchange standards, that enable Lafite to communicate with NS mail servers.

In addition, you may want to load two optional but useful programs, Lafite Find and Mail Scavenger. Lafite Find facilitates searching for messages contained in a mail folder. Mail Scavenger helps to restore the internal structure of a mail file in the unlikely event that it becomes damaged.

---

## How to Load the Programs You Need

---

To load Lafite or another program from a floppy disk, type (LOAD '{FLOPPY}FILENAME.DCOM) at the prompt in your Interlisp-D Executive window. However, if a program is frequently used in your company it is probably stored on a file server rather than on a floppy disk. The following instructions apply to loading packages from a file server, with the words *FILESERVER* and *DIRECTORY* standing for the names of the actual file server and directory on which these programs are stored.

First, you must log into the system. To log in, type (LOGIN) at the prompt in the executive window (see figure 1). On the next line, the system will print the word Login, followed by a colon. Type your log-in name after the colon and press the carriage return. On the same line, the system will prompt you for your password. Type in your password (each letter will appear as an asterisk on the screen) and press the carriage return again. The system will print your name as a confirmation of your log-in.

```
Interlisp-D Executive
J
NIL
5+(LOGIN)
Login: WEISBLATT (password) *****
WEISBLATT
6+(LOAD '{ERIS}<LISP>KOTO>LIBRARY>LAFITE.DCOM)

{ERIS}<LISP>KOTO>LIBRARY>LAFITE.DCOM;1
compiled on 12-Mar-85 03:14:25
FILE CREATED 12-Mar-85 03:12:23
LAFITECOMS

{ERIS}<LISP>KOTO>LIBRARY>LAFITEBROWSE.DCOM;1
compiled on 12-Mar-85 00:56:03
FILE CREATED 12-Mar-85 00:52:59
LAFITEBROWSECOMS

{ERIS}<LISP>KOTO>LIBRARY>LAFITESEND.DCOM;1
compiled on 12-Mar-85 00:58:30
FILE CREATED 12-Mar-85 00:51:52
LAFITESENDCOMS

{ERIS}<LISP>KOTO>LIBRARY>LAFITEMAIL.DCOM;1
compiled on 12-Mar-85 00:57:43
FILE CREATED 12-Mar-85 00:52:27
LAFITEMAILCOMS
{ERIS}<LISP>KOTO>LIBRARY>LAFITE.DCOM;1
7+
```

*Figure 1. The executive window. The user has logged in and loaded Lafite*

You can now load Lafite. Type (LOAD '{FILESERVER}<DIRECTORY>LAFITE.DCOM) at the prompt. (If the DIRECTORIES variable in your initialization file contains a search path to the directory on which Lafite is stored, you can just type (LOAD 'LAFITE.DCOM)). The system will print the file's full name, then give the date the file was created. When it is finished loading the file, it will again print the file name and return you to the top-level prompt.

To load NS Mail, type (LOAD '{FILESERVER}<DIRECTORY>NSMAIL.DCOM).

To load Lafite Find, type (LOAD '{FILESERVER}<DIRECTORY>LAFITEFIND.DCOM).

Finally, to load Mail Scavenger, type (LOAD '{FILESERVER}<DIRECTORY>MAILSCAVENGE.DCOM).

You must log into the network before you can use Lafite. If you have not logged in already, do so now, following the instructions given in chapter 3.

In order to obtain your mail, you must be an *authenticated user*; i.e., your name and password must be accepted by your mail service. When you start up Lafite, it obtains your name and password from the Lisp executive and submits them to the authenticator for verification. If the authentication fails, you are prompted to log in. Lafite will not allow you to send or receive messages until you have been authenticated.

You can start Lafite by typing one of two commands at the prompt in the executive window: (LAFITE) or (LAFITE 'ON). Lafite will automatically supply your Active Mail folder. You can look at another folder, say *MAILFOLDER*, by typing (LAFITE 'ON *'MAILFOLDER*). You can also type (LAFITE 'ON NIL) to start Lafite without opening any mail folder, then later select a folder from the Browse menu after you enter Lafite; see chapter 5, "Reading Messages." You can also write and send (but not read, move, or otherwise manipulate) Lafite messages by selecting SendMail from the background menu; see chapter 9, "Sending Messages."

---

## The Status Window

---

When you start Lafite, two windows will appear on your workstation's screen: the *status window* and the *browser window* (the latter will not appear if you typed (LAFITE 'ON NIL)). The other two important types of Lafite windows are the *message display window* and the *message composition window*. The browser and message display windows are fully discussed in chapter 5, "Reading Messages," and the message composition window is covered in chapter 9, "Writing Messages."

The Lafite status window contains a region for Lafite status information and a small, fixed menu of commands (see figure 2). When you first start Lafite, the status region informs you that Lafite is initializing and displays the time. This message quickly disappears, to be followed by either "New Mail for *YourName*" or "No New Mail at *Time*." Your in-boxes are checked for new mail approximately every five minutes; Lafite reports in the status region if you have new mail. If you click in the status region, the background process wakes up and reports status immediately instead of waiting its normal interval.



Figure 2. The status window, showing the status region, the title bar, and the menu of commands



After you have sent a message an additional window, your *out-box*, appears beneath the status window; it lists the dates and subjects of the messages you sent most recently. See chapter 10, "Sending Messages," for more information on the out-box.

The status window (and all other Lafite windows except the message display window) has its own fixed menu of commands, which are printed in boldface type. You invoke a command by moving the mouse until the cursor points at that command, then clicking the left mouse button. When you hold down the button, the command will appear *inverted* (i.e., white letters on a black background). When you release the button, the command will appear *grayed* (i.e., black letters on a gray background). If, while holding down the mouse button, you change your mind about the command you intend to select, simply move the cursor off the inverted command name until it is restored to its normal state, then release the button. In some cases, clicking the middle mouse button instead of the left performs some different but related operation or brings up an additional menu; such commands are described below.

The menu region of the Lafite status window contains three commands—Browse, Send Mail, and Quit. The Browse command is discussed in chapter 5, "Reading Messages," and the Send Mail command is discussed in chapter 10, "Sending Messages." The Quit command is discussed in chapter 11, "Leaving Lafite."

If you are using this manual as a tutorial, ask several people in your company to send you messages before going on to chapter 5, "Reading Messages."

## 5. Reading Messages

### The Browse Command

To access your mail from the Lafite status window, click the left mouse button on the Browse command. A menu will appear listing all your mail folders (see figure 3). Your new mail is conventionally stored in your Active Mail folder, and additional folders are added to the menu when you browse them or move messages into them. The creation of mail folders is discussed in chapter 8, "Filing Messages."



Figure 3. A folder menu, showing that the user has only one mail folder (the Other Folder item is used to create new folders)

To read the mail contained in a particular folder, such as Active Mail, click on the name of that mail folder with the left mouse button.

### The Browser Window

Clicking on the name of a folder in the Browse menu opens a browser window onto that folder (see figure 4). The main part of the browser window displays a table of contents, which gives a one-line summary of each message, and a horizontal menu above the table of contents that contains commands that apply specifically to that folder. The topmost part of the browser window is the *prompt region*. In the prompt region, the system prints status information related to the browser and sometimes prompts you for information. The black *title bar* just above the table of contents labels the window.

| Reading table of contents... done.                    |        |                  |                                                 |
|-------------------------------------------------------|--------|------------------|-------------------------------------------------|
| Display                                               | Delete | Undelete         | Answer Forward Hardcopy Move To Update Get Mail |
| Mail browser for {ERIS}<GRIMBLE>MAIL>LAFITEFIG,MAIL;1 |        |                  |                                                 |
| 1                                                     | 17 Apr | Frances Grimble: | Displaying a selected message [153 chars]       |
| 2                                                     | 17 Apr | Frances Grimble: | Message marks [135 chars]                       |
| 3                                                     | 17 Apr | Frances Grimble: | "Delete" and "Undelete" [146 chars]             |
| 4                                                     | 17 Apr | Frances Grimble: | Reshaping message display windows [156 cha      |
| 5                                                     | 17 Apr | Frances Grimble: | Thumbing [131 chars]                            |
| 6                                                     | 17 Apr | Frances Grimble: | "Move to" and "Browse" [145 chars]              |
| 7                                                     | 17 Apr | Frances Grimble: | "New Mail" and "Get Mail" [148 chars]           |
| 8                                                     | 17 Apr | Frances Grimble: | "Hardcopy" [133 chars]                          |
| 9                                                     | 17 Apr | Frances Grimble: | Composing messages [141 chars]                  |
| 10                                                    | 17 Apr | Frances Grimble: | Recipient names [138 chars]                     |
| 11                                                    | 17 Apr | Frances Grimble: | "Deliver" [132 chars]                           |
| 12                                                    | 17 Apr | Frances Grimble: | Public distribution lists [147 chars]           |

Figure 4. A browser window, showing the prompt region, the menu of commands, the title bar, and the table of contents

The table of contents is an index of the messages in the mail folder being viewed. The first time you browse a folder Lafite has to

construct the table of contents, which it reports in the prompt region as "Parsing folder." Each summary line in the table of contents is numbered (the numbers are unique to that table of contents) and contains the date sent, the sender's name, the subject, and the number of characters in the message.

Each entry can also have a single-character *message mark*, which is an additional tidbit of information about the message displayed in the summary line. A message originally comes with a question mark, meaning it is unexamined. Some Lafite commands change the mark automatically. For example, displaying a message removes the question mark. You can change the mark directly by selecting with the mouse in the narrow area immediately to the left of the message number, where the mark is printed. Simply click in the mark position and type in the new mark. The message mark is the only piece of information in the table of contents you can directly modify.

---

## The Get Mail Command

---

The table of contents in your folder does not display your new messages until Lafite retrieves them from your in-box. To instruct Lafite to move the contents of your in-box to your current mail file, point the cursor at the Get Mail command and click the left mouse button. The command then appears on a gray background and the prompt region reports on the activities of your mail server(s). When all the messages have been transferred, the gray background disappears and the prompt window tells you how many new messages you have. The table of contents is updated to include the new messages placed in your mail file; it is scrolled to display the first one.

Lafite automatically places a question mark next to each unread message in the table of contents (except for the ones you sent to yourself).

---

## The Display Command and Selection Pointer

---

To read a message, you must first select its entry in the table of contents. After obtaining new mail from your in-box, Lafite automatically selects the first new message for you and places a *selection pointer* (a small black triangle pointing to the right) next to it. When you first browse a folder, Lafite selects the first unread message; if there are no unread messages in the folder, Lafite places the selection pointer next to the last message.

To display a selected message, point the cursor at the Display command in the browser window menu and click the left mouse button. A message display window will open and fill up with the message you selected (see figure 5). At this point, the question mark (if any) next to its entry in the table of contents disappears.

Message 584 from {ERIS}\XGRIMBLE\MAIL\ACTIVE,MAIL;1 [1248 chars]  
 Date: 18 Apr 85 12:17  
 From: Frances Grimble;PARC:xerox  
 Subject: GODZILLA (1984)  
 To: Frances Grimble;PARC:Xerox

GODZILLA  
 A film review by Mark R. Leeper

Way back in the early fifties, Toho Pictures of Japan made a serious monster film inspired by BEAST FROM 20,000 FATHOMS. The film was called GOJIRA (pronounced GO-jee-RA) and was reportedly about how the Americans used a nuclear bomb to try to kill a centuries-old dragon or dinosaur that was worshiped by the natives of a local island. The enraged and now radioactive monster vented his wrath on Tokyo. The film became an allegory of the closing days of World War II. Japan was being hit by something incomprehensively powerful of unknown origin that just totally wiped out any place it appeared. Finally a courageous Japanese scientist uses his own powerful weapon against Gojira, but only after he has taken safeguards

*Figure 5. A message display window, showing the title bar and the message display region. The message display window lacks the message composition window's prompt region, command menu, and TEdit pop-up menus; you cannot use it for editing*

To examine the next message listed in the table of contents, click on Display again. The selection pointer will move to the next entry, and this message will replace the previous one in the message display window. Lafite will skip over deleted messages when advancing the selection pointer. To display a deleted message you must select it explicitly, then click on Display.

You may explicitly select any entry in the table of contents by moving the cursor into the line desired and clicking the left mouse button. Any existing selection pointers will be removed, and a new one will be placed at the indicated entry. You can have several messages selected at once. If you want to add a message to the current selection, click the middle mouse button instead of the left. To remove a message from the current selection, hold down the shift key while clicking any mouse button.

If you want to select several entries in a row, place the cursor at the first message you want to read and click the left mouse button. Then move to the last message you want to read and click the right mouse button. Lafite will place selection pointers next to all the messages in between (deleted messages are not included unless the control key—Props on an 1108 or 1186 workstation—is down). You may then read them in succession by clicking on Display each time you finish reading a message.

If you have selected more than one message, clicking on Display after you read the last selected message will cause the cursor to cycle back to the first selected message. The only way to break this cycle is to explicitly select a single message.

If you want several messages displayed on the screen at once, you must create additional message display windows. To do so, click on Display with the middle mouse button. Lafite will prompt you to create a window. After you have done so, the message you selected to display will appear in the new window, and the previously displayed message will remain in the original message display window. You may create as many message display windows as you wish.

## Text Scrolling

---

Lafite browser and message display windows may contain more text than you can see at any one time. (This is also true of message composition windows, which are discussed in chapter 9, "Writing Messages.") You can view this text by using a scroll bar to scroll the window's contents (see figure 6). The Lafite browser window has two scroll bars, one just beyond the left margin and one just below the bottom margin. The message display window has only the left scroll bar. The left scroll bar is used for vertical scrolling, and the lower scroll bar is used for horizontal scrolling.

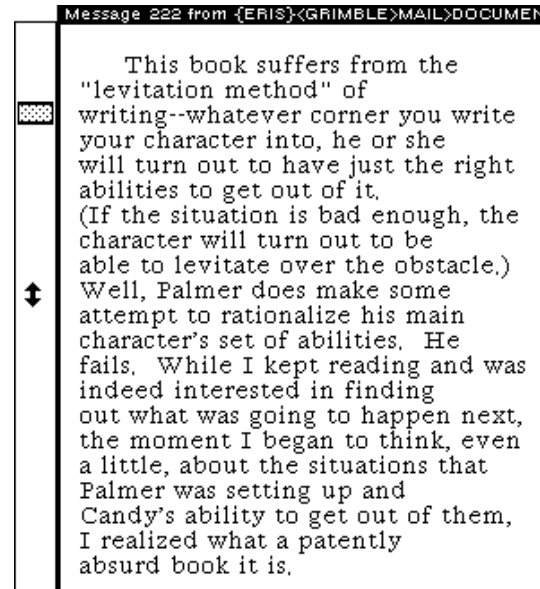


*Figure 6. A message display window showing a vertical scroll bar. The gray area represents the position of the text being read in relation to the message as a whole; the double-headed shape of the cursor means that the message is not currently being scrolled*

A scroll bar can be brought into view by moving the mouse cursor onto that area of the screen. The cursor appears as a double-headed arrow until you begin scrolling, when it changes shape to point in the correct direction. The gray rectangle within the scroll bar represents the position of the text you are viewing in relation to the file as a whole. Within a vertical scroll bar, the left mouse button scrolls the contents of the window up, and the right mouse button scrolls them down. Within a horizontal scroll bar, the left mouse button scrolls the contents of the window to the left, and the right mouse button scrolls them to the right.

You can control the amount of text that is scrolled by moving your mouse cursor to different positions on the scroll bar. Scrolling with the cursor at the bottom of the vertical scroll bar moves the text about one window length at a time. Placing the cursor at the top of the bar moves the text about one line at a time. There is less variation in the amount of text you can scroll using a Lafite browser horizontal scroll bar, but placing the cursor at the left end scrolls the text by smaller amounts than placing it on the right. To produce continuous scrolling, hold down the mouse button; release it when the part of the file you want to read is shown.

---

## Text Thumbing

---

If you want to skip to a distant part of the text quickly, *thumbing* is better than scrolling. Thumbing is analogous to opening a book by placing your thumb at the approximate position of the section you want to read and pulling the book open at that point. To thumb the contents of a Lafite window, place the cursor in the part of the scroll bar that represents the part of the file you want to read. For instance, to reach the top of a file, place the cursor at the top of the vertical scroll bar. Then press the middle mouse button. The cursor will become a gray triangle pointing toward the window. When you release the mouse button, the text will move to show the part you want to read. If you haven't thumbed to quite the right place, you can find the text you want by scrolling.

## 6. Deleting Messages and Updating the Mail Folder

---

---

### The Delete Command

---

After examining the messages in your mail file, you may wish to delete some of them. Select the messages you want to delete, then invoke the Delete command by pointing the cursor at it and clicking the left mouse button. Lafite will draw a black line through each deleted entry in the table of contents (see figure 7).

| Display                                               | Delete | Undelete         | Answer                                   | Forward | Hardcopy | Move To | Update | Get Mail |
|-------------------------------------------------------|--------|------------------|------------------------------------------|---------|----------|---------|--------|----------|
| Mail browser for {ERIS}<GRIMBLE>MAIL>LAFITEFIG,MAIL;1 |        |                  |                                          |         |          |         |        |          |
| 7                                                     | 17 Apr | Frances Grimble: | "New Mail" and "Get Mail" [140 chars]    |         |          |         |        |          |
| 8                                                     | 17 Apr | Frances Grimble: | "Hardcopy" [133 chars]                   |         |          |         |        |          |
| 9                                                     | 17 Apr | Frances Grimble: | Composing messages [141 chars]           |         |          |         |        |          |
| 10                                                    | 17 Apr | Frances Grimble: | Recipient names [138 chars]              |         |          |         |        |          |
| 11                                                    | 17 Apr | Frances Grimble: | "Deliver" [132 chars]                    |         |          |         |        |          |
| 12                                                    | 17 Apr | Frances Grimble: | Public distribution lists [147 chars]    |         |          |         |        |          |
| 13                                                    | 17 Apr | Frances Grimble: | "Get," "Put" and "Save Form" [151 chars] |         |          |         |        |          |

Figure 7. A browser window showing a table of contents from which three messages have been deleted

When processing newly arrived mail, you may wish to delete a message immediately after displaying it. In this case, Lafite will automatically select and display the next message (assuming you have not examined or deleted it).

---

### The Undelete Command

---

Messages marked for deletion are not actually removed from your mail file at the time the Delete command is given. If you discover that you have inadvertently deleted a message you want to keep, select it, point the cursor at the Undelete command, and click the left mouse button. The line drawn through the table-of-contents entry of the selected message will be removed. You can select several messages and delete or undelete them all at once; see chapter 5, "Reading Messages," for instructions on selecting messages.

---

### The Update Command

---

With the exception of retrieving new mail, any change you make to a folder being browsed, such as deleting a message or changing a message mark, is not made permanent until you issue the Update command. That is, if you were to boot your machine, start up Lafite, and browse the same folder again, you would find it in the same state as when you last updated (or as when you last browsed

it if you never updated). The Update command ensures that your mail file accurately reflects the contents of the browser, so that you could return to the same state if for any reason your machine failed and you had to start up again from scratch. It also provides the opportunity to reclaim the space occupied by deleted messages. Thus, it is important to regularly update your browsers.

When you click on the Update command, a menu appears listing your options: Write Out Changes Only or Expunge Deleted Messages (see figure 8). Ordinarily, you want to choose Expunge Deleted Messages, which, in addition to writing out all changes, permanently removes all deleted messages and renumbers the remaining ones. However, Expunge can take a long time if your folder is large and there are many “holes” in it caused by deleting messages, since Expunge has to compact the file. Thus, if you are processing a lot of mail and just want to be sure periodically that your changes are saved, with the intention to expunge the folder later, you can choose the Write Out Changes Only option, which writes out your changes without compacting out the deleted messages. This option is usually faster than Expunge.



Figure 8. The update options menu

---

## Other Ways to Update

---

When you attempt to close or shrink a browser window, using the Close or Shrink commands from the standard right-button window menu, Lafite will check whether you have made any changes to the folder, and if so, prompt you with a menu giving your updating options: Write Out Changes Only, Expunge Deleted Messages, or Don't Update File. Choosing either of the first two options is the same as clicking on Update and choosing that option, followed by closing or shrinking the window, as you requested. If you choose Don't Update File, then no update occurs—your changes are not saved.

If the only change you made to the folder was to retrieve new mail to it, or move messages into it from another folder (see chapter 8, “Filing Messages”), then you are instead offered the options of Update Table of Contents or Just Close. The former is, in this case, all the updating that the folder requires.

You may find it convenient to keep the browser for your Active Mail folder always on your screen, shrinking it down to an icon to save screen space when you are not reading your mail (see figure 9). If you do this regularly, you may never need to use the Update command directly, instead combining the update with the shrinking of the window.



Figure 9. The icon for a shrunken mail folder, in this case Active Mail. You can shrink any Lafite window by placing the mouse cursor on the title bar of the window, pressing the right



*button, and selecting Shrink from the menu of window commands that appears*

To print a copy of a message in your mail file, select it in the table of contents and click the Hardcopy command with the left mouse button. Several status messages will appear in the Interlisp-D prompt window reporting on the activities of your printer. When your message has been printed, its mark is changed to *h*. Lafite considers hard copy to be printed as soon as it has been accepted by the printer. There may be some additional delay before your message is actually printed.

You may select several messages to be printed together. Each message will be printed on a separate page (unless you have customized Lafite to batch printing requests).

You can print an unsent message from a message composition window using the right-button window menu's Hardcopy command; see chapter 9, "Writing Messages."

Most users prefer not to keep all their messages in their one Active Mail folder, but rather disperse them into several different folders, usually sorted by topic. This makes it easier to locate old messages of interest. Lafite allows you to create additional mail folders as you wish and to move your messages freely among them.

## The Move To Command

To move messages from one folder to another, select the message(s) that you wish to move. Then click on the Move To command with the left mouse button. A menu will appear listing your existing mail folders, plus the item "Other Folder." If you wish to move the messages into one of the existing folders, simply select the folder's name from the menu. Lafite then prompts you to confirm the move: "Click LEFT to confirm move to *FOLDER*." If you click the left mouse button, Lafite completes the move, deleting the message(s) from the first folder's table of contents and marking them with an *m*. If you click the right mouse button, the move is aborted.

If you wish to move the messages into a folder not listed on the menu or to a new folder, select the "Other Folder" item. Lafite then prompt you with "*FILENAME* (CR to abort):" in the prompt region. Type the name of the folder, followed by a carriage return (or a bare carriage return if you change your mind and want to abort the command). Lafite then prompts you as above to confirm the move: "Click LEFT to confirm move to *FOLDERNAME*," followed by "[*OLD FILE*]" if the file you named already exists, or "[*NEW FILE*]" if it doesn't (see figure 10). If you click the left mouse button, Lafite performs the move, creating a new folder if necessary, and adds the folder's name to the menu for future selection.

| Click LEFT to confirm move to DOCUMENTATION                                               |        |          |                    |                                               |          |        |          |  |  |
|-------------------------------------------------------------------------------------------|--------|----------|--------------------|-----------------------------------------------|----------|--------|----------|--|--|
| Display                                                                                   | Delete | Undelete | Answer             | Forward                                       | Hardcopy | Update | Get Mail |  |  |
| Mail browser for {ERIS}<GRIMBLE>MAIL>LAFITEFIG.MAIL;1 -- Default "Move To": DOCUMENTATION |        |          |                    |                                               |          |        |          |  |  |
| ►                                                                                         | 1      | 17 Apr   | Frances Grimbble:F | Displaying a selected message [153 chars]     |          |        |          |  |  |
|                                                                                           | 2      | 17 Apr   | Frances Grimbble:F | Message marks [135 chars]                     |          |        |          |  |  |
|                                                                                           | 3      | 17 Apr   | Frances Grimbble:F | "Delete" and "Undelete" [146 chars]           |          |        |          |  |  |
|                                                                                           | 4      | 17 Apr   | Frances Grimbble:F | Reshaping message display windows [156 chars] |          |        |          |  |  |
|                                                                                           | 5      | 17 Apr   | Frances Grimbble:F | Thumbing [131 chars]                          |          |        |          |  |  |
|                                                                                           | 6      | 17 Apr   | Frances Grimbble:F | "Move to" and "Browse" [145 chars]            |          |        |          |  |  |
|                                                                                           | 7      | 17 Apr   | Frances Grimbble:F | "New Mail" and "Get Mail" [148 chars]         |          |        |          |  |  |
| m                                                                                         | 8      | 17 Apr   | Frances Grimbble:F | "Hardcopy" [133 chars]                        |          |        |          |  |  |
|                                                                                           | 9      | 17 Apr   | Frances Grimbble:F | Composing messages [141 chars]                |          |        |          |  |  |
|                                                                                           | 10     | 17 Apr   | Frances Grimbble:F | Recipient names [138 chars]                   |          |        |          |  |  |
|                                                                                           | 11     | 17 Apr   | Frances Grimbble:F | "Deliver" [132 chars]                         |          |        |          |  |  |
|                                                                                           | 12     | 17 Apr   | Frances Grimbble:F | Public distribution lists [147 chars]         |          |        |          |  |  |
|                                                                                           | 13     | 17 Apr   | Frances Grimbble:F | "Get," "Put" and "Save Form" [151 chars]      |          |        |          |  |  |

Figure 10. A browser window showing a Move To confirmation prompt and the name of the current default Move To folder

In either case, once the move is complete, the information "Default 'Move To': *FOLDERNAME*" appears in the title bar of the browser.

This is for the benefit of the “accelerated” form of the Move To command. If you select the Move To command with the middle mouse button, Lafite performs the move to the folder named in the title bar without bringing up a menu of choices.

If you are already viewing the destination folder of a Move To in a browser on your screen, the newly moved messages are appended immediately to the browser’s table of contents (although the browser is not scrolled to show them, as it would be with the Get Mail command).

---

## How to Create New Folders

---

A new folder is automatically created if you ask to move mail into a folder that doesn’t exist yet. You can also create a new folder without using the Move To command. Click on the Browse command from the Lafite status window and select “Other Folder” from the menu that appears. A small prompt window will appear with the prompt “FILENAME (CR to abort)” (see figure 11). Type the desired name of the folder and press the carriage return. If no folder of that name already exists, Lafite prompts you to confirm that you want a new folder, to make sure that you didn’t merely mistype the name of an existing folder: “Click LEFT to confirm creating *FOLDERNAME*.” If you click the left mouse button, Lafite will create the folder and prompt you to shape a browser window for the folder by showing a rectangle of dashed lines. The browser’s table of contents will initially be blank, and the prompt region will report “Folder is empty.” You can now retrieve new mail to this folder, using the Get Mail command, or move messages into it from another folder, as described above.



*Figure 11. The prompt window for creating a new mail folder from the status window. The user first selected the Browse command, then the Other Folder item from the folder menu*

---

## How to View Other Folders

---

When you start up Lafite without specifying which folder you wish to view, Lafite automatically browses your default folder (usually ACTIVE.MAIL). If you instead wish to view a different mail folder at start-up time, then start Lafite by typing (LAFITE 'ON 'FOLDERNAME) to the executive window. Once Lafite is started, you can view any folder by clicking on the Browse command from the Lafite status window and selecting the folder from the menu offered, or selecting “Other Folder” and typing its name.

## The Forget Folder Command

Once you create a folder, it remains in your folder menu (the one offered by the Browse and Move To commands) until you either expunge all messages from the folder, thereby deleting the folder, or you explicitly ask that the folder be removed from the menu. This latter way is needed, for example, if you have browsed various public mail folders that you no longer wish to see in your folder menu.

You can remove a folder from the Browse menu by invoking the Browse command with the middle mouse button. The resulting menu provides four options: Browse, Forget Folder, Forget Message Form, and Browse Laurel File (see figure 12). Browse Laurel File can be used only within Xerox. The Browse command is the same as the one in the Lafite status window, and the Forget Form command is discussed in chapter 9, "Writing Messages." To use the Forget Folder command, simply click on it with the left mouse button and select the folder you want to remove with the same mouse button. The Interlisp-D prompt window will tell you that the folder has been forgotten.



Figure 12. The middle-button Browse menu, showing the options for browsing files and deleting menu items

"Forgetting" a folder only removes the folder from the menu; it does not delete the file. You can always browse the folder again by selecting Other Folder from the menu and typing the folder's name. Within Lafite, you can permanently delete a folder by deleting all its messages, expunging it, and closing its browser window. In this case, the folder's name is automatically removed from the folder menu. You can also delete a folder outside of Lafite by using the File Browser Lisp Library package to delete and expunge the folder and its table-of-contents file (named *FOLDERNAME.MAIL-LAFITE-TOC*), or using the function *DELFILE* to delete those two files, by typing *(DELFILE '{FILESERVER}<DIRECTORY>FOLDERNAME)* to the top-level executive. In either case, Lafite is unaware that you have deleted the folder behind its back, so you must use the Forget Folder command if you wish to remove the folder's name from the folder menu.

*Writing* a message is the process of composing its header and body. *Sending* a message is the process of transmitting the message to its specified recipients. This chapter gives instructions for writing and editing messages, and chapter 10 tells you how to send them.

Lafite provides five kinds of forms you can use for writing messages. You may *compose* a new message, or *answer* one you have received, or *forward* existing messages to a new recipient. You can also use specialized message forms to *report* problems or suggestions regarding Interlisp, Lafite, or TEdit, or use message forms that you have personally composed.

All the message forms have the same essential structure, but they have different kinds and amounts of information already filled in when presented to you for further editing. Some message forms are built by browser commands; others from the Send Mail command. The characteristics common to every type of message form are described first, then the differences between the forms. Once you select a message form, the process of writing and sending each kind of message is the same.

---

## The Message Composition Window

---

Each type of form is contained in a message composition window (see figure 13). The message composition window contains four regions. The topmost region is the prompt region, where Lafite gives you messages and sometimes prompts you for information. Below it is a menu with two commands, Deliver and Save Form; these are explained in chapter 10. The black title bar informs you that you are in the message editor. At the bottom is the message composition region, which has two parts separated by a blank line: the header and the body.

Figure 13. A message composition window showing a Standard message form. The top three lines of the text compose the skeleton message header; the text >>Message<< shows where you can begin the body of your message

## The Message Header

---

The Standard form shown in figure 13 contains a *skeleton message header*. This header has two blank *fields* to be filled in that provide information to your recipient and enable Lafite to deliver your message. These are the Subject field and the To field. The third field, the CC field, is optional; Lafite automatically inserts your name in it.

The skeleton message header contains only the fields that you normally need to fill in yourself to send a message. There are a large number of fields that can possibly appear in a message header, as you can see from viewing the existing mail in your mail folders. Some fields (the date and sender information) are added automatically when Lafite delivers the message; some are added by mail servers along the way; and some are fields that you can add yourself if there is a need.

Each separate field of the header begins on a new line with a word immediately followed by a colon. The word preceding the colon specifies the meaning of the text following the colon. For example, To: specifies that the words following the colon are to be interpreted as recipient names, and Subject: specifies that the text following the colon is the subject of the message, which is of interest to the recipient but not part of the delivery information.

Lafite identifies items it expects you to replace by supplying keywords between reversed, double angle brackets. To make it easier to fill in these items, the first field is highlighted as a delete selection. Typing any text will cause the keyword and brackets to be deleted. You can then fill in the field with the text of your choice. Each successive field (except the CC field) can be reached by pressing the Next key on the 1108 or 1186 keyboard (or the middle blank key on the 1132 keyboard). The final item in most message forms is >>Message<<, where you can begin the body of your message. You can also select fields with the mouse.

Because Lafite allows you to edit any part of your outgoing messages, you must observe certain rules to keep them intelligible to Lafite. The header must contain at least one recipient, and the names of multiple recipients in one field must be separated by commas. Fields must be separated by single carriage returns, so that each begins on a different line. The header and body must be separated by a double carriage return, leaving a blank line between them when the message is displayed in the Lafite window. All of Lafite's initial message forms have a blank line following the header. Do not remove it or insert a blank line inside the header; otherwise information contained in the header will be misunderstood by Lafite.

## The Subject Field

The Subject field is used to state the topic of your message (see figure 14). The topic should accurately express the content of your message so that interested people will take the time to read the message, but uninterested people can delete it without reading it. An accurate Subject field also facilitates differentiating the message later on from others in the same folder. For example, if your message contains ideas for improving Lafite, the topic might be "Suggestions for new printing features in Lafite," rather than "Lafite" or "Suggestions."

| Deliver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Save Form |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| Message Editor                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |
| Subject: REVIEW: Time's Arrows                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |           |
| To: Books:PARC:Xerox                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |           |
| Reply-to: JimDay:PASA:Xerox                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |           |
| Title: Time's Arrows                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |           |
| Author: Morris, Richard                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |           |
| Publisher: Simon & Schuster                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |           |
| Date: 1984                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |           |
| <p>Richard Morris holds a Ph.D. in Theoretical Physics and is the author of five previous books on science. In this book he presents a fascinating overview of the attitudes toward time that have influenced Western thought, science, and technology. We all know what time is, don't we? It's what a clock measures. Nothing could be simpler. But considered in detail, the topic of time is seen to be very complex, with many puzzles for philosophers, psychologists, and physicists.</p> |           |

Figure 14. A message composition window showing several filled-in user-specified fields. Note that, because the message is addressed to a public distribution list, the sender has included a Reply-To field

## The To Field

The To field (and the CC field, see below) is where you specify who is to receive your message. Lafite allows you to specify multiple recipients in the To field, provided their names are separated by commas. Conventionally, your main recipients are listed in the To field and the CC field is used to copy the message to additional recipients.

You need to know the registered address of each individual or group to whom you want to send a message. You can often discover these addresses by looking at the headers of incoming messages.



**Recipient Addresses.** NS recipient addresses have three parts—a *name*, a *domain*, and an *organization*—separated by colons, in the form “Cheryl Jones:PARC:Xerox.” A name can represent either a person or a group. Every name belongs to a particular domain, which is usually a small division of an organization that shares common resources, such as file servers and printers. And every domain belongs to an organization, which is usually a single company or a major division within a company. Thus, Cheryl Jones is an employee at PARC, a division of the company Xerox.

Within a domain, a person or group can have several alternative names known as *aliases*. An alias can always be used in place of a person or a group's real name. Thus, if Cheryl Jones's alias is Jones, her mail can be addressed to Jones:PARC:Xerox. If you want to create an alias for your name, see your network administrator.

Lafite allows you to omit the domain and/or organization for recipients who are in your own domain and organization, just as you may omit an area code when telephoning your neighbor across the street. However, you must include the domain and organization for other recipients. If a recipient's domain or organization is left out and is not the same as your own, Lafite will redisplay the message, and the Interlisp prompt window will tell you which recipient's address needs clarification.

Your default domain and organization (the variables CH.DEFAULTDOMAIN and CH.DEFAULTORGANIZATION) are usually set in your initialization file, or are set automatically if your network has only one accessible domain. You can change them by typing (DC INIT) to bring up a DEdit window on your initialization file, resetting the variables, then typing (MAKEFILE '{FILESERVER}<DIRECTORY>INITFILENAME) to save the new version of the file. For more information on DEdit and the MAKEFILE function, see the *Interlisp-D Reference Manual*.

**Public Distribution Lists.** The mail system provides a way to address messages to groups of recipients called *public distribution lists*. NS public distribution list addresses have the same form as individuals' addresses. Using a distribution list as the recipient of a message causes the message to go to all the individuals included in the group. For example, the To line “To: AISBU:PARC:Xerox” will cause the message to be delivered to all the members of the Artificial Intelligence Systems Business Unit in the Palo Alto Research Center domain.

Public distribution lists are stored and maintained on a Clearinghouse by a list owner or a network administrator; individual members can add themselves to some lists. If you want to create, add your name to, or remove your name from a public distribution list, contact your list owner or administrator, or use the NS Maintain program (documented in the *Lisp Library Packages Manual*).

While you are permitted to address a message to any public distribution list, you should think very carefully about your choice of message and list so as not to bother recipients with messages they don't care to read. Check with experienced users to see which lists should be used for which kinds of messages (see also appendix A, “Using Lafite Courteously”). Also, add a Reply-To field if appropriate (see below).

You need not worry about a recipient appearing in more than one distribution list. The mail system will detect duplicate names among recipients and ensure that each receives the message only once. Thus, you may use multiple distribution lists freely within the To and CC fields of the message header.

NS Maintain enables you to obtain lists of names, aliases, domains, organizations, and members of groups on your Clearinghouse network. For information, see the *Lisp Library Packages Manual*.

## The CC Field

The CC field is used to list persons other than your main recipient(s) who are to receive your message. Names must be separated from each other by commas. When you send your message, these people will automatically receive it along with those specified in the To field. Lafite automatically puts your name in the CC field; you can delete it if you wish. If you wish to send no copies of the message, not even to yourself, delete the entire field.

## The From and Sender Fields

Lafite automatically supplies a From field containing your name, which is not visible before the message is delivered. If you want the message to appear to be "from" someone else (e.g., if you are sending the message from somebody else's logged-in Lafite), or from more than one user, you can supply your own From field. In this case, Lafite will supply a Sender field to show who actually sent the message (see figure 15).

```

Message 84 from {ERIS}<GRIMBLE>MAIL>ACTIVE,
Date: 26 Jun 85 13:28
Sender: Frances
Grimble;PARC;xerox
Subject: I need help moving!
From: Cheryl Jones;PARC;Xerox
To: Frances.
 Grimble;PARC;Xerox

I'm moving from a house in
Sunnyvale to an apartment in
Mountain View this weekend.
Some of my furniture is too
heavy for me to handle, so I am
having a moving party on
Sunday in the hopes of getting
some help. The address is 550

```

Figure 15. A message display window showing the From and Sender fields. In this case, the sender (Cheryl Jones) wrote and sent the message from the recipient's logged-in Lafite

## The Date Field

During delivery, Lafite inserts a Date field in the form "Day Month Year Time [in international hours]." Thus, if you send a message on April 1, 1985 at 1:45 p.m., the Date field will read "1 Apr 85 13:45." You cannot supply your own Date field.

## The Reply-To Field

When sending a message to a distribution list, or to a large number of individual users, it is important to keep in mind your audience and what you want to have happen if someone wishes to answer your message. Very often it is the case, especially with large distribution lists, that you want replies to be sent only to you, or to some small designated set of people. For example, you send a message announcing a meeting and want people to tell you if the

time is bad for them. Or you are taking a poll. Or you are simply making an announcement to which you don't even expect replies. Message system etiquette demands that messages not be inappropriately sent to large numbers of people, as would ordinarily occur if a recipient of your message used the Answer command and didn't take the time to notice that a distribution list was copied on the resulting message.

It is for this reason that the Reply-To field exists. If a message contains a Reply-To field, most message-handling systems, including Lafite, will compose answering messages addressed only to the user(s) named in the Reply-To field, not to the entire recipient list of the original message. To add a Reply-To field to your message, simply insert a line beginning "Reply-to:" followed by one or more addresses (in the same form as the To field), anywhere in the header (but conventionally at the end of the header). Be sure you still have a blank line between the header and the body of the message.

---

### The In-Reply-To Field

---

When you use the Answer command (see below) to reply to a message, Lafite automatically adds an In-Reply-To field to the message form. This field contains your name and the date of the message you are answering, in the form "In-reply-to: Cheryl Jones:PARC:Xerox's message of 5 Mar 85 17:29." You can edit it if you wish.

---

## The Message Editor

---

You have available to you the full power of TEdit when composing messages. Lafite message composition windows are TEdit windows and are used in the same way. Because TEdit has many more features than can be covered here, we will describe only the most basic procedures for writing and editing messages. If you are familiar with TEdit, you can skip this section. If, after reading this section, you want to know more about TEdit, see the TEdit documentation in the *Lisp Library Packages Manual*.

---

### The Type-in Point

---

The blinking caret in the message composition window is called the *type-in point* (see figure 16). The Interlisp environment has only one type-in point active at a time; you tell the environment you want to type in the message composition window by clicking a mouse button on that window. Now whatever text you type will appear at the type-in point.

| Deliver                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         | Save Form |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| <p>Message Editor</p> <p>One of my earliest memories was going to see the film WAR OF THE WORLDS. I was not yet three years old and my parents, who usually hate science fiction, for some reason went to see it. I hated it. And we sat through it twice. By the time I was six, I would have sold both my parents into slavery to see the film again. I was bitten by science fiction early and hard. And the paragon of science fiction writers had to be H. G. Wells, I thought.</p> <p>Finding in the library the Dover book SEVEN SCIENCE FICTION NOVELS BY H. G. WELLS was a high point of my youth. I remember how.</p> |           |

Figure 16. A message composition window showing the type-in point and text wraparound. The caret after "how" is the type-in point. The text will be adjusted automatically when the window is reshaped or the message is printed

TEdit automatically breaks text between words and sends the overflow to the next line. Don't use a carriage return when you think you have reached the end of a line; use one only when you want to insert blank space between paragraphs or sections. TEdit automatically adjusts the text of your message to fit the width of your recipient's message display window or your printer's requirements, but it leaves blank lines in place.

## Text Selection

You may want to change the text of your message after you type it in by deleting it, copying it, or moving it. To change the text of your message, you first say where you want the change made by making a selection. Then you say what you want done by giving a command. Just making a selection has no effect on the message; only commands can change it. Each selection supersedes the previous one; that is, there is only one selection at a time. Deletion and other operations are applied to the currently selected text. You can tell when text is selected because it is highlighted by underlining or reverse video (usually white-on-black); it may have a caret flashing at one end.

There are two regions within an editor window, and which region the mouse cursor is in determines what kind of selection happens. The regions are the area containing text and the line bar that forms the left border of the window (this is not the same as the scroll bar described in chapter 5).

You select text by first pointing with the mouse, then pushing one of its buttons. The left mouse button always selects the smallest units. In the text region, it selects the character you're pointing at; in the line bar, it selects the single line you're pointing at.

The middle mouse button selects larger units. In the text region, it selects the word the cursor is over, and in the line bar it selects the paragraph the cursor is next to.

The right mouse button always extends a selection. If you select a place in the text with the left or center mouse button, move the mouse cursor somewhere else, and click the right mouse button, all

the text in between the two points is selected. If the existing selection is a whole line or paragraph, the extended selection is also a whole line or paragraph.

## Text Deletion

---

There are three ways you can select text for deletion. If you hold down the control key while selecting text, the selected text will be deleted when you release the key. You can also delete text by selecting it and then pressing the Delete key. You can delete text one character at a time by pressing the backspace key; the character to the left of the caret will disappear. Figure 13 shows some text highlighted as a delete selection.

## Text Moving

---

To move text, first click the left mouse button to put the caret in the place you want the text moved to. Then select the text to be moved while holding down the Move key (or the control and shift keys on an 1132 workstation). When you release the key(s), the text will be moved to where the caret is. You can use this method to move text within a message composition window or to move it from one window to another.

## Text Copying

---

If you want to copy text, first put the caret in the place you want the text copied to. Then hold the Copy key down while selecting the text to be copied. When you release the Copy key, the text will be copied to the location of the caret. You can copy text within a window or from one window to another. However, the most efficient way to copy an entire file from one window to another is to use the Include or the Get command; these are described below.

## The Include Command

---

It is often desirable to electronically mail a document, such as a report, that is not a letter. You can conveniently do this by including that file in a Lafite message. First, you open a message composition window and fill in the header information. Next, you type in any covering message you want to send and place the blinking caret where you want to insert the file. Then put the mouse cursor on the black title bar of the window and press the left or middle mouse button.

The basic TEdit command menu will pop up; it contains the commands Put, Get, Include, Find, Looks, Substitute, Quit, and Expanded Menu (see figure 17). While you can use all these commands from within Lafite, only the Include, Put, and Get, commands are described in this manual. For detailed information you should consult the TEdit documentation.



Figure 17. The basic TEdit command menu. Selecting Expanded Menu will bring up a menu containing most of the commands on the

*basic menu, plus commands that enable you to access additional menus—the Character Looks, Paragraph Looks, and Page Layout menus*

To include a file in a message, select Include with the left mouse button. TEdit will ask you for the name of the file you want to load. Type the file name in the prompt region and press the carriage return. After a pause to read the file, your message composition window will fill up with the designated file. The entire file will be highlighted by underlining. To get rid of the underlining, simply click the left mouse button anywhere in the file.

Be careful to give TEdit the exact name of the file you want to include; do not change spacing or punctuation. If TEdit tells you that it can't find a file, check to see if you typed the name correctly.

## The Put Command

You can save messages in TEdit files using the Put command and finish or deliver them later on. Bring up the TEdit command menu as described above and select Put with the left mouse button. TEdit will ask you for the name of the file to put to (see figure 18). The syntax for naming files stored on your disk is {DSK}FILENAME. For files stored on a file server, it is {FILESERVER}<DIRECTORY>FILENAME. You don't need to specify a file server or directory if you want TEdit to use your default file server and directory. Make sure that no blank spaces appear in your file name.

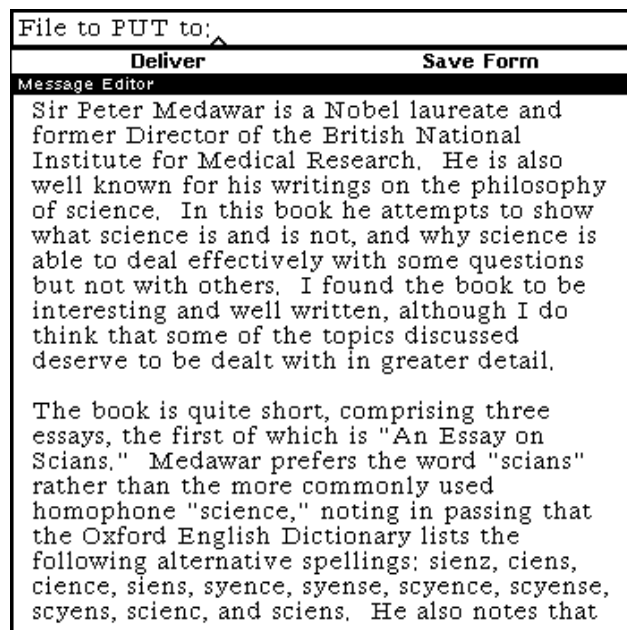


Figure 18. A message composition window after the user has chosen the Put command. The message will be saved and retrieved as a TEdit file

Type the file name in the prompt region and press the carriage return. If you have already saved the file, TEdit will prompt you with the file name you specified earlier. You can either change this file name or confirm the earlier name by pressing the carriage return. TEdit will tell you that it is putting the file, then that it is done. You can now close your message composition window and later retrieve the message with the Get command.

The Put and Get commands are very useful for saving messages that can't be delivered due to network problems. (Or you can shrink the message composition window like any other Interlisp window; the icon looks like an envelope and is labeled "Unsent.")

You can also save messages with the Save Form command and retrieve them from the middle-button Send Mail menu; see below.

## The Get Command

---

To retrieve a previously composed, undelivered message, bring up a Lafite message form and select Get from the TEdit menu with the left mouse button. TEdit will prompt you for the file to get. Type its name in the prompt region and press the carriage return. The mouse cursor will change to an hourglass until TEdit finishes getting the file. Once the file is loaded, you can edit or send your message. If you get a Lafite mail folder into TEdit and edit it, change the file's name when you save it. If you continue to use the file as a Lafite folder, Lafite will be unable to parse the folder the next time you open it (see chapter 12, "Troubleshooting Lafite Problems").

## The Hardcopy Command

---

You can use the Hardcopy command in the Interlisp-D window menu to print messages before you send them. You bring up the window menu by placing the mouse cursor on the title bar of a message composition window and pressing the right mouse button. The menu of window manipulation commands shown in figure 19 will appear. Select Hardcopy and wait for TEdit to tell you it is done formatting the message for print. You may have to wait a little longer before your printer has it ready for you.

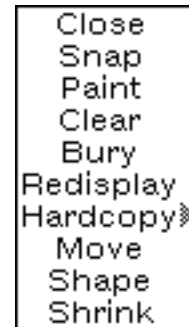


*Figure 19. The Interlisp-D right-button window menu. The TEdit documentation in the Lisp Library Packages Manual fully describes how to use this menu with the text editor*

## Fancy Message Formats

---

Lafite supplies you with an attractive default message design. If you don't specify otherwise, your messages are displayed in a 12-point Times Roman font. They are printed in a 12-point Classic font on an Interpress printer or a 12-point Times Roman font on a Press printer. They have no line or paragraph leading, and the margins are set flush left and ragged right.

If you wish, you can change this design using TEdit's font and formatting options. (You can also change Lafite's default fonts by resetting global variables; see chapter 13, "Customizing Lafite.") These include several additional fonts and type sizes, boldface and italic type, justified margins, and tabs. Detailed instructions on designing documents with TEdit are contained in the TEdit documentation. You can send specially designed messages either formatted (containing the special format) or unformatted (in the default design). See chapter 10 for instructions on sending formatted messages.

## Messages With Bit Maps

Bit maps can be included in messages sent to other Lafite users (see figure 20). To insert a bit map in a Lafite message, first get the picture onto your screen. Place the caret in the place in the message composition window where you want the image to appear. Hold down the shift key and depress the right mouse button in the background. This will bring up a rectangle for Snap, a window menu command that makes a copy of any portion of your screen and allows it to be placed elsewhere on the screen, for instance in a message composition window. Move the cursor into the rectangle, turning it white-on-black. When you release the key and button, the cursor changes from an arrow to a prompt design. Move the prompt (as if it were the cursor) to the bit map you want to include in the text. Press down the left button and outline the bit map as closely as possible, in the same way you would create a region for a window. When you let up the left button, the bit map will appear where you left the caret flashing in the message composition window.

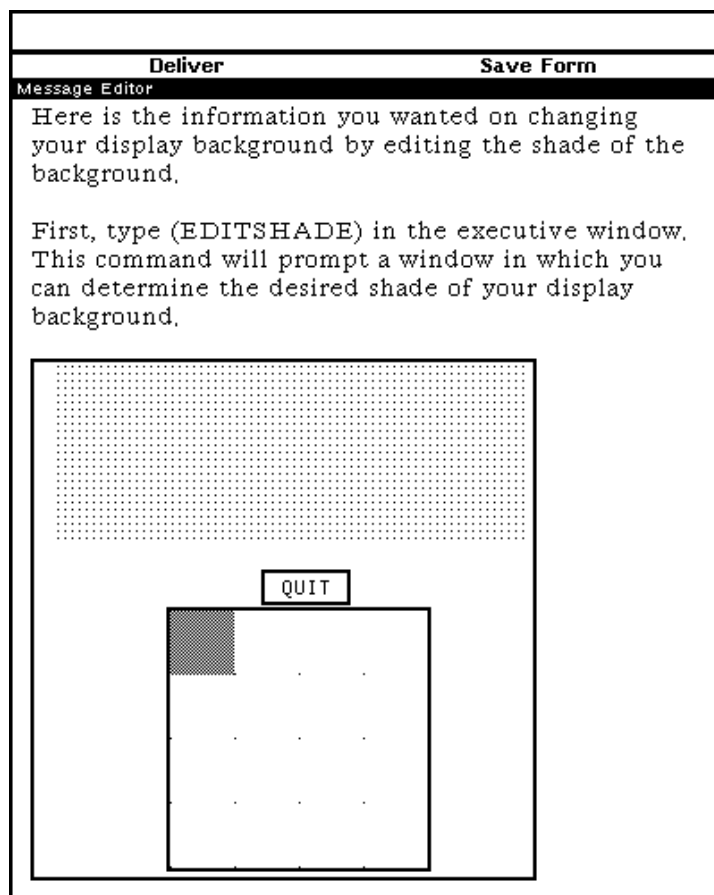


*Figure 20. A Lafite message containing a bit map. It is often convenient to include a screen image in a Lafite message when you are giving technical information or reporting problems with the system*

You can edit a bit map by clicking inside it with the left mouse button, which brings up a menu of editing operations. You can delete, copy, or move a bit map as if it were a single text character. If you insert a carriage return before and after a bit map, you can adjust its position with the TEdit paragraph-formatting menu (see the TEdit documentation). Bit maps are saved when you save a file, and they are printed when you hard-copy the message (providing your printer can print bit maps).



Messages containing bit maps are automatically sent as formatted messages; see chapter 10.

---

## Message Forms

---

As described above, Lafite provides five kinds of forms you can use for writing messages, each of which is accessed by a different command or commands. The kinds of forms are the Standard form, the *Answer* form, the *Forward* form, the *Report* form, and the *Personally Created* (or saved) form. This section describes each form and how to use it.

---

### The Standard Form

---

The Standard message form is a message composition window containing blank To and Subject fields and a CC field with your name in it (see figure 13). You can access the Standard message form in two ways. The first is to use the Send Mail command in the Lafite status window. If you click on this command with the left mouse button, Lafite will open a window containing the standard message form. If you click on Send Mail with the middle mouse button, Lafite will present you with a menu of several forms, including the names of any you have created yourself plus Lisp Report, TEdit Report, Lafite Report, Saved Form, and Standard Form (see figure 21). To access the Standard form, select it from the proffered menu. (Personally Created forms and Report forms are discussed below.)



Figure 21. The middle-button Send Mail menu. In this case the user has no Personally Created forms

You can also access this second menu without otherwise entering Lafite by clicking on your screen background menu with the right mouse button and selecting SendMail from the menu that pops up. If you choose SendMail without having entered Lafite from the executive window, you cannot read, move, or otherwise manipulate messages—only write and send them using the Standard, Personally Created, and Report forms.

---

### The Answer Form

---

Selecting the Answer command from the browser window menu constructs a message form that is a skeleton reply to the selected message or to the first selected message if more than one is selected (see figure 22). The header of the form contains To and CC fields listing the sender of the message and its recipients, a Subject field of the form "Re: *Subject of selected message*," and an In-Reply-To field containing the name of the sender and the date and time the message was sent. For example, if you are replying to a message sent to you by Cheryl Jones on March 19, 1985 at 10:28 a.m., the field will be "In-Reply-To: Cheryl Jones:Domain:Organization's message of 19 Mar 85 10:28."

| Deliver                                                                                                                                                                          | Save Form |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| Message Editor                                                                                                                                                                   |           |
| Subject: Re: Gold of the Immigrants Bread<br>In-reply-to: Fournier's message of Wed, 26<br>Jun 85 08:50<br>To: Fournier;PASA;Xerox<br>cc: Grimbale,PARC;Xerox<br><br>>>Message<< |           |

Figure 22. An Answer form, showing the skeleton message header created by Lafite. The user can edit the header as he or she desires

The form provided by the Answer command is not always exactly what you might want. Lafite takes the viewpoint that it is easier to delete than to add text, and therefore includes all information that seems to be relevant. For example, Lafite often includes your name in the CC field of the answer form, even though you may not want a copy of the message you are composing. You should not expect that the answer form will be exactly right; always examine the header to be certain it contains the desired information.

In particular, if you reply to a message that was directed to a distribution list, the Answer command will copy the distribution list name into the CC field of the Answer form (unless the sender included a Reply-To field). You should consider carefully whether it is appropriate for your reply to be sent to that distribution list, and if not delete the distribution list's name.

After you deliver the answer to a message, an *a* will appear in the browser window as that message's mark.

## The Forward Form

Selecting the Forward command from the browser window menu constructs a message form containing the complete text of all the selected messages, and a message header whose subject is the sender and subject of the first selected message, in brackets. You should fill in the desired recipients and a covering message if you desire one. You can also change the subject if you wish.

After you deliver a forwarded message, an *f* will appear as that message's mark.

## Specialized Report Forms

The Report forms are used to report problems or make suggestions regarding Interlisp or an application program. Although they always appear on the middle-button Send Mail menu, you cannot use them unless your organization has set up a local support address or addresses to which you can send the reports (see chapter 13, "Customizing Lafite," for instructions). If you choose one of the Report forms and an address has not been set up, a message will be printed in the prompt window saying that the address is not

available. If you cannot use the Report forms, please move on to the section below on creating customized message forms.

The following three sections describe the headers of the Report forms, including the information that is provided automatically and the fields you fill in. We have assigned meanings to the fields and made recommendations for reporting problems and requesting features. These guidelines are included because we have found them to be effective and feel they may be helpful to our users. However, they may differ from the policies of your local support organization, so check with that organization before using these forms.

## **The Lisp Report**

---

The Lisp Report has a header similar to that of the Standard form, plus a section giving certain information about the hardware and software you are using and asking you to supply information about your problem or feature request. The Subject field is filled in with "Lisp: >>Terse summary of problem<<." Replace the words in reverse angle brackets with a subject that accurately identifies your problem or request. Subjects like "Floppy problem" are imprecise; instead, use subjects like "Attempt to write file when floppy door is open causes awful noise." Feature requests generally start with the word "Want," e.g., "Want to make windows circular rather than rectangular."

The To field contains the address of your local Lisp support. You do not need to modify it. The CC field initially contains your name.

Lafite automatically fills in the date your sysout was made, the kind of workstation you are using, the microcode version, and the memory size. In addition, there is a Frequency field containing the words ">>Always, Intermittent, Once<<." The Frequency field is used to report the reproducibility of a problem; it is usually irrelevant for feature requests. If the problem occurs every time you try to perform a task, choose Always and delete the angle brackets and the Intermittent and Once options. (This is most easily done by copy-selecting Always while the bracketed field is delete-selected.) If the problem doesn't always happen, choose Intermittent; if you saw it happen once, choose Once. If the Frequency field is irrelevant, leave it blank or delete the whole line.

Use the Impact field to describe how seriously a problem or the lack of a feature affects your ability to get work done. The names apply to bug reports, but feature requests should be rated along analogous lines. Choose Fatal if the problem causes system crashes, loss of work, etc., or if a feature is required for project completion. Choose Serious if the problem can be worked around but seriously interferes with your work. Choose Moderate if the problem is tolerable but clearly a problem. Choose Annoying for annoying problems and requests for features that "would be nice." Choose Minor for very minor feature requests or problems that may not even be bugs.

Replace the words "detailed problem description" with a description of the problem and what you were doing when you encountered it, plus any other information that might be useful. Always be as precise and detailed as possible; it is extremely difficult to solve a technical problem described only as "Floppy doesn't work."

## **The Lafite Report**

---

The Lafite Report differs from the Lisp Report in that the Subject field is filled out as "Subject: Lafite: >>Terse summary of problem<<" and the Lafite system date appears above the Lisp system date (see figure 23). You should report problems and requests for new features in the same way as in a Lisp Report.

| Deliver                                                 | Save Form |
|---------------------------------------------------------|-----------|
| Message Editor                                          |           |
| Subject: Lafite: >>Terse summary of problem<<           |           |
| To: LafiteSupport.pa                                    |           |
| cc: Grimbale.pa                                         |           |
| Lafite System Date: 12-Mar-85 03:12:27                  |           |
| Lisp System Date: 16-Apr-85 13:02:15                    |           |
| Machine: Dandelion (131#61#)                            |           |
| Microcode version: 26,24                                |           |
| Memory size: 5777                                       |           |
| Frequency: >> Always, Intermittent, Once <<             |           |
| Impact: >> Fatal, Serious, Moderate, Annoying, Minor << |           |
| >>detailed problem description<<                        |           |

Figure 23. The Lafite Report form, showing the fields you need to fill in and the ones Lafite fills in automatically

## The TEdit Report

The TEdit Report is exactly like the Lafite Report, except that "TEdit" appears in the subject instead of "Lafite," and the TEdit system date is given instead of the Lafite system date.

## Personal Message Forms

You can create, save, and reuse Personal message forms or form letters with Lafite's Save Form command. Simply write your message in the message composition window, then choose Save Form from the window's menu with the left mouse button. Lafite will print the following message in the prompt region: "Save form under name:." Type the name you want the form to have (it cannot contain a space) and press the carriage return. If no extension is given for the file name, Lafite defaults it to Lafite-Form. Lafite will print the full file name in the prompt region, then close the message composition window.

When composing your Personal form, you will probably want to use the same convention as the regular Lafite forms do for items that will need to be filled in when the form is used to send a message. That is, enclose a describing word or phrase in reverse angle brackets, so that TEdit's Next key can be used to easily skip from one such item to the next. In addition, if you include the text ">>Self<<" anywhere in the message, it will be automatically replaced with the name of the logged-in user when the form is requested. The most common use of this is to put the line CC: >>Self<< in the header. This is better than putting your own name there, since your form can be used by other people without having to edit the CC field. Of course, if you edit such a form and save it away again, you must remember to replace your name with >>Self<< again, since the >>Self<< is replaced with your name when you bring up the form.

There are two ways you can access Personally Created forms. After you save a form, its name is added to the middle-button Send Mail menu, and you can select it just as you would any of the other forms on that menu. You can also select Saved Form from the same menu; this brings up a small window asking for the file name. Type in the name of the form and press the carriage return.

When you select, edit, and deliver a Personally Created form, you are only editing a copy of the form; the original remains unchanged. Of course, if you want to change the actual form you can edit it and save it again with the Save Form command.

Note that Save Form is intended for the creation of new message *forms*. If you merely wish to save a partially composed message for later completion, TEdit's Put command is more appropriate.

---

## The Forget Message Form Command

---

Once you have saved a form, its name will continue to appear in the middle-button Send Mail menu. If you no longer want that form in the menu, use the Forget Message Form command in the middle-button Browse menu to remove it. Select Browse with the middle mouse button, then Forget Message Form from the proffered menu (see figure 12). This brings up a menu of Personally Created forms; select the one you want to forget with the left mouse button. The Interlisp-D prompt window will tell you the form has been forgotten. Note that, as with the Forget Folder command, this does not delete the actual file. To delete the file, type `(DELFILE '{FILESERVER}<DIRECTORY>FILENAME.LAFITE-FORM)` in the executive window. You can also use the File Browser Lisp Library package.

Once you have written the message you want to send, you can initiate its delivery to the recipient by pointing the cursor at Deliver and clicking the left mouse button. Lafite will fill in your name and the date (though they won't appear in the window) and proceed to send the message. A gray background will appear behind the Deliver command, then the command will change to Abort (see figure 24). If you decide not to send a message after buttoning Deliver, select the Abort command and Lafite will halt the delivery of the message.

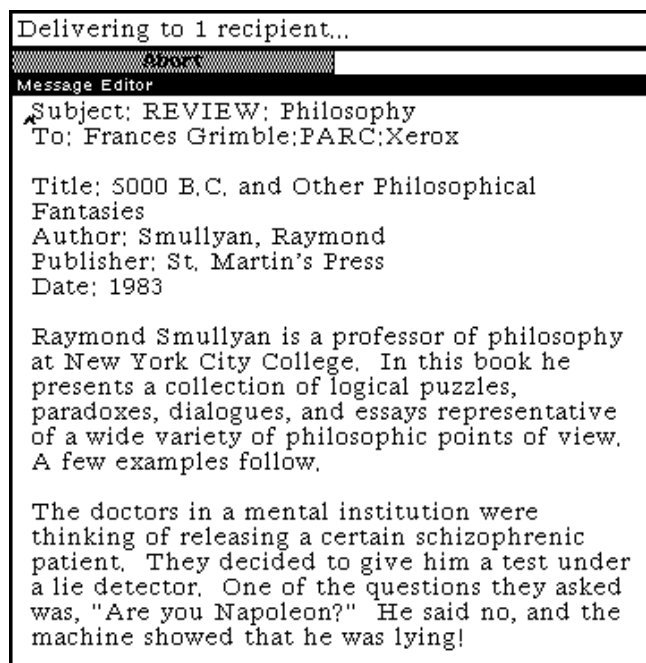


*Figure 24. A message composition window after Abort was chosen. After Lafite aborts the delivery, the word "abort" will appear in the prompt region and the message will be redisplayed*

Invoking Deliver will send the message to all its intended recipients. Each recipient will receive only one copy of the message, even if his or her name appears more than once (a name might be on several distribution lists specified in the To or CC field).

If Lafite discovers an error in the list of recipients, it will tell you that the "recipient is not understood" and give you an opportunity to correct the mistake and invoke Deliver again. When the list is acceptable (i.e., all specified recipients are known to have in-boxes), Lafite will deliver the message. After successful delivery, the word Abort will disappear. Lafite will tell you that it is done and close the message composition window. Delivery happens in the background, so you can perform other tasks while delivery proceeds.

After a message is delivered, it is entered into your out-box, a window attached to the bottom of your status window (see figure 25). This window contains one-line descriptions of the two messages you sent most recently. The out-box is treated as a

menu—selecting a line in it brings up the corresponding message for further editing and delivery. The out-box can be independently closed if you are no longer interested in the messages displayed therein.



*Figure 25. A status window showing the out-box. The number of messages retained can be changed using the LAFITEOUTBOXSIZE variable; see chapter 13, “Customizing Lafite”*

If you decide not to do anything with a message you started to compose, simply close the message composition window with the standard right-button window menu. Lafite asks you to confirm flushing the message; its text will not be saved.

---

## Delivery of Formatted Messages

---

Only Lafite users can read a formatted message; all other mail readers will see only the plain text of the message. If you try to deliver a formatted message, Lafite asks if you want to retain the formatting information, putting up a menu with the choices Send Formatted Message, Send Plain Text, and Abort. Send Formatted Message retains all the formatting information. This choice is made automatically if the message contains a bit map, because there is no way to send the image without the formatting. If you send a message containing a bit map to a non-Lafite user, the content of the bit map will appear as a string of trash, which may disturb some message-reading systems.

Send Plain Text sends only the text of the message. The message will appear to the recipient in whatever font his or her mail reader usually uses, and all paragraph formatting (centering, justification, special tab stops) will vanish. Sometimes a message appears to Lafite to be “formatted” when it really only contains inconsequential variations in fonts or line spacing, typically the result of copy-selecting text from another window. In such cases, where you did not intend special formatting, it is best to choose Send Plain Text to reduce the message overhead and avoid imposing your default font choice on your recipients.

Abort does not send the message, but returns you to the message editor to continue editing the message.

There are two ways you can leave Lafite. You can select Quit from the status window (or the middle-button Quit menu) which stops Lafite and closes all browser windows. If any of your open mail folders need updating, Lafite prompts you with a menu asking what degree of updating should be performed: Write Out Changes Only, Expunge Deleted Messages, Don't Update File, and Don't Quit; the option you select is applied to all open folders (see figure 26). You must select one of the first three options before Lafite will let you quit. When you do, all your Lafite windows will disappear, indicating that you have successfully quit Lafite.

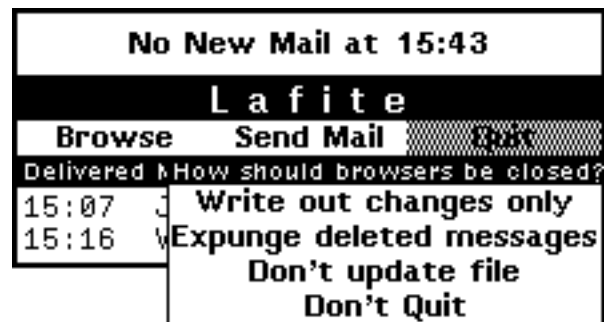


Figure 26. The update menu that appears when you choose Quit and an open mail folder (or folders) needs updating

You can also leave Lafite by typing (LAFITE 'OFF) in the executive window, which has the same effect as selecting Quit.

If you select Quit with the middle mouse button, you get a menu that includes the original Quit command and the Restart command. The Restart command turns Lafite off and then immediately back on again—it is equivalent to typing (LAFITE 'OFF) followed by (LAFITE 'ON NIL). It is mainly useful if you have changed some of your Lafite personalization and you want Lafite to notice the new settings.

You may want to keep the Lafite status window in view instead of quitting Lafite each time you finish reading or sending a batch of mail. The status window keeps you constantly informed about the state of your in-boxes, and it takes up little room on your workstation screen.





[This page intentionally left blank]

Although Lafite is a robust system, you may occasionally encounter minor problems when using it. This chapter describes the most common Lafite problems and how to prevent and/or recover from them.

---

### Lafite Confuses You With a Previous User

---

If you log into Lafite with the correct name and password but the Browse menu shows no folders or shows someone else's folders, Lafite may be confusing you with a previous user. If you are running in a sysout in which someone else has been using Lafite, you need to take some action to get Lafite to work on your mail files and in your name. When you change the user identity by logging in as yourself, Lafite notices that the current user has changed and attempts to authenticate you. However, Lafite still doesn't know how you want your Lafite customized; in particular, what your known mail folders are.

To get Lafite to recognize your identity, you should first turn Lafite off by choosing Quit from the status window or by calling (LAFITE 'OFF). Then log in again. Type (GREET) in the executive window; you will be asked for the name of your initialization file if there isn't one on your workstation's disk. When this has been loaded, restart Lafite.

---

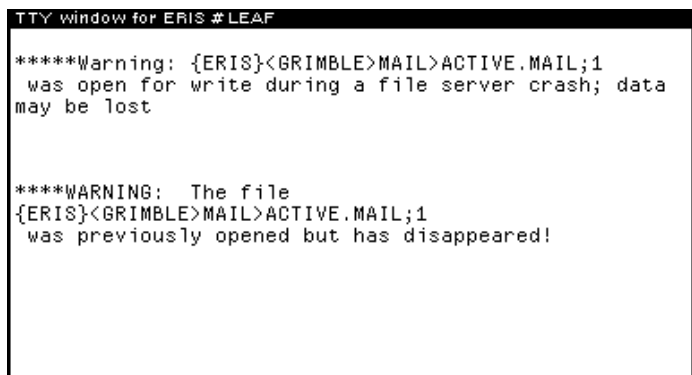
### File Server Is Slow

---

If your mail files are stored on a remote file server that is particularly unresponsive, the mail server connection over which new mail is retrieved may *time out* (that is, end communication with your file server) before the file server acknowledges receipt of the messages. The usual consequence of this is that your in-box is not flushed, so your new mail is in two places: your in-box, awaiting retrieval, and your mail file, to which it was just retrieved. A less common occurrence is that the mail server times out partway through the retrieval process, resulting in a Lisp break. You can type ^ after the prompt in the break window to return to the state before the Get Mail started.

If this is often a problem for you, you may want to adopt the following procedure to maintain the flexibility of remote mail files while utilizing the speed and reliability of the local disk. Keep most of your mail files on the remote server, as usual, but keep your Active Mail file, the one to which you usually retrieve mail, on your local disk. Retrieve mail to this file and dispatch from there to your remote files (using Move To) some or all of the messages you wish to keep. Mail files on disk have very predictable performance

during Get Mail, which is good for both you and the mail server. Files on disk are also less subject to other vagaries of remote servers (e.g., sudden crashes) that sometimes cause problems with mail files (see figure 27). And if you tend to delete much of your incoming mail after reading it once, you may find it faster to keep your Active Mail on disk, even if your remote server isn't unreliable.



```
TTY window for ERS #LEAF

*****Warning: {ERS}<GRIMBLE>MAIL>ACTIVE.MAIL;1
 was open for write during a file server crash; data
 may be lost

****WARNING: The file
{ERS}<GRIMBLE>MAIL>ACTIVE.MAIL;1
 was previously opened but has disappeared!
```

*Figure 27. A warning window opened by Lafite after a file server crash. The folder has “disappeared” because the file server is down, therefore the mail folders on it cannot be accessed by Lafite*

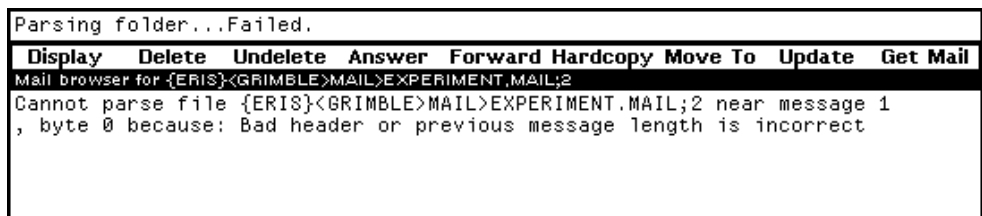
You can also choose to keep most or all of your mail files on disk, backing them up to a file server periodically. The Lisp Library package CopyFiles is helpful for doing the backup automatically.

---

## Mail File Does Not Parse

---

The first time Lafite browses a mail folder, or any time that it suspects that the folder's table of contents is obsolete, it tries to *parse* the folder. A good mail folder consists of a set of messages, each set off by some internal information about the message's length and status. Parsing a folder consists of scanning for this internal information and building a table of contents from it. If a folder has been damaged in some way, the parsing operation fails, most commonly because a message's stated length is inconsistent with what is actually stored in the file (see figure 28). In this case, Lafite aborts the Browse command and prints a message in the browser window describing how far it got. The information Lafite prints includes the header of the last message parsed and a byte pointer (character position) in the file where the problem was encountered. The mail file is valid up to that pointer position.



```
Parsing folder...Failed.

Display Delete Undelete Answer Forward Hardcopy Move To Update Get Mail
Mail browser for {ERS}<GRIMBLE>MAIL>EXPERIMENT.MAIL;2
Cannot parse file {ERS}<GRIMBLE>MAIL>EXPERIMENT.MAIL;2 near message 1
, byte 0 because: Bad header or previous message length is incorrect
```

*Figure 28. A browser window showing an error message printed by Lafite when it was unable to parse the mail folder*

The most common way to get a mail file into an inconsistent state is to abort a Move To or Update command somewhere in the middle (either manually, or because a remote server crashed). In the case of Move To, the problem is usually that the last message in the file is “too short,” since it was never completely written. If the first operation you perform on the destination file after the crash is to

browse it, Lafite will (usually) detect this situation and let you browse the file anyway, with a warning that the last message is truncated. Updating the file then corrects the length of the last message. If you neglect to browse the destination file before moving additional messages to it, however, your mail file will not parse.

Another way you can damage a mail file is to get it into a TEdit file, edit it, and save it under the original name. The next time you browse that mail folder, Lafite will be unable to parse the file.

The solution to “mail file does not parse” problems is to scavenge the mail file with the Lisp Library package Mail Scavenger, following the instructions in chapter 14, “Using Lafite-Related Lisp Library Packages.”

---

## Table of Contents Is Inconsistent With Mail File

---

Lafite breaks with a message to this effect when it tries to operate on a message that isn't where Lafite thought it was in the file. The most common cause of this is aborting the Expunge command. The appropriate action is to close the browser, select Don't Update, delete the table-of-contents file (the file with -LAFITE-TOC appended to the name) from your directory, and then browse the file again. If this is not successful, you may need to scavenge the file with Mail Scavenger. If you made many changes to the browser (deletions, for example) that you would rather not lose, you can try selecting Write Out Changes Only when you close the browser. This may succeed if the inconsistencies in the table of contents did not intersect with your changes.

Some aspects of Lafite's behavior, such as whether each message of a batch you print starts on a new page and how many messages are shown in your out-box, are controlled by *global variables* set in your initialization file. Each variable is set to a *default value* that most users have found acceptable. However, you may want to reset some global variables to tailor Lafite to your needs. You edit Lafite variables in the same way as any other part of your initialization file, by typing (DC INIT) to bring up a DEdit window on your initialization file, resetting the variables, then typing (MAKEFILE '{FILESERVER}' <DIRECTORY>INITFILENAME) to save the new version of your initialization file. For detailed information on DEdit and the MAKEFILE function, see the *Interlisp-D Reference Manual*.

---

## Lafite's Global Variables

---

LAFITEDEFAULTHOST&DIR [Variable]

is the directory on which Lafite looks for the LAFITE.PROFILE and all mail folders and message forms you access if you don't supply an explicit directory. Lafite uses its own host and directory names for mail folders, etc., rather than the current connected directory because you may want to keep your mail folders someplace special (e.g., the local disk or your log-in directory), and the connected directory can change. LAFITEDEFAULTHOST&DIR is provided to tell Lafite where you usually keep your mail. LAFITEDEFAULTHOST&DIR should be in the same form as the variable LOGINHOST/DIR (i.e., {FILESERVER}<DIRECTORY>). If LAFITEDEFAULTHOST&DIR is NIL, then the value of LOGINHOST/DIR is used; i.e., your log-in host and directory.

LAFITEFORMDIRECTORIES [Variable]

is a search path for Lafite forms, initially NIL. When you choose the Saved Form command underneath Send Mail, the form name that you enter is first searched for on your default directory (LAFITEDEFAULTHOST&DIR), and if it is not found there, Lafite searches the directories in the list LAFITEFORMDIRECTORIES. LAFITEFORMDIRECTORIES is typically set to a list of one or more public directories on which generally useful forms have been collected.

LAFITESTATUSWINDOWPOSITION [Variable]

specifies where the status window appears when Lafite is invoked. It is a POSITION or NIL (in which case you are asked to specify a position when Lafite starts).

LAFITEBROWSERREGION [Variable]

is a REGION used to describe where the primary (i.e., first) browser window is to be placed on the screen. Initially, it is set to something "reasonable" for the standard initial display configuration. If you set it to NIL, you are prompted to specify a region the first time any such window is created.

LAFITEDISPLAYREGION [Variable]

is a REGION used to describe where the primary (i.e., first) message display window is to be placed on the screen. Initially, it is set to something “reasonable” for the standard initial display configuration. If you set it to NIL, you are prompted to specify a region the first time any such window is created.

LAFITEEDITORREGION [Variable]

is a REGION used to describe where the primary (i.e., first) message composition window is to be placed on the screen. Initially, it is set to something “reasonable” for the standard initial display configuration. If you set it to NIL, you are prompted to specify a region the first time any such window is created.

DEFAULTMAILFOLDERNAME's [Variable]

value is used if no mail folder is supplied to the function LAFITE, i.e., you call (LAFITE 'ON). Initially, this value is ACTIVE.MAIL.

LAFITEMAIL.EXT [Variable]

is the default extension for names of mail folders. It is initially MAIL.

LAFITETOC.EXT [Variable]

is the string appended to the name of a mail folder to produce the name of its table-of-contents file. It is initially LAFITE-TOC.

LAFITEFORM.EXT [Variable]

is the default extension for the names of user-defined form files. It is initially LAFITE-FORM.

LISPSUPPORT [Variable]

specifies the address to which Lisp Report messages are sent. Its value is a single string containing one or more addresses, all in the form you would put in the To line of a message header. If LISPSUPPORT is NIL, then the Lisp Report form cannot be used. It is initially NIL. Site administrators may want to set this variable in their site initialization files to the mail address of their local Lisp Support.

LAFITESUPPORT [Variable]

specifies the address to which Lafite Report messages are sent. Its value is a single string containing one or more addresses, all in the form you would put in the To line of a message header. If LAFITESUPPORT is NIL, then the Lafite Report form cannot be used. It is initially NIL. Site administrators may want to set this variable in their site initialization files to the mail address of their local Lafite Support.

TEDITSUPPORT [Variable]

specifies the address to which TEdit Report messages are sent. Its value is a single string containing one or more addresses, all in the form you would put in the To line of a message header. If TEDITSUPPORT is NIL, then the TEdit Report form cannot be used. It is initially NIL. Site administrators may want to set this variable in their site initialization files to the mail address of their local TEdit Support.

LAFITEDISPLAYFONT [Variable]

is a font used for displaying messages. You may change it without changing the other Lafite fonts. It should be a FONTDESCRIPTOR as returned by FONTCREATE. In your initialization file, this is usually specified with a FONTCREATE expression, e.g., (VARS (LAFITEDISPLAYFONT (FONTCREATE 'MODERN12))). Initially, it is Times Roman 12.

LAFITEEDITORFONT [Variable]

is a font used for composing messages. You may change it without changing the other Lafite fonts. It should be a FONTDESCRIPTOR as returned by FONTCREATE. Initially, it is Times Roman 12.

LAFITEHARDCOPYFONT [Variable]

is a font used for printing messages. You may change it without changing the other Lafite fonts. It should be a FONTDESCRIPTOR as returned by FONTCREATE. Initially, it is Times Roman 12, which prints as Classic 12 on an Interpress printer.

LAFITEBROWSERFONT [Variable]

is the font used for displaying the table of contents in the browser window. It is initially Gacha 10.

LAFITEMENUFONT [Variable]

is the font used for the items in all Lafite menus. It is initially Helvetica 10 Bold.

LAFITETITLEFONT [Variable]

is the font used for the title bar ("Lafite") in the Lafite status window. It is initially Helvetica 12 Bold.

LAFITEENDOFOFMESSAGEFONT [Variable]

is the font in which the LAFITEENDOFOFMESSAGESTR is displayed (see below). It is initially Times Roman 10 Italic.

LAFITEENDOFOFMESSAGESTR [Variable]

is a string containing the text of the "End of Message" token displayed at the end of a message. If LAFITEENDOFOFMESSAGESTR is NIL, then no "End of Message" token appears.

LAFITENEWPAGEFLG [Variable]

is initially T. If so, the Hardcopy command starts each message on a new page. Otherwise it separates each message by a line of dashes.

LAFITEHARDCOPYBATCHFLG [Variable]

allows you to postpone printing messages until several can be done at once. Printing several messages at once helps to keep your short messages from being lost amongst other users' long output, saves paper, and saves time because you don't have to wait for individual messages to be formatted for printing.

When LAFITEHARDCOPYBATCHFLG is T, Lafite batches your hard-copy requests. It doesn't actually print the messages until you do an Update, at which point Lafite sends them all to the printer in one batch. When you have hard copy pending, the Hardcopy command is speckled to remind you of this fact. The Update command has an additional choice, Do Hardcopy Only, in case you want to get your batched hard copy printed without doing an actual Update.

When batching hard copy, LAFITENEWPAGEFLG applies only to the messages selected at each Hardcopy invocation; each new set of messages starts on a new page, independent of the setting of LAFITENEWPAGEFLG. LAFITEHARDCOPYBATCHFLG is initially NIL.

LAFITEHARDCOPY.MIN.TOC [Variable]

is a positive number if non-NIL. Whenever Lafite is instructed to produce hard copy for more than LAFITEHARDCOPY.MIN.TOC messages, it also produces a table of contents as a cover page for



the hard copy. Currently, this flag is noticed only if LAFITEHARDCOPYBATCHFLG is NIL. It is initially NIL.

MAILWATCHWAITTIME [Variable]

is the number of minutes between polling for new mail from your mail servers. It is initially set to five.

LAFITEFLUSHMAILFLG [Variable]

is initially T. If it is NIL, Lafite won't flush your in-box when retrieving new mail, so the mail will still be there when you invoke Get Mail again.

LAFITEIFFROMMETHENSEENFLG [Variable]

makes sure, if T, that messages sent from you are considered "seen" (and hence do not have the question mark), even though you have not yet displayed them. It is initially T.

LAFITEDISPLAYAFTERDELETEFLG [Variable]

can be set to T or ALWAYS. If T, Lafite displays the next message if you delete the message in the message display window and the next message is undeleted and unexamined (i.e., it is marked with a question mark). If ALWAYS, Lafite displays the next undeleted message even if it has already been seen. It is initially T.

LAFITEMOVETOCONFIRMFLG [Variable]

controls whether Lafite requires confirmation of the Move To command. If ALWAYS, all moves require confirmation; if LEFT, then only left-button moves (selecting the destination from a menu) require confirmation; if MIDDLE, then only middle-button moves (using the default Move To folder) require confirmation; if NIL, then no moves require confirmation. It is initially ALWAYS.

LAFITEBUFFERSIZE [Variable]

is the number of 512-character buffers used by the stream managing the file behind an open browser window. If you regularly receive very long messages, you might want to increase this to improve performance of Display followed by Hardcopy or Move To. Initially 20, which handles messages up to about 10,000 characters long.

LAFITEOUTBOXSIZE [Variable]

specifies the number of delivered messages to be retained in your out-box. As you send more messages, older ones fall off the end. Increasing this number gives you a longer "history" from which you can select and reedit old messages, but the desire for a longer history should be balanced with the knowledge that you are tying down the resources used by each of those messages. Setting LAFITEOUTBOXSIZE to zero or NIL disables the out-box feature: after delivery, messages completely vanish. LAFITEOUTBOXSIZE is initially two.

LAFITENEWMAILTUNE [Variable]

is a list of the form acceptable to the function PLAYTUNE or NIL, in which case it is ignored. It is played when Lafite discovers you have new mail waiting.

LAFITEGETMAILTUNE [Variable]

is a list of the form acceptable to the function PLAYTUNE or NIL, in which case it is ignored. It is played when a Get Mail command completes.

## How to Add a New Message Form to Lafite

The normal way to add a new message form to Lafite is to edit an existing form (or build one from scratch) and save it using the Save Form menu item (see chapter 9, "Writing Messages"). However, you can also provide message forms that compute the text at the time you request the form, as, for example, the Lisp Report does. To add your own items to the Message Forms menu, add a standard three-element menu item to the variable LAFITESPECIALFORMS and then set the variable LAFITEFORMSMENU to NIL (this is where the menu is cached). The three-element menu item should yield a LITATOM as its "value," that atom being interpreted as follows:

1. If the atom has a function definition, the function is called (with no arguments) and the returned value (a string or a TEdit text stream) is used.
2. If the atom has a value, its value (a string or a TEdit text stream) is used.
3. Otherwise, a copy of the file by that name is used.

For example, if you wanted to add a message form that contained the date the user's version of TEdit was made (as in the TEdit Report form), you could add

("TEdit Support" (QUOTE MAKETEDITFORM)

"Make a form to report a problem with TEdit")

to LAFITESPECIALFORMS; MAKETEDITFORM could be defined to be

```
[LAMBDA NIL
 (PROG (OUTSTREAM)
 (SETQ OUTSTREAM (OPENTEXTSTREAM " "))
 (printout OUTSTREAM "Subject: TEdit: >>Subject<<" T)
 (printout OUTSTREAM "To: " TEDITSUPPORT T)
 (printout OUTSTREAM "cc: " (USERNAME T)
 (printout OUTSTREAM "TEdit-System-Date:"
 TEDITSYSTEMDATE T T)
 (printout OUTSTREAM ">>Message<<" T)
 (RETURN OUTSTREAM])
```

where TEDITSUPPORT and TEDITSYSTEMDATE are variables set by TEdit. Lafite supplies one function to make this kind of message form easier to construct.

```
(MAKEXXXSUPPORTFORM 'SYSTEMNAME
 'ADDRESS 'SYSTEMDATE)[Function]
```

creates a message form (a TEdit stream) to be mailed to the maintainers of *SYSTEMNAME*. *SYSTEMNAME* is the name of the subsystem (a string); *SYSTEMDATE*, if non-NIL, is a date (string) of importance to include in the message; and *ADDRESS* is the mail system address of the intended recipient(s) or an association list in the same form as the variable LISPSUPPORT. If the *ADDRESS* is NIL, MAKEXXXSUPPORTFORM prints a message in the prompt window that the form is not supported, and returns NIL.

For example, MAKETEDITFORM is actually defined as

```
[LAMBDA NIL
 (MAKEXXXSUPPORTFORM
 "TEdit" TEDITSUPPORT TEDITSYSTEMDATE)]
```

and selecting TEdit Report in the Message Forms menu produces a form like the one in figure 29.

| Deliver                                                 | Save Form |
|---------------------------------------------------------|-----------|
| Message Editor                                          |           |
| Subject: TEdit; >>Terse summary of problem<<            |           |
| To: TEditSupport.pa                                     |           |
| cc: Weisblatt.pa                                        |           |
| TEdit System Date: 3-Oct-85 17:52:44                    |           |
| Lisp System Date: 22-Nov-85 11:32:58                    |           |
| Machine: Dandelion (222 #14 #)                          |           |
| Microcode version: 26,32                                |           |
| Memory size: 15777                                      |           |
| Frequency: >> Always, Intermittent, Once <<             |           |
| Impact: >> Fatal, Serious, Moderate, Annoying, Minor << |           |
| >>detailed problem description<<                        |           |

Figure 29. The TEdit report form that was created by following the instructions given above

## 14. USING LAFITE-RELATED LISP LIBRARY PACKAGES

There are two separate packages that can be used with Lafite: Lafite Find and Mail Scavenger. Lafite Find helps you search for particular messages in a mail folder, and Mail Scavenger helps to restore mail files that have been damaged.

### Lafite Find

Lafite Find is a package that helps you search for particular messages in a mail folder. The search is a simple string or keyword search that examines either specific header fields of a message (From, Subject) or the entire message. You can look for either one message or all messages matching a requested pattern.

#### How to Load Lafite Find

You load Lafite Find by typing (LOAD '{FILESERVER}<DIRECTORY>LAFITEFIND.DCOM). You can then bring up a menu of search commands by clicking on the black title bar of a Lafite browser window with the middle mouse button (see figure 30). Three commands are listed on this menu: Find, Find Related, and Find Again. The Find command is the general search command; the other two are special cases.

|                                                |        |                  |                                           |              |                            |
|------------------------------------------------|--------|------------------|-------------------------------------------|--------------|----------------------------|
| Reading table of contents... done.             |        |                  |                                           | Find         |                            |
| Display                                        | Delete | Undelete         | Answer For                                | Find Related | by Move To Update Get Mail |
| Mail browser for {ERIS}<GRIMBLE>MAIL>LAFITEFIG |        |                  |                                           | Find Again   |                            |
| 2                                              | 17 Apr | Frances Grumble: | Message marks [135 chars]                 |              |                            |
| 3                                              | 17 Apr | Frances Grumble: | "Delete" and "Undelete" [146 chars]       |              |                            |
| 4                                              | 17 Apr | Frances Grumble: | Reshaping message display windows [156 ch |              |                            |
| 5                                              | 17 Apr | Frances Grumble: | Thumbing [131 chars]                      |              |                            |
| 6                                              | 17 Apr | Frances Grumble: | "Move to" and "Browse" [145 chars]        |              |                            |
| 7                                              | 17 Apr | Frances Grumble: | "New Mail" and "Get Mail" [148 chars]     |              |                            |
| 8                                              | 17 Apr | Frances Grumble: | "Hardcopy" [133 chars]                    |              |                            |
| 9                                              | 17 Apr | Frances Grumble: | Composing messages [141 chars]            |              |                            |
| 10                                             | 17 Apr | Frances Grumble: | Recipient names [138 chars]               |              |                            |
| 11                                             | 17 Apr | Frances Grumble: | "Deliver" [132 chars]                     |              |                            |

Figure 30. A browser window in which the user has brought up the top-level Lafite Find menu. You can bring up the menu anywhere on the title bar by clicking the middle mouse button

#### Search Direction Commands

If you select the Find command in this first menu, you are prompted with a menu to determine the direction of search. It lists four options: Find Next One, Find Next All, Find Previous One, and Find Previous All (see figure 31). To search forward in your mail folder one message at a time, use the Next One command. To search forward for all appropriate messages, use Next All. The Previous One and Previous All commands search backward. The Next searches all search forward from the last selected message in

the browser; the Previous searches all search backward from the first selected message.

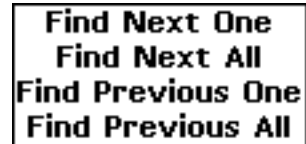


Figure 31. The search direction commands menu

The Find Next All and Find Previous All commands start the search with the currently selected message, unlike the Find Next One and Find Previous One commands, which omit the currently selected message. This is mainly so that the Find Related command includes the original message in the set of messages it selects.

## Search Type Commands

---

Selecting any of the above commands brings up another menu, prompting for the type of search. This menu lists four options: From, Subject, Body, and Related (see figure 32). The From and Subject options enable you to search for messages by the content of their From or Subject fields, the Body search examines the entire message, and the Related search is a special kind of Subject search.

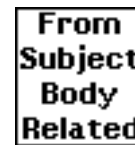


Figure 32. The search type commands menu

### The From and Subject Searches

---

The From search looks at the From fields of messages sent by others and the To fields of messages sent by you. Thus, roughly speaking, From searches look at the same names you see in the browser window between the date and subject of each message. The Subject search examines the Subject fields of the messages in your mail folder.

### The Body Search

---

The Body search can be used to locate a message by arbitrary strings in the message itself, rather than just in its From or Subject fields. For example, a Body search can locate messages that are addressed to (or cc'ed to) "Jones," or that were sent in the month of "Feb," or that mention "sugar" in their content. However, such searches are potentially much slower, since they have to search much more territory.

### The Related Search

---

The Related search is a special kind of Subject search. It looks for messages whose subject is the same as the currently selected message, plus or minus the "Re: " that the Answer command adds to a subject (see chapter 9, "Writing Messages"). The easiest way to perform a Related search is to use the Find Related command in the initial menu, which is equivalent to the sequence Find, Find Next All, Related. If you want to do a Related search backward, the Find Related command has a subcommand Find Related Backward, which you can access by rolling the mouse cursor out of the menu to the right.

Find Related does not check that the messages it finds are really related to the selected message, merely that the subject matches

the search string formed by taking the subject of the selected message and, if necessary, removing the initial "Re:." Thus, this command may find too many messages.

Find Related is very useful for locating all the messages in an interchange where everyone used the Answer command to reply to the first (or subsequent) message.

---

## How to Use Lafite Find

After you select the type of search (except for Related), you are prompted to type a string to search for. The search is case-insensitive; e.g., "Jones" matches both "JONES" and "jones." The string searches are also substring searches, so you need not type the entire user name in a From search, just enough to uniquely identify the sender.

For example, to search for the next message sent by user Jones, click Find, then Find Next One, then From. Lafite then prompts you in the prompt region with "Find From string:." Type the name of the sender (Jones) and then press the carriage return. If Lafite finds a message matching the string you typed, it selects the message and scrolls the browser, if necessary, to expose that message. If it finds no such message, it tells you so in the prompt region.

If you want to find all the messages from Jones, then follow the same procedure, but choose Find Next All instead of Find Next One. Lafite Find selects all the messages from Jones, and you can then cycle through them with the Display command, delete them all, move them all to a different folder, etc.

If the message found by a Find Next One or Find Previous One command is not the message you were looking for, or you want to locate the next such message, you can repeat the search by selecting the Find Again command in the initial (middle-button title bar) menu.

---

## Mail Scavenger

The Lisp Library package Mail Scavenger is used to rebuild the internal pointers in a mail file that has been damaged (see chapter 12, "Troubleshooting Lafite Problems"). When Lafite detects damage in a file, it usually reports "Can't parse file" and terminates its Browse command. The simplest remedy is to scavenge the file with the Mail Scavenger, then browse the file again.

---

## How to Use Mail Scavenger

To load the Mail Scavenger, type (LOAD '{FILESERVER}<DIRECTORY>MAILSCAVENGE.DCOM) (see figure 33). To call the Mail Scavenger, type (MAILSCAVENGE 'FILENAME). This scavenges, or attempts to fix, the file named FILENAME. (FILENAME defaults to the extension MAIL and your Lafite directory, exactly as with Lafite's Browse command.) Mail Scavenger will print some information in your executive window about what it is doing to correct the file, such as "Patching length field of header in message number 1." It will then ask, "Do you want to replace {FILESERVER}<DIRECTORY>MAILFILE with the newly scavenged version?" If you answer Yes, the newly scavenged file will replace the damaged mail file. If you answer No, a new version of your mail file will be created. In this case, Mail Scavenger returns the name of the temporary file it wrote,

{DSK}<DIRECTORY>FILENAME.SCAVENGE), which you can then rename or delete as you wish. Ordinarily you should reply Yes, unless you are suspicious about Mail Scavenger's report on what it had to correct.

The finished mail file that Mail Scavenger produces should always be readable; i.e., Lafite will not complain about it. However, it is a good idea to browse the file and check any messages the Mail Scavenger mentioned correcting; these may be missing several characters or be malformed in other ways. You should also check neighboring messages—some of the characters in these messages might really be parts of other messages, and some messages may be duplicates. You may want to delete the damaged messages.

```

Interlisp-D Executive
NIL
17+ (LOAD 'MAILSCAVENGE.DCOM)
=
{ERIS}<LISP>INTERMEZZO>LIBRARY>MAILSCAVENGE.
DCOM;1

{ERIS}<LISP>INTERMEZZO>LIBRARY>MAILSCAVENGE.
DCOM;1
compiled on 21-Feb-85 17:50:30
FILE CREATED 21-Feb-85 17:49:29
MAILSCAVENGECOMS

{ERIS}<LISP>INTERMEZZO>LIBRARY>BSEARCH.DCOM;
1
compiled on 7-May-84 23:20:16
FILE CREATED 7-May-84 23:19:59
BSEARCHCOMS
{ERIS}<LISP>INTERMEZZO>LIBRARY>MAILSCAVENGE.
DCOM;1
22+ (MAILSCAVENGE 'EXPERIMENT)

Rebuilding header for message number 1
Do you want to replace the mail file {ERIS}<
GRIMBLE>MAIL>EXPERIMENT.MAIL;2 with the newl
y-scavenged version? yes
{ERIS}<GRIMBLE>MAIL>EXPERIMENT.MAIL;2
23+

```

Figure 33. An executive window in which the user has loaded the Mail Scavenger and scavenged the folder Experiment.Mail

---

## How to Scavenge Files in Place

Calling (MAILSCAVENGE.IN.PLACE 'FILENAME) is similar to calling MAILSCAVENGE, except that it destructively scavenges the file in place. This is faster than MAILSCAVENGE, but you have to be brave and assume MAILSCAVENGE is not overwriting anything valuable as it scans the file.

---

## Additional Arguments to MAILSCAVENGE

If you wish, you can call the MAILSCAVENGE function with the arguments *ERRORMSGSTREAM* and *TEMPDIR* as well as *FILENAME*. *ERRORMSGSTREAM* is the stream on which Mail Scavenger writes the information about what it is doing; the default is T. You can also specify the argument *TEMPDIR*, which is the host/directory on which Mail Scavenger writes its intermediate file. *TEMPDIR* defaults to {DSK} unless you are on a workstation without a local disk file system, in which case *TEMPDIR* defaults to the same directory as *FILENAME*.

|                                                          |           |
|----------------------------------------------------------|-----------|
| <b>1. Introduction</b>                                   | <b>1</b>  |
| What Lafite Is                                           | 1         |
| What This Manual Is About                                | 1         |
| <b>2. Things to Know Before You Start</b>                | <b>3</b>  |
| <b>3. Obtaining Lafite</b>                               | <b>5</b>  |
| What Programs You Need                                   | 5         |
| How to Load the Programs You Need                        | 5         |
| <b>4. Entering Lafite</b>                                | <b>7</b>  |
| The Status Window                                        | 7         |
| <b>5. Reading Messages</b>                               | <b>9</b>  |
| The Browse Command                                       | 9         |
| The Browser Window                                       | 9         |
| The Get Mail Command                                     | 10        |
| The Display Command and Selection Pointer                | 11        |
| Text Scrolling                                           | 12        |
| Text Thumbing                                            | 13        |
| <b>6. Deleting Messages and Updating the Mail Folder</b> | <b>15</b> |
| The Delete Command                                       | 15        |
| The Undelete Command                                     | 15        |
| The Update Command                                       | 16        |
| Other Ways to Update                                     | 16        |
| <b>7. Printing Messages</b>                              | <b>19</b> |
| <b>8. Filing Messages</b>                                | <b>21</b> |
| The Move To Command                                      | 21        |
| How to Create New Folders                                | 22        |
| How to View Other Folders                                | 23        |
| The Forget Folder Command                                | 23        |
| <b>9. Writing Messages</b>                               | <b>25</b> |
| The Message Composition Window                           | 25        |
| The Message Header                                       | 26        |
| The Message Editor                                       | 31        |
| The Type-in Point                                        | 31        |
| Text Selection                                           | 32        |
| Text Deletion                                            | 33        |
| Text Moving                                              | 33        |
| Text Copying                                             | 33        |



|                                                       |                           |           |
|-------------------------------------------------------|---------------------------|-----------|
|                                                       | The Include Command       | 33        |
|                                                       | The Put Command           | 34        |
|                                                       | The Get Command           | 35        |
|                                                       | The Hardcopy Command      | 36        |
|                                                       | Fancy Message Formats     | 36        |
|                                                       | Messages With Bit Maps    | 36        |
| Message Forms                                         |                           | 38        |
|                                                       | The Standard Form         | 38        |
|                                                       | The Answer Form           | 38        |
|                                                       | The Forward Form          | 39        |
|                                                       | Specialized Report Forms  | 40        |
| The Forget Message Form Command                       |                           | 42        |
| <b>10. Sending Messages</b>                           |                           | <b>45</b> |
| Delivery of Formatted Messages                        |                           | 46        |
| <b>11. Leaving Lafite</b>                             |                           | <b>49</b> |
| <b>12. Troubleshooting Lafite Problems</b>            |                           | <b>51</b> |
| Lafite Confuses You With a Previous User              |                           | 51        |
| File Server Is Slow                                   |                           | 51        |
| Mail File Does Not Parse                              |                           | 52        |
| Table of Contents Is Inconsistent With Mail File      |                           | 53        |
| <b>13. Customizing Lafite</b>                         |                           | <b>55</b> |
| Lafite's Global Variables                             |                           | 55        |
| How to Add a New Message Form to Lafite               |                           | 59        |
| <b>14. Using Lafite-Related Lisp Library Packages</b> |                           | <b>63</b> |
| Lafite Find                                           |                           | 63        |
|                                                       | How to Load Lafite Find   | 63        |
|                                                       | Search Direction Commands | 63        |
|                                                       | Search Type Commands      | 64        |
|                                                       | How to Use Lafite Find    | 65        |
| Mail Scavenger                                        |                           | 66        |
|                                                       | How to Use Mail Scavenger | 66        |

|                                             |    |
|---------------------------------------------|----|
| <b>Appendix A: Using Lafite Courteously</b> | 69 |
| Foreward                                    | 69 |
| Introduction                                | 69 |
| Communication Patterns                      | 70 |
| The Wrong Number                            | 71 |
| Rudeness and Vulgarly                       | 73 |
| Message System Costs                        | 75 |
| Unsolicited Mail                            | 76 |
| The Chain Reaction                          | 77 |
| Off-the-Record Responses                    | 78 |
| Masquerading                                | 78 |
| Wizards Versus Naive Users                  | 79 |
| The Moral of This Tale                      | 79 |
| <b>Glossary of Technical Terms</b>          | 81 |
| <b>Quick-Reference Section</b>              | 85 |
| Message Marks                               | 85 |
| Lafite Commands                             | 85 |
| Status Window Commands                      | 85 |
| Browser Window Commands                     | 86 |
| Message Composition Window Commands         | 87 |
| <b>Index</b>                                | 89 |

---

|                                                          |           |
|----------------------------------------------------------|-----------|
| <b>1. Introduction</b>                                   | <b>1</b>  |
| What Lafite Is                                           | 1         |
| What This Manual Is About                                | 1         |
| <b>2. Things to Know Before You Start</b>                | <b>3</b>  |
| <b>3. Obtaining Lafite</b>                               | <b>5</b>  |
| What Programs You Need                                   | 5         |
| How to Load the Programs You Need                        | 5         |
| <b>4. Entering Lafite</b>                                | <b>7</b>  |
| The Status Window                                        | 7         |
| <b>5. Reading Messages</b>                               | <b>9</b>  |
| The Browse Command                                       | 9         |
| The Browser Window                                       | 9         |
| The Get Mail Command                                     | 10        |
| The Display Command and Selection Pointer                | 11        |
| Text Scrolling                                           | 12        |
| Text Thumbing                                            | 13        |
| <b>6. Deleting Messages and Updating the Mail Folder</b> | <b>15</b> |
| The Delete Command                                       | 15        |
| The Undelete Command                                     | 15        |
| The Update Command                                       | 16        |
| Other Ways to Update                                     | 16        |
| <b>7. Printing Messages</b>                              | <b>19</b> |
| <b>8. Filing Messages</b>                                | <b>21</b> |
| The Move To Command                                      | 21        |
| How to Create New Folders                                | 22        |
| How to View Other Folders                                | 23        |
| The Forget Folder Command                                | 23        |
| <b>9. Writing Messages</b>                               | <b>25</b> |
| The Message Composition Window                           | 25        |
| The Message Header                                       | 26        |
| The Message Editor                                       | 31        |
| The Type-in Point                                        | 31        |
| Text Selection                                           | 32        |
| Text Deletion                                            | 33        |
| Text Moving                                              | 33        |
| Text Copying                                             | 33        |

|                                                       |                           |    |
|-------------------------------------------------------|---------------------------|----|
|                                                       | The Include Command       | 33 |
|                                                       | The Put Command           | 34 |
|                                                       | The Get Command           | 35 |
|                                                       | The Hardcopy Command      | 36 |
|                                                       | Fancy Message Formats     | 36 |
|                                                       | Messages With Bit Maps    | 36 |
| Message Forms                                         |                           | 38 |
|                                                       | The Standard Form         | 38 |
|                                                       | The Answer Form           | 38 |
|                                                       | The Forward Form          | 39 |
|                                                       | Specialized Report Forms  | 40 |
| The Forget Message Form Command                       |                           | 42 |
| <b>10. Sending Messages</b>                           |                           | 45 |
| Delivery of Formatted Messages                        |                           | 46 |
| <b>11. Leaving Lafite</b>                             |                           | 49 |
| <b>12. Troubleshooting Lafite Problems</b>            |                           | 51 |
| Lafite Confuses You With a Previous User              |                           | 51 |
| File Server Is Slow                                   |                           | 51 |
| Mail File Does Not Parse                              |                           | 52 |
| Table of Contents Is Inconsistent With Mail File      |                           | 53 |
| <b>13. Customizing Lafite</b>                         |                           | 55 |
| Lafite's Global Variables                             |                           | 55 |
| How to Add a New Message Form to Lafite               |                           | 59 |
| <b>14. Using Lafite-Related Lisp Library Packages</b> |                           | 63 |
| Lafite Find                                           |                           | 63 |
|                                                       | How to Load Lafite Find   | 63 |
|                                                       | Search Direction Commands | 63 |
|                                                       | Search Type Commands      | 64 |
|                                                       | How to Use Lafite Find    | 65 |
| Mail Scavenger                                        |                           | 66 |
|                                                       | How to Use Mail Scavenger | 66 |

|                                             |     |
|---------------------------------------------|-----|
| <b>Appendix A: Using Lafite Courteously</b> | 69  |
| Foreward                                    | 69  |
| Introduction                                | 69  |
| Communication Patterns                      | 70  |
| The Wrong Number                            | 71  |
| Rudeness and Vulgarity                      | 73  |
| Message System Costs                        | 75  |
| Unsolicited Mail                            | 76  |
| The Chain Reaction                          | 77  |
| Off-the-Record Responses                    | 78  |
| Masquerading                                | 78  |
| Wizards Versus Naïve Users                  | 79  |
| The Moral of This Tale                      | 79  |
| <b>Appendix B: Using Lafite at Xerox</b>    | 81  |
| The Grapevine Mail System                   | 81  |
| Grapevine Protocols                         | 81  |
| Grapevine Recipient Addresses               | 82  |
| Maintain                                    | 85  |
| If You're on Both Networks                  | 89  |
| Lafite Modes                                | 89  |
| From Grapevine to NS and Vice Versa         | 90  |
| Lafite-Related Lisp Users' Packages         | 91  |
| From Laurel to Lafite                       | 92  |
| The Browse Laurel File Command              | 92  |
| <b>Glossary of Technical Terms</b>          | 95  |
| <b>Quick-Reference Section</b>              | 99  |
| Message Marks                               | 99  |
| Lafite Commands                             | 99  |
| Status Window Commands                      | 99  |
| Browser Window Commands                     | 100 |
| Message Composition Window Commands         | 101 |
| <b>Index</b>                                | 103 |

# A USER'S GUIDE TO LAFITE

XEROX

THE INTERLISP MAIL SYSTEM

Copyright © 1986, Xerox Corporation  
All rights reserved

Copyright protection claimed includes all forms and matters of copyrightable material and information now allowed by statutory or judicial law or hereinafter granted, including without limitation, material generated from the software programs which are displayed on the screen such as icons, screen display looks, etc.

# A USER'S GUIDE TO LAFITE

XEROX

THE INTERLISP MAIL SYSTEM



Copyright © 1986, Xerox Corporation  
All rights reserved

Copyright protection claimed includes all forms and matters of copyrightable material and information now allowed by statutory or judicial law or hereinafter granted, including without limitation, material generated from the software programs which are displayed on the screen such as icons, screen display looks, etc.

---

---

**LAFITEABBREV**

---

---

By: Lennart Lövstrand  
(Lovstrand.EuroPARC@Xerox.COM)

Requires: LAFITE, NSMAIL

INTERNAL

This document last edited on December 21, 1988.

## INTRODUCTION

As the name suggests, LAFITEABBREV gives you the ability to refer to mail addresses using personal, easy to remember abbreviations. If you define (say) "Joe" to stand for "John Doe:Xyzzy North:ACME Corporation", then all you need to type to send a message to Joe is *Joe*. If you don't want to be bothered with his long address even when you read messages from him, you can optionally let the abbreviation apply both ways. When receiving messages from him (or where you both are recipients), his full address will be replaced by your abbreviation and thus only show as simple *Joe* in all header lines and the Lafite browsers.

In addition to this rather straight forward string-to-string text substitution, LAFITEABBREV will also do pattern matching using multiple wildcards in both the abbreviation and fully expanded strings. This is of course particularly useful for cases when many people share the same substring in their addresses, such as the NS domain. Say for example that Jane Doe worked at the same place as her brother. Then we could abbreviate "\*:Xyzzy North:ACME Corporation" as "\*:X:ACME" and refer to John as "John Doe:X:ACME" and Jane as "Jane Doe:X:ACME". In fact, we could even create the abbreviation "\* Doe"  $\Rightarrow$  "\* Doe:Xyzzy North:ACME Corporation" and refer to our friends as just "John Doe" and "Jane Doe" with the obvious expansions.

Abbreviations are purely personal; all other recipients of the message will see the real, expanded addresses. The abbreviations are on a strictly one-to-one mapping basis, ie. you can't implement private DLs by making an abbreviation expand into more than one address. As is the case with normal mail addresses, all abbreviations and expansions are case-insensitive.

## EXAMPLES

|                             |         |                   |                              |
|-----------------------------|---------|-------------------|------------------------------|
| An individual abbreviation: | Sir     | $\Rightarrow$     | Tom Moran:EuroPARC:RX        |
| A group abbreviation:       | * (APU) | $\Rightarrow$     | *@MRC-APU.CAM.AC.UK:GV:Xerox |
| Hiding the local domain:    | *       | $\Leftarrow$      | *:EuroPARC:RX                |
| Hiding gateway syntax:      | *@*.EDU | $\Leftrightarrow$ | *%*:EDU:Xerox                |

**MODULE EXPLANATION**

LAFITEABBREV advises the Lafite NSMAIL routines that encode and decode string addresses to NSNAMES. It operates transparently and automatically with respect to the rest of the mail system while being controlled by the variables mentioned below. The most central of these is the list of abbreviations itself, LAFITE.ABBREVS:

LAFITE.ABBREVS

[Variable]

This variable contains all abbreviations and their corresponding real addresses. It is formed as a list of translations, where each translation is a list of (up to) four elements: the abbreviation, the full address, an optional direction, and an optional comment. Both the abbreviation and full address should be strings; the direction may be either of :IN, :OUT, :BOTH, :NONE, or left empty in which case it defaults to :BOTH. Finally, the comment, if present, should be in normal Interlisp format, ie. (\* text). It has no functional purpose and serves only as a memory aid. A full translation thus looks like:

*(abbreviation full-address direction comment)*

eg: ("Pete" "Piotr Kropotkin:RusskiPARK:ZeRoks" :OUT (\* ; "Old chap"))

which means that an address of *Pete* will be expanded to *Piotr Kropotkin:RusskiPARK:ZeRoks* whenever you send a message to him, but his full address will always be presented as such when receiving mail from him or whenever his address is mentioned in the header lines of a message you receive. For an explanation on translation directions, read more below.

The default list of translations is:

```
@ "%*":GV:Xerox :IN
@ "%*":GV:Xerox
@. * "%*":*Xerox :IN
*.pa ":PA:Xerox
```

This will present most ARPA Internet addresses in their proper form and also allow you to see Palo Alto Grapevine addresses as if you were on Grapevine. Note that the first rule above is necessary because some external GV addresses are being enclosed in double quotes by the GV/XNS Mail Gateway (for no apparent reason, one might add, since there exist no common conventions for parsing such constructs and it technically is malforming the address).

LAFITE.ABBREV.DIRECTIONS

[Variable]

This variable controls the overall behaviour of LAFITEABBREV. It can take either of the following values:

|       |                                                                    |
|-------|--------------------------------------------------------------------|
| :IN   | Translate full addresses to abbreviations when receiving messages. |
| :OUT  | Translate abbreviations to full addresses when sending messages.   |
| :BOTH | Both of the above.                                                 |
| :NONE | Neither of the above, ie. don't ever do any translations at all.   |

The setting of LAFITE.ABBREV.DIRECTIONS limits the scope of individual translation rules, so if you for example set LAFITE.ABBREV.DIRECTIONS to :OUT, no translations will ever be done on incoming messages, not even if a rule has an explicit direction of :IN or :BOTH.

The default value of LAFITE.ABBREV.DIRECTIONS is :BOTH.

LAFITE.ABBREV.MOVE.GAZE.RIGHT

[Variable]

A quite handy application of LAFITEABBREV is to present ARPA Internet (RFC-822) addresses in their proper form, ie. translating addresses like Jakobsson%Score.Stanford.EDU:Xerox to Jakobsson@Score.Stanford.EDU and vice versa. Unfortunately, since the pattern matching algorithm uses a strict left-to-right order, an address like user%host%foo.bar:EDU:Xerox would end up translated as user@host%foo.bar.EDU, which is not quite what was intended. However, if LAFITE.ABBREV.MOVE.GAZE.RIGHT is set to a non-NIL value, translations resulting in a percent sign coming anywhere after an atsign will be automatically changed so that the rightmost percent sign is substituted for an atsign instead. Don't worry too much if you didn't understand any of the above; just think of it as applied magic (aka a *kludge*).

The default value of LAFITE.ABBREV.MOVE.GAZE.RIGHT is T.

LAFITE.ABBREV.TRACE

[Variable]

Set this variable to T or a window of your choice (eg the PROMPTWINDOW) to trace all translations that are done for you. Reset to NIL to stop tracing.

The default value of LAFITE.ABBREV.TRACE is NIL.

(LAFITE.ABBREV ADDRESS DIRECTION)

[Function]

This is the function that does the actual translations; it is used automatically by LAFITEABBREV's advices, but mentioned here if you want to test it manually or use it elsewhere. The address should be the string to expand/abbreviate and the direction either of :IN or :OUT, indicating whether this address is about to be sent or received. The result is the translated address.

(LAFITE.ABBREV.MATCH PATTERN STRING TEMPLATE)

[Function]

This is a small, case insensitive pattern matching algorithm supporting multiple wildcards on both the pattern and the template side. Asterisks will match substrings of zero or more characters, anything else has to match literally for the comparison to succeed. The result is constructed using the template, if supplied. Some examples:

| PATTERN    | STRING      | TEMPLATE | RESULT      |
|------------|-------------|----------|-------------|
| "Foo"      | "FOO"       | NIL      | "FOO"       |
| "Foo"      | "FOO"       | "Bar"    | "Bar"       |
| "Foo*Bar"  | "FOOBAZBAR" | NIL      | "FOOBAZBAR" |
| "Foo*Bar"  | "FOOBAZBAR" | "1*2"    | "1BAZ2"     |
| "Foo*B*ar" | "FOOBAZBAR" | "1*2*3"  | "12AZB3"    |
| "Foo*B*ar" | "FOOBAZBA"  | "1*2*3"  | NIL         |

## BUGS

Only XNS is supported (ie. GV addresses aren't touched). Messages from you won't be presented as "To: <Recipients>" in the browser if you abbreviate your own name. No nice'n'cuddly user interface included.

---

---

## LAFITE-INDENT

---

---

By: sML (Lanning.pa@Xerox.com)

### INTERNAL

This document last edited December 3, 1987.

### INTRODUCTION

It is common practice in conversations conducted via electronic mail to include bits and pieces of previous messages in reply messages. The included text is indented to set it apart visually from the main body of the message. LAFITE-INDENT provides some support for this style of e-mail use.

### USING LAFITE-INDENT

LAFITE-INDENT adds the item "Indent" to the default TEdit menu. Selecting the "Indent" item indents the current selection. In addition, it inserts carriage-returns in the text to ensure that no line in the selection is too long. The particular method of indentation is controlled by global variables described below. The "Indent" menu item also contains a number of subitems described below.

Indent

[TEdit menu item]

This item will indent the current selection by prepending (the value of) \*TEDIT-INDENT-STRING\* to each line. To ensure that resulting text is properly formatted, lines will be broken to contain no more than \*TEDIT-INDENT-LINE-LENGTH\* characters. Existing carriage returns in the current selection may be replaced by spaces. For example the result of indenting the highlighted text below:

---

Date: 9 Sep 86 13:34:12  
Subject: Eat?  
To: JFrench  
From: Wu

Hi Jane!

I was wondering if you might be able to make it over to our house tonight for dinner. Carol's got a turkey that she wants to BBQ, Hank is coming over with a couple of bottle of his home brew beer, and Diane's got a salad fresh from her garden.

Even if you can't make it for dinner, you might want to stop by to visit. Remember Crazy Tom from Berkeley? He's in the area on a job interview and he'll be there.

Let me know soon...

---

Date: 9 Sep 86 13:34:12  
Subject: Eat?  
To: JFrench  
From: Wu

Hi Jane! I was wondering if you might be able to make it over to our house tonight for dinner. Carol's got a turkey that she wants to BBQ, Hank is coming over with a couple of bottle of his home brew beer, and Diane's got a salad fresh from her garden.

Even if you can't make it for dinner, you might want to stop by to visit. Remember Crazy Tom from Berkeley? He's in the area on a job interview and he'll be there.

Let me know soon...

---

### Example 1: Before and after "Indent"

#### Indent & keep lines

[TEdit menu item]

Like Indent, Indent & keep lines will indent the current selection. Unlike Indent, returns in the current selection are kept. This can be useful when indenting the headers of mail messages. For example the result of "Indent & keep lines" on the following text:

---

Date: 9 Sep 86 13:34:12  
Subject: Eat?  
To: JFrench  
From: Wu

Hi Jane!

---

Date: 9 Sep 86 13:34:12  
Subject: Eat?  
To: JFrench  
From: Wu

Hi Jane!

---

### Example 2: Before and after "Indent & keep lines"

If Indent had been used instead of "Indent & keep lines", the lines would have been run together, resulting in:

Date: 9 Sep 86 13:34:12 Subject: Eat? To: JFrench From: Wu

#### Set indent

[TEdit menu item]

The "Set indent" message lets you change the current indentation string. It will prompt you for the new value of \*TEDIT-INDENT-STRING\*.

#### Unindent

[TEdit menu item]

Unindent will remove all indentation from the current selection. In the process, it will replace all "extra" carriage returns by spaces. For example:

---

Date: 9 Sep 86 13:34:12  
Subject: Eat?  
To: JFrench  
From: Wu

Hi Jane! I was wondering if you might be able to make it over to our house tonight for dinner. Carol's got a turkey that she wants to BBQ, Hank is coming over with a couple of bottle of his home brew beer, and Diane's got a salad fresh from her garden.

Even if you can't make it for dinner, you might want to stop by to visit. Remember Crazy Tom from Berkeley? He's in the area on a job interview and he'll be there.

Let me know soon...

---

Date: 9 Sep 86 13:34:12  
 Subject: Eat?  
 To: JFrench  
 From: Wu

Hi Jane! I was wondering if you might be able to make it over to our house tonight for dinner. Carol's got a turkey that she wants to BBQ, Hank is coming over with a couple of bottle of his home brew beer, and Diane's got a salad fresh from her garden.

Even if you can't make it for dinner, you might want to stop by to visit. Remember Crazy Tom from Berkeley? He's in the area on a job interview and he'll be there.

Let me know soon...

---

### Example 3: Before and after "Unindent"

Open line

[TEdit menu item]

"Open line" will open a blank line at the current position in the textstream. Text following the current position maintain its distance from the left margin. For example:

---

Date: 9 Sep 86 13:51:20  
 Subject: Re: Eat?  
 To: Wu  
 From: JFrench

Hi Jane! I was wondering if you might be able to make it over to our house tonight for dinner. Carol's got a turkey that she wants to BBQ, Hank is coming over with a couple of bottle of his home brew beer, and Diane's got a salad fresh from her garden.

---

Date: 9 Sep 86 13:51:20  
 Subject: Re: Eat?  
 To: Wu  
 From: JFrench

Hi Jane! I was wondering if you might be able to make it over to our house tonight for dinner. Carol's got a turkey that she wants to BBQ, Hank is coming over with a couple of bottle of his home brew beer, and Diane's got a salad fresh from her garden.

---

### Example 4: Before and after "Open line"

Insert <RETURN>s

[TEdit menu item]

This will replace TEdit's line breaks with real carriage returns (in the current selection). This way you know what line breaks recipients of your message will see.

Break long lines

[TEdit menu item]

This item inserts <RETURN>s in the text, making sure that no line contains more than \*TEDIT-INDENT-LINE-LENGTH\* characters. Functionally, this is just like the "Indent" item, but without the indentation.

### VARIABLES

The following variables can be changed to tailor how text is indented. They are defined as INITVARS in LAFITE-INDENT, so they can be given different values before or after loading the LAFITE-INDENT.

**\*TEDIT-INDENT-STRING\***

[GlobalVar]

This is the string that is used to indent each line of text. The default value is (the value of) (ALLOCSTRING 4 " "); a string consisting of four spaces.

**\*TEDIT-INDENT-LINE-LENGTH\***

[GlobalVar]

This is the maximum number of characters on a line that the indentation will allow. Extra line breaks will be added to ensure that no line has more than \*TEDIT-INDENT-LINE-LENGTH\* characters. The default value is 72.

## **BUGS**

Unfortunately, these operations are not (always) UNDOable.



---

---

**LAFITETIMEDDELETE**

---

---

By: John Maxwell (Maxwell.pa)

REQUIRES: LAFITE

INTERNAL

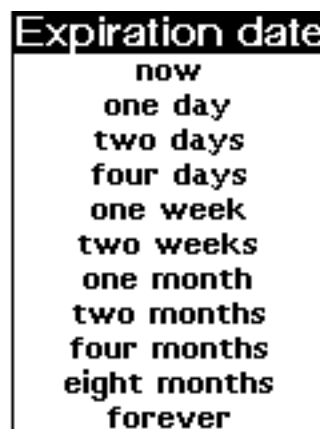
This document last edited on October 7, 1987.

## **INTRODUCTION**

The package LafiteTimedDelete allows users to specify expiration dates on their mail messages as they read them. After a message has expired, it will be marked for deletion the next time the user invokes the command "Delete Expired Msgs". The package is useful for specifying deletion dates on dated information such as announcements of talks. It can also be used as a "sunset" clause on certain messages, saying in effect that if a message hasn't been acted on by this date, mark it for deletion. Since the command for deleting expired messages only marks them for deletion and doesn't actually delete them, the user can always intervene and save the message from deletion.

## **SETTING EXPIRATION DATES**

To set an expiration date, simply select the message(s) that you wish to have deleted at some future time, and then click the "Delete" command on the Lafite browser with the middle button. This will produce a menu of durations like this:



Selecting a duration will determine how long until the message(s) expire, measured from the current day. If no selection is made, then the operation is aborted.

When a message is marked for future deletion, a number from 1 to 9 will appear in the mark field to the left of the message id. This number indicates how long until the message should expire, *measured from the date of the message*. The number is approximately the logarithm of the number of days until the message expires. Thus "1" means 1 day, "2" means 2 days, "3" means 4 days, "4" means 1 week, "5" means 2 weeks, "6" means 1 month, "7" means 2 months, "8" means 4 months, and "9" means 8 months.

Selecting the duration "now" is equivalent to normal deletion. Selecting the duration "forever" is equivalent to undeletion, with the side effect that all expiration marks are removed.

### **DELETING EXPIRED MESSAGES**

To delete all of the messages that have expired, invoke the "Delete Expired Msgs" command in the browser's middle button menu. (The middle button menu is obtained by holding down the middle button while over the black bar in the mail folder.) The program will then examine all of the messages in that folder, looking for messages that have expired. When it finds a message that has expired, it will mark it to be deleted. Finally, it will print in the browser's prompt window the number of messages that it marked for deletion.

Messages expire at 12 noon on the Nth day from the date given in the date field of browser. Thus if on Wednesday, October 7th you mark a message sent that day to be deleted in two days, then the message would be deletable after 12 noon on Friday, October 9th. However, the messages don't actually get marked for deletion until you invoke the command "Delete Expired Msgs". You usually only need to invoke this command once per day since no new messages will expire later on, so if you are unhappy with 12 noon as an expiration time, you can move it by only invoking the command after the day is over (or early in the morning the next day.)

### **CAVEATS**

Since LAFITETIMEDDELETE uses the mark field of the message header to save the information about how long until the message expires, you may run into conflicts with other uses of the mark field. For instance, if you set the expiration date on a message and then forward it, you will lose the expiration information because the forwarder puts an "f" in the mark field. If you notice this, you can restore the mark manually by clicking the mark field and typing a new number.

Also, since LAFITETIMEDDELETE is limited in the number of distinctions that it can make in the time until expiration, it may set the expiration of a message to a time much later than you might want. For instance, suppose that you are reading a message that was mailed yesterday and you want to set its expiration date to one week from today. Since it can only record expirations from the date of the message and not from the current date, it must set the message to expire in eight days. If the program used the one week expiration mark, then the message would expire in seven days, or six days from

today. Since that might expire the message before you intend, the program plays it safe and uses the two week expiration mark. A six day expiration might be OK with the user, but it would be awkward to ask him, especially if there were multiple messages selected. If the user is dissatisfied with the mark, he can always change it manually as described above.

Please send all comments, questions, and bug reports to Maxwell.pa.

# MAILSCAVENGE

## For Scavenging Mail Folders

The Lisp Library package MAILSCAVENGE is used to rebuild the internal pointers in a mail file that has been damaged. Lafite generally reports "Can't parse file" and terminates its Browse command when it detects damage in a file. The simplest remedy is to call MAILSCAVENGE, then browse the file again.

(MAILSCAVENGE *FILENAME* *ERRORMSGSTREAM* *TEMPDIR*)

[Function]

This function scavenges the file named *FILENAME*. *FILENAME* defaults to extension MAIL and your Lafite directory, exactly as with Lafite's Browse command. If *ERRORMSGSTREAM* is specified, it is a stream on which MAILSCAVENGE writes information about what it is doing to correct the file. *TEMPDIR* is a host/directory specification for where MAILSCAVENGE should write its intermediate file. *TEMPDIR* defaults to {DSK}, unless you are on an 1108 without a local disk file system, in which case it defaults to the same directory as *FILENAME*.

When MAILSCAVENGE finishes, it asks if you want to replace the original file with the scavenged file. If you reply "No", MAILSCAVENGE returns the name of the temporary file it wrote, which you can then rename or delete as you wish. Ordinarily, you should reply "Yes", unless you find the report of what MAILSCAVENGE had to correct to be suspicious.

(MAILSCAVENGE.IN.PLACE *FILENAME* *ERRORMSGSTREAM*)

[Function]

This is similar to MAILSCAVENGE, but destructively scavenges the file in-place. This is faster than MAILSCAVENGE, but you have to be brave and assume MAILSCAVENGE is not overwriting anything valuable as it scans the file.

MIME decoding package, by Ron Kaplan, 2/2000.

This file recognizes and decodes email messages that conform to the MIME internet standard. It interfaces to Lafite so that it can parse the MIME headers and recognize various kinds of attachments. The attachments are represented in the message display window as an image-object box. Clicking on that box will let you extract the attachment and write it to a file on a directory which is obtained by evaluating the value of ATTACHMENTDIR. This is initialized to

```
(CONCAT "{dsk}/tilde/" (L-CASE (USERNAME)) "/attachments")
```

which, for example, would evaluate to "{dsk}/tilde/kaplan/attachments".

The MIME package tries to show text attachments in-line, and it offers a Print option for attachments (or even just included text) that it recognizes as Postscript.

The extracted files can be accessed by standard PC software packages. For certain Mac-derived application types, it tries to provide the appropriate PC extension so that double clicking will launch the appropriate application.

---

---

**NSMAIL**

---

---

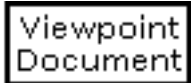
**INTRODUCTION**

The module NSMAIL implements the protocols to allow Lafite to be used to send and retrieve Xerox NS Mail. Load the file NSMAIL.LCOM. To run this in Lyric, you must have loaded the LispUsers module NSRANDOM as well (q.v. for important loading information). If you don't have NSRANDOM loaded, you can't use the "Put to file" command described below.

If you have both Grapevine and NSMAIL implementations loaded, you must set Lafite's mode to NS. Use the "NS Mode" subitem underneath Lafite's Quit command, or call (LAFITEMODE 'NS). You must also be a registered NS user, and have a mailbox.

**ATTACHMENTS**

The main difference between this and earlier versions of NSMAIL is that "attachments" are no longer left in your mailbox to be read later with, for example, Viewpoint. Instead, Lafite retrieves the entire attachment and encapsulates it into an image object that is enclosed as part of the text message, immediately following the header. A typical attachment appears in a mail message as:

Attachment: 

If you click inside the object with any mouse button, you are offered a menu of things you can do with the attachment. The choices vary according to the type of attachment:

- |              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| View as text | This brings up a window in which is displayed the raw content of the attachment as ascii bytes. Runs of non-ascii bytes are replaced by nulls to reduce the amount of garbage. Some attachments are utter gibberish, but some, such as Viewpoint documents and Interpress masters, contain sections that are plain text. With this command, you may be able to decide whether you care to do anything further with the attachment. (Sorry, there is no Viewpoint to TEdit converter, nor are there plans for one.) |
| Put to file  | This prompts you for a file name, and creates a file to contain the attachment. The file must be on an NS file server for this command to be very useful; otherwise, information will be lost. Once the file is so stored, you can retrieve it from Viewpoint and manipulate it just as if you had originally retrieved it as mail in Viewpoint.                                                                                                                                                                   |

|                 |                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Send to Printer | This command is only available for attachments that are in the form of an Interpress master. The command prompts you for a printer (must be one that accepts Interpress, of course), and sends the attachment to it for printing.                                                                                                                                                                   |
| Expand folder   | This command is only available for attachments that are in the form of a "folder". A folder is a mechanism for collecting several objects into a single one. The Expand folder command splits the attachment up into its component objects, each of which can be manipulated in the same way as a top-level attachment. For example, if the folder contains an Interpress master, you can print it. |

If you use the Put to file command on a folder, the name component of the file name you type will be treated as the name of a new subdirectory, and the components of the folder will appear as files in that subdirectory. For other types of attachments, Put to file (usually) produces an ordinary (non-directory) file.

Messages containing attachments are otherwise just like formatted messages—you can move them to other folders, and you can forward them (assuming the mail is received by another Lafite recipient and did not have to pass through a mail gateway).

There is currently no mechanism for creating your own attachments to send to other users.

## MISCELLANY

If you prefer the old behavior of leaving the attachments behind in the mailbox, set the variable NSMAIL.LEAVE.ATTACHMENTS to T, but this use is discouraged. You must take care to regularly retrieve your mail from somewhere (such as Viewpoint) that will flush out all the mail; otherwise, the mail with attachments, whether you want them or not, accumulate on the server.

When in NS mode, Lafite will want your NS login identity. Normally, if your NS password differs from your default password, you will be prompted to login. You can also call (LOGIN 'INS::) yourself to set your NS login.

You can freely intermix Grapevine and NS mail in the same mail folder if you like, but the Answer command always treats the selected message as if it were one in the current mode. So if you try to answer a Grapevine message while in NS mode, some confusion may result. Also, the status window always shows you mail status only of the current mode.

---

---

## UNIXMAIL

---

---

By: Bob Bane (Bane.mv@envos.Xerox.com)

### INTRODUCTION

UNIXMAIL is a new mail sending and receiving mode for Lafite. It sends mail via Unix hosts using the SMTP mail transfer protocol and can receive mail either by reading a Unix mail spool file or by calling the Berkeley mail program.

### INSTALLATION

Turn Lafite off, load the file UNIXMAIL, make sure UNIXMAIL is configured appropriately (check the settings of the variables below, and make sure any other modules UNIXMAIL may need are loaded), then restart Lafite. If you are running Lafite on a machine that is isolated from the Xerox mail environment, you will probably want to set the variable LAFITE.USE.ALL.MODES to NIL and call (LAFITEMODE 'UNIX) before you turn Lafite back on.

### CONFIGURING

See **SENDING MAIL** and **RECEIVING MAIL** below for the exact meanings of the variables you will be asked to set.

D-machines:

UNIXMAIL.SEND.MODE must be set to SOCKET and UNIXMAIL.SEND.HOST must be set to the name of a TCP host that will accept SMTP connections. UNIXMAIL.RECEIVE.MODE must be set to SPOOL and UNIXMAIL.SPOOL.FILE must be set to the pathname of your Unix mail spool file.

Unix-based emulators:

The default values of UNIXMAIL.SEND.MODE and UNIXMAIL.RECEIVE.MODE (PROCESS and SPOOL, respectively) will work if you normally send and receive mail from the machine where Medley is running.

### OTHER MODULES YOU MAY NEED

UNIXMAIL may need other library modules to work. The modules needed vary depending on what hardware you are using:

D-machines:

TCP is mandatory for Unix sending and may be used for Unix receiving, NFS is optional for Unix receiving

Unix-based emulators:

one of TCPOPS or UNIXCOMM is mandatory for sending

### SENDING MAIL

UNIXMAIL can send mail in one of two ways, depending on the setting of UNIXMAIL.SEND.MODE:

UNIXMAIL.SEND.MODE [Variable]

If its value is the atom PROCESS, UNIXMAIL will send mail by doing SMTP with a Unix process-stream, normally running /usr/etc/mconnect. This option only works on Medley running one of the Unix-based emulators.

If its value is the atom SOCKET, UNIXMAIL will send mail by doing SMTP with a TCP host. For this to work, an appropriate version of TCP must be loaded: either the TCP library module for D-machines or the TCPOPS library module for emulators that support it.



UNIXMAIL.SEND.MODE defaults to PROCESS.

Each of these send modes can be configured as well:

UNIXMAIL.SEND.PROCESS [Variable]

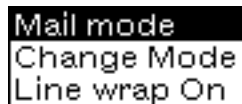
When UNIXMAIL.SEND.MODE is PROCESS, the value of this variable is the program run to create the SMTP process-stream. Initially the string `"/usr/etc/mconnect"`

UNIXMAIL.SEND.HOST [Variable]

When UNIXMAIL.SEND.MODE is SOCKET, the value of this variable is the name of the host UNIXMAIL will attempt to contact via TCP to open an SMTP stream over socket 25. Initially NIL; on a Unix-based emulator this means to try the machine Medley is running on. This variable must be set when running on a D-machine.

UNIXMAIL.WRAP.LINES [Variable]

This flag controls whether or not outgoing mail has its lines word-wrapped to a fixed length. It defaults to T, meaning word-wrapping is done. UNIXMAIL patches the Change Mode menu entry of the standard Lafite message form, adding an entry for toggling UNIXMAIL.WRAP.LINES:



UNIXMAIL.WRAP-LIMIT [Variable]

If UNIXMAIL.WRAP.LINES is true, this variable is the length in characters to which lines are wrapped. Default value is 72.

UNIXMAIL.TABWIDTH [Variable]

If UNIXMAIL.WRAP.LINES is true, this variable is the width tab characters are assumed to expand into for word-wrapping purposes. Default value is 8.

UNIXMAIL.RECIPIENT.PATTERNS [Variable]

This variable is a list of patterns that are applied to outgoing UNIXMAIL addresses; it can be used to catch bogus addresses and modify them before sending. List entries are of the form *(pattern . function)* where *pattern* is a list of arguments that will be passed along with the address to STRPOS; if STRPOS returns non-NIL, *function* is called with the address and the result replaces the address. For example, if mail from UUCP host `black-silicon` is arriving via path `mimsy!black-silicon`, but the address in its headers is missing the `mimsy`, this entry on UNIXMAIL.RECIPIENT.PATTERNS will add it back:

```
(("black-silicon!" NIL NIL T) LAMBDA (R) (CONCAT "mimsy!" R))
```

This means that whenever `(STRPOS "black-silicon!" address NIL NIL T)` returns non-NIL, *address* will have `"mimsy!"` prepended before the message is sent.

## RECEIVING MAIL

UNIXMAIL can receive mail in one of two ways, depending on the setting of UNIXMAIL.RECEIVE.MODE:

UNIXMAIL.RECEIVE.MODE [Variable]

If its value is the atom SPOOL, UNIXMAIL will receive mail by reading a Unix mail pool file.

If its value is the atom MAILER, UNIXMAIL will receive mail by running a Berkeley mailer as a Unix process-stream, normally `/usr/ucb/mail`. This option only works on Medley running one of the

Unix-based emulators, and is a bit slower than SPOOL mode; it is primarily useful when you wish to occasionally switch between Lafite and the Berkeley mailer.

UNIXMAIL.RECEIVE.MODE defaults to SPOOL.

Each of these receive modes can be configured as well:

UNIXMAIL.RECEIVE.PROCESS [Variable]

When UNIXMAIL.RECEIVE.MODE is MAILER, the value of this variable is the program run to create the SMTP process-stream. Initially the string `"/usr/ucb/mail -N"`; the `-N` means to not print any banner or read any initialization file on starting the mailer.

UNIXMAIL.DONT.RECEIVE.STATUS [Variable]

When UNIXMAIL.RECEIVE.MODE is MAILER, the value of this variable is a set of message status letters; UNIXMAIL will leave behind any message whose status is included. Initially `" "`, which means to read all messages regardless of status; another useful value would be `"O"` which means leave old messages behind.

UNIXMAIL.SPOOL.FILE [Variable]

When UNIXMAIL.RECEIVE.MODE is SPOOL, the value of this variable is the file UNIXMAIL will receive mail from. Any time this file has characters in it, Lafite will say you have new mail; when Lafite gets mail from this file, it will read all messages in the file and then set its size to zero. Initially `NIL`; on a Unix-based emulator this means to try the file `"{UNIX}/usr/spool/mail/username"`, where **username** is the value of (UNIX-USERNAME). To access a Unix mail spool file from a D-machine, it will probably be necessary to load and configure either the TCP or NFS modules and then set UNIXMAIL.SPOOL.FILE appropriately.

# MAIKOCOLOR

---



---

## Introduction

---

This module is the Envos Lisp software driver for running the COLOR module under Maiko (Maiko is the MACHINETYPE of Medley running on the Sun workstations). The machine independent functionality is provided by and documented with the COLOR module. There are no MAIKOCOLOR functions that the user needs to call directly. The user calls functions described in the COLOR documentation.

---

## Requirements

---

In order to run COLOR, you need a Sun 3 or Sun 4 with a color display (CG 4), plus a color emulator (lde). COLOR will \*not\* run on a non-color emulator, attempting to use color capabilities will cause an error trap into URaid. You additionally need:

COLOR  
LLCOLOR

---

## Installation

---

MAIKOCOLOR may be loaded into a sysout running on any D-Machine (or non-color emulator), as long as color is not initialized. Thus an initial sysout can be made which runs on all systems, by loading COLOR, and then writing the sysout.

To install, simply (FILESLOAD 'MAIKOCOLOR), which will load all the additional files necessary for running color.

---

## Initialization

---

To actually use color, the user must be running on a color capable emulator (lde).

To Initialize color running under Maiko, simply type

(COLORDISPLAY 'ON 'MAIKOCOLOR)

Lisp will permanently allocate a color "screen", and will attempt to map the color frame buffer to that screen.

At this point, the user should refer to the documentation for COLOR.

---

## Known Bugs

---

NOTE: As this is currently unreleased software undergoing active development, this list of bugs should not be construed as being a limitation or defect of the final product. This list is only included to point out the current state of affairs of the software.

1. Color screen is never GC'ed
2. Color TEdit slower than B / W TEdit
3. Can't create big color windows

# MASTERSCOPE

---

MasterScope is an interactive program for analyzing and cross referencing user programs. It contains facilities for analyzing user functions to determine what other functions are called; how and where variables are bound, set, or referenced; and which functions use particular record declarations. MasterScope can analyze definitions directly from a file as well as in-memory definitions.

MasterScope maintains a database of the results of the analyses it performs. Via a simple command language, you may interrogate the database, call the editor on those expressions in functions that were analyzed which use variables or functions in a particular way, or display the tree structure of function calls among any set of functions.

MasterScope is interfaced with the editor and file manager so that when a function is edited or a new definition loaded in, MasterScope knows that it must reanalyze that function.

With the Medley release, MasterScope now understands Common Lisp **defun**, **defmacro**, and **defvar**.

## Requirements

---

MSANALYZE, MSPARSE, MSCOMMON, MS-PACKAGE

You may also want to make use of Browser, DataBaseFns, and SEdit or DEdit.

## Installation

---

Load MASTERSCOPE.DFASL and the other .DFASL files from the library.

## MasterScope Command Language

---

You communicate with MasterScope using an English-like command language, e.g., WHO CALLS PRINT. With these commands, you can direct that functions be analyzed, interrogate the MasterScope database, and perform other operations. The commands deal with sets of functions, variables, etc., and relations between them (e.g., call, bind). Sets correspond to English nouns; relations correspond to verbs.

A set of atoms can be specified in a variety of ways, either explicitly, e.g., FUNCTIONS ON FIE specifies the atoms in (FILEFNLSLT 'FIE), or implicitly, e.g., NOT CALLING Y, where the meaning must be determined in the context of the rest of the command. Such sets of atoms are the basic building blocks with which the command language deals.

MasterScope also deals with relations between sets.

For example, the relation CALL relates functions and other functions; the relations BIND and USE FREELY relate functions and variables. These relations get stored in the MasterScope database when functions are analyzed. In addition, MasterScope "knows" about file manager conventions; CONTAIN relates files and various types of objects (functions, variables).

Sets and relations are used (along with a few additional words) to form sentence-like commands.

For example, the command `WHO ON 'FOO USE 'X FREELY` prints out the list of functions contained in the file `FOO` which use the variable `X` freely. The command `EDIT WHERE ANY CALLS 'ERROR` calls `EDITF` (see *IRM*) on those functions which have previously been analyzed that directly call `ERROR`, pointing at each successive expression where the call to `ERROR` actually occurs.

## MasterScope Commands

The normal mode of communication with MasterScope is via commands. These are sentences in the MasterScope command language which direct MasterScope to answer questions or perform various operations.

MasterScope commands are typed into the Executive window, preceded by a period (.) to distinguish them from other commands to the Exec. MasterScope keywords can be in any package, so MasterScope commands can be issued in any type of Exec. The commands may be typed uppercase or lowercase.

To use a keyword as a variable or function name, you must use a single quote in front of it, e.g., `.WHO SETS 'SETS`.

Note: Any MasterScope command may be followed by `OUTPUT FILENAME` to send output to the given file rather than the terminal, e.g., `.WHO CALLS WHO OUTPUT CROSSREF`.

`ANALYZE SET`

[MasterScope command]

Analyzes the functions in *SET* (and any functions called by them) and includes the information gathered in the database. MasterScope does not reanalyze a function if it thinks it already has valid information about that function in its database. You may use the command `REANALYZE` to force reanalysis.

Note that whenever a function is referred to in a command as a subject of one of the relations, it is automatically analyzed; you need not give an explicit `ANALYZE` command. Thus, `WHO IN MYFNS CALLS FIE` automatically analyzes the functions in `MYFNS` if they have not already been analyzed.

Note also that only `EXPR` definitions are analyzed; that is, MasterScope does not analyze compiled code. If necessary, the definition is `DWIMIFYed` before analysis. If there is no in-core definition for a function (either in the function definition cell or an `EXPR` property), MasterScope attempts to read in the definition from a file. Files which have been explicitly mentioned previously in some command are searched first. If the definition cannot be found on any of those files, MasterScope looks among the files on `FILELST` for a definition. If a function is found in this manner, MasterScope prints a message "(reading from *FILENAME*)". If no definition can be found at all, MasterScope prints a message "*FN* can't be analyzed". If the function previously was known, the message "*FN* disappeared!" is printed.

`REANALYZE SET`

[MasterScope command]

Causes MasterScope to reanalyze the functions in *SET* (and any functions called by them) even if it already has valid information in its database. This would be necessary if you had disabled or subverted the file manager; e.g., performed `PUTD`'s to change the definition of functions.

ERASE *SET* [MasterScope command]

Erases all information about the functions in *SET* from the database. ERASE by itself clears the entire database.

SHOW PATHS *PATHOPTIONS* [MasterScope command]

Displays a tree of function calls. This is described fully in "SHOW PATHS" below.

*SET RELATION SET* [MasterScope command]

*SET IS SET* [MasterScope command]

*SET ARE SET* [MasterScope command]

These commands have the same format as an English sentence with a subject (the first *SET*), a verb (*RELATION* or *IS* or *ARE*), and an object (the second *SET*). Any of the *SET*s within the command may be preceded by the question determiners *WHICH* or *WHO* (or just *WHO* alone).

For example, *WHICH FUNCTIONS CALL X* prints the list of functions that call the function *X*.

*RELATION* may be one of the relation words in present tense (*CALL*, *BIND*, *TEST*, *SMASH*, etc.) or used as a passive (e.g., *WHO IS CALLED BY WHO*). Other variants are allowed, e.g., *WHO DOES X CALL*, *IS FOO CALLED BY FIE*, etc.

The interpretation of the command depends on the number of question elements present:

If there is no question element, the command is treated as an assertion and MasterScope returns either *T* or *NIL*, depending on whether that assertion is true. Thus, *ANY IN MYFNS CALL HELP* prints *T* if any function in *MYFNS* call the function *HELP*, and *NIL* otherwise.

If there is one question element, MasterScope returns the list of items for which the assertion would be true.

For example,

```
MYFN BINDS WHO USED FREELY BY YOURFN
```

prints the list of variables bound by *MYFN* which are also used freely by *YOURFN*.

If there are two question elements, MasterScope prints a doubly indexed list:

```
_ . WHO CALLS WHO IN /FNS
RECORDSTATEMENT -- /RPLNODE
RECORDECL1 -- /NCONC, /RPLACD, /RPLNODE
RECREDECLARE1 -- /PUTHASH
UNCLISPTRAN -- /PUTHASH, /RPLNODE2
RECORDWORD -- /RPLACA
RECORD1 -- /RPLACA, /SETTOPVAL
EDITREC -- /SETTOPVAL
```

EDIT WHERE *SET RELATION SET* [- *EDITCOMS*] [MasterScope command]

(WHERE may be omitted.) The first *SET* refers to a set of functions. The *EDIT* command calls the editor on each expression where the *RELATION* actually occurs.

For example, `EDIT WHERE ANY CALL ERROR` calls `EDITF` on each (analyzed) function which calls `ERROR` stopping within a TTY: at each call to `ERROR`. Currently you cannot `EDIT WHERE` a file which `CONTAINS` a datum, nor where one function `CALLS` another `SOMEHOW`.

*EDITCOMS*, if given, is a list of commands passed to `EDITF` to be performed at each expression.

For example,

```
EDIT WHERE ANY CALLS MYFN DIRECTLY - (SW 2 3) P
```

switches the first and second arguments to `MYFN` in every call to `MYFN` and prints the result. `EDIT WHERE ANY ON MYFILE CALL ANY NOT @ GETD` calls the editor on any expression involving a call to an undefined function.

Note that `EDIT WHERE X SETS Y` points only at those expressions where `Y` is actually set, and skips over places where `Y` is otherwise mentioned.

`SHOW WHERE SET RELATION SET` [MasterScope command]

Like the `EDIT` command except merely prints out the expressions without calling the editor.

`EDIT SET [- EDITCOMS]` [MasterScope command]

Calls `EDITF` on each function in *SET*. *EDITCOMS*, if given, is passed as a list of editor commands to be Executed.

For example,

```
EDIT ANY CALLING FN1 - (R FN1 FN2)
```

replaces `FN1` by `FN2` in those functions that call `FN1`.

`DESCRIBE SET` [MasterScope command]

Prints the `BIND`, `USE FREELY` and `CALL` information about the functions in *SET*.

For example, the command `DESCRIBE PRINTARGS` might print out:

```
PRINTARGS [N, FLG]
 binds: TEM, LST, X
 calls: MSRECORDFILE, SPACES, PRIN1
 called by: PRINTSENTENCE, MSHELP, CHECKER
```

This shows that `PRINTARGS` has two arguments, `N` and `FLG`; binds internally the variables `TEM`, `LST` and `X`; calls `MSRECORDFILE`, `SPACES` and `PRIN1`; and is called by `PRINTSENTENCE`, `MSHELP`, and `CHECKER`.

You can specify additional information to be included in the description. `DESCRIBELST` is a list each of whose elements is a list containing a descriptive string and a form. The form is evaluated (it can refer to the name of the function being described by the free variable `FN`). If it returns a non-`NIL` value, the description string is printed followed by the value. If the value is a list, its elements are printed with commas between them.

For example, the entry

```
("types: " (GETRELATION FN ' (USE TYPE) T)
```



would include a listing of the types used by each function.

**CHECK SET**

[MasterScope command]

Checks for various anomalous conditions (mainly in the compiler declarations) for the files in *SET* (if *SET* is not given, *FILELST* is used).

For example, this command warns about:

- Variables which are bound but never referenced
- Functions in *BLOCKS* declarations which aren't on the file containing the declaration
- Functions declared as *ENTRIES* but not in the block
- Variables which may not need to be declared *SPECVARS* because they are not used freely below the places where they are bound

**FOR VARIABLE SET I.S.TAIL**

[MasterScope command]

This command provides a way of combining *CLISP* iterative statements with *MasterScope*. An iterative statement is constructed in which *VARIABLE* is iteratively assigned to each element of *SET*, and then the iterative statement tail *I.S.TAIL* is executed.

For example,

```
FOR X CALLED BY FOO WHEN CCODEP DO (PRINTOUT T X , , ,
(ARGLIST X) T)
```

prints out the name and argument list of all of the compiled functions which are called by *FOO*.

## MasterScope Relations

A relation is specified by one of the keywords below. Some of these "verbs" accept modifiers.

For example, *USE*, *SET*, *SMASH* and *REFERENCE* all may be modified by *FREELY*. The modifier may occur anywhere within the command. If there is more than one verb, any modifier between two verbs is assumed to modify the first one.

For example, in

```
USING ANY FREELY OR SETTING X,
```

*FREELY* modifies *USING* but not *SETTING*. The entire phrase is interpreted as the set of all functions which either use any variable freely or set the variable *X*, whether or not *X* is set freely. Verbs can occur in the present tense (e.g., *USE*, *CALLS*, *BINDS*, *USES*) or as present or past participles (e.g., *CALLING*, *BOUND*, *TESTED*). The relations (with their modifiers) recognized by *MasterScope* are:

**CALL**

[MasterScope relation]

Function *F1* calls *F2* if the definition of *F1* contains a form (*F2* --). The *CALL* relation also includes any instance where a function uses a name as a function, as in

```
(APPLY (QUOTE F2) --), (FUNCTION F2), etc.
```

(*CALL* and *CALLS* are equivalent.)

CALL SOMEHOW [MasterScope relation]

One function calls another SOMEHOW if there is some path from the first to the other. That is, if F1 calls F2, and F2 calls F3, then F1 CALLS F3 SOMEHOW.

This information is not stored directly in the database; instead, MasterScope stores only information about direct function calls, and (re)computes the CALL SOMEHOW relation as necessary.

USE [MasterScope relation]

If unmodified, the relation USE denotes variable usage in any way; it is the union of the relations SET, SMASH, TEST, and REFERENCE.

SET [MasterScope relation]

A function SETs a variable if the function contains a form

(SETQ var --), (SETQQ var --), etc.

SMASH [MasterScope relation]

A function SMASHes a variable if the function calls a destructive list operation (RPLACA, RPLACD, DREMOVE, SORT, etc.) on the value of that variable.

MasterScope also finds instances where the operation is performed on a part of the value of the variable. For example, if a function contains a form (RPLACA (NTH X 3) T), it is noted as SMASHing X.

If the function contains a sequence (SETQ Y X), (RPLACA Y T), then Y is noted as being SMASHed, but not X.

TEST [MasterScope relation]

A variable is TESTed by a function if its value is only distinguished between NIL and non-NIL.

For example, the form (COND ((AND X --) --)) tests the value of X.

REFERENCE [MasterScope relation]

This relation includes all variable usage except for SET.

Note: The verbs USE, SET, SMASH, TEST and REFERENCE may be modified by the words FREELY or LOCALLY. A variable is used FREELY if it is not bound in the function at the place of its use. It is used LOCALLY if the use occurs within a PROG or LAMBDA that binds the variable.

MasterScope also distinguishes between CALL DIRECTLY and CALL INDIRECTLY. A function is called directly if it occurs as CAR-of-form in a normal evaluation context. A function is called indirectly if its name appears in a context which does not imply its immediate evaluation, for example (SETQ Y (LIST (FUNCTION FOO) 3)). The distinction is whether or not the compiled code of the caller would contain a direct call to the callee.

Note that an occurrence of (FUNCTION FOO) as the functional argument to one of the built-in mapping functions which compile open is considered to be a direct call.

In addition, CALL FOR EFFECT (where the value of the function is not used) is distinguished from CALL FOR VALUE.

|                                                                                                                                                                                                                                           |                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| BIND                                                                                                                                                                                                                                      | [MasterScope relation] |
| The BIND relation between functions and variables includes both variables bound as function arguments and those bound in an internal PROG or LAMBDA expression.                                                                           |                        |
| USE AS A FIELD                                                                                                                                                                                                                            | [MasterScope relation] |
| MasterScope notes all uses of record field names within FETCH, REPLACE or CREATE expressions.                                                                                                                                             |                        |
| FETCH                                                                                                                                                                                                                                     | [MasterScope relation] |
| Use of a field within a FETCH expression.                                                                                                                                                                                                 |                        |
| REPLACE                                                                                                                                                                                                                                   | [MasterScope relation] |
| Use of a record field name within a REPLACE or CREATE expression.                                                                                                                                                                         |                        |
| USE AS A RECORD                                                                                                                                                                                                                           | [MasterScope relation] |
| MasterScope notes all uses of record names within CREATE or TYPE? expressions. Additionally, in (fetch (FOO FIE) of X), FOO is used as a record name.                                                                                     |                        |
| CREATE                                                                                                                                                                                                                                    | [MasterScope relation] |
| Use of a record name within a CREATE expression.                                                                                                                                                                                          |                        |
| USE AS A PROPERTY NAME                                                                                                                                                                                                                    | [MasterScope relation] |
| MasterScope notes the property names used in expressions such as GETPROP, PUTPROP, GETLIS, etc., if the name is quoted; e.g. if a function contains a form (GETPROP X (QUOTE INTERP)), then that function USEs INTERP as a property name. |                        |
| USE AS A CLISP WORD                                                                                                                                                                                                                       | [MasterScope relation] |
| MasterScope notes all iterative statement operators and user defined CLISP words as being used as a CLISP word.                                                                                                                           |                        |
| CONTAIN                                                                                                                                                                                                                                   | [MasterScope relation] |
| Files CONTAIN functions, records, and variables. This relation is not stored in the database but is computed using the file manager.                                                                                                      |                        |
| DECLARE AS LOCALVAR                                                                                                                                                                                                                       | [MasterScope relation] |
| DECLARE AS SPECVAR                                                                                                                                                                                                                        | [MasterScope relation] |
| MasterScope notes internal calls to DECLARE from within functions.                                                                                                                                                                        |                        |
| ACCEPT                                                                                                                                                                                                                                    | [MasterScope relation] |
| SPECIFY                                                                                                                                                                                                                                   | [MasterScope relation] |
| KEYCALL                                                                                                                                                                                                                                   | [MasterScope relation] |
| MasterScope notes keyword arguments of Common Lisp functions when they are analyzed and when they are called.                                                                                                                             |                        |

FOO ACCEPTS :BAR is true if FOO is a Common Lisp function that accepts the keyword :BAR. FOO ACCEPTS &ALLOW-OTHER-KEYS is true if FOO has &ACCEPT-OTHER-KEYS in its lambda list.

FOO SPECIFIES :BAR is true if FOO is a function that calls any function with the keyword :BAR; the function in question must ACCEPT :BAR.

FOO KEYCALLS BAR is true if FOO is a function and calls BAR with one or more keywords it ACCEPTS.

|              |                        |
|--------------|------------------------|
| FLET         | [MasterScope relation] |
| LABEL        | [MasterScope relation] |
| MACROLET     | [MasterScope relation] |
| LOCAL-DEFINE | [MasterScope relation] |

MasterScope tracks uses of Common Lisp local definition forms (it currently does not expand them while analyzing them, however).

FOO FLETS BAR is true if FOO is a function with a FLET defining BAR local to FOO.

LABELS and MACROLETS are similar. LOCAL-DECLARES is the union of FLETS, LABELS, and MACROLETS.

## Abbreviations

The following abbreviations are recognized:

FREE=FREELY  
 LOCAL=LOCALLY  
 PROP=PROPERTY  
 REF=REFERENCE

Also, the words A, AN and NAME (after AS) are "noise" words and may be omitted.

## MasterScope Templates

MasterScope uses templates (see "Effecting MasterScope Analysis" below) to decide which relations hold between functions and their arguments.

For example, the information that SORT SMASHes its first argument is contained in the template for SORT. MasterScope initially contains templates for most system functions which set variables, test their arguments, or perform destructive operations. You may change existing templates or insert new ones in MasterScope's tables via the SETTEMPLATE function (below).

MasterScope also constructs templates to handle Common Lisp functions with keyword arguments. These constructed templates are noticed by FILES? and can be saved if desired, or MasterScope can recreate them by analyzing the functions again.

## MasterScope Set Specifications

A set is a collection of things (functions, variables, etc.). A set is specified by a set phrase, consisting of a determiner (e.g., ANY, WHICH, WHO) followed by a type (e.g., FUNCTIONS, VARIABLES) followed by a specification (e.g., IN MYFNS). The determiner, type and specification may be used alone or in combination.

For example,

```
ANY FUNCTIONS IN MYFNS,
VARIABLES IN GLOBALVARS, and
WHO
```

are all acceptable set phrases.

Note: Sets may also be specified with relative clauses introduced by the word *THAT*, e.g. *THE FUNCTIONS THAT BIND 'X*.

*'ATOM* [MasterScope set specification]

The simplest way to specify a set consisting of a single thing is by the name of that thing.

For example, in the command *WHO CALLS 'ERROR*, the function *ERROR* is referred to by its name. Although the *'* (apostrophe) can be left out, to resolve possible ambiguities names should usually be quoted; e.g., *WHO CALLS 'CALLS* returns the list of functions which call the function *CALLS*.

*'LIST* [MasterScope set specification]

Sets consisting of several atoms may be specified by naming the atoms.

For example, the command *WHO USES ' (A B)* returns the list of functions that use the variables *A* or *B*.

*IN EXPRESSION* [MasterScope set specification]

The form *EXPRESSION* is evaluated, and its value is treated as a list of the elements of a set.

For example, *IN GLOBALVARS* specifies the list of variables in the value of the variable *GLOBALVARS*.

*@ PREDICATE* [MasterScope set specification]

A set may also be specified by giving a predicate which the elements of that set must satisfy. *PREDICATE* is either a function name, a *LAMBDA* expression, or an expression in terms of the variable *X*. The specification *@ PREDICATE* represents all atoms for which the value of *PREDICATE* is non-NIL.

For example, *@ EXPRP* specifies all those atoms which have *EXPR* definitions; *@ (STRPOS X CLISPCHARRAY)* specifies those atoms which contain *CLISP* characters. The universe to be searched is either determined by the context within the command (e.g., in *WHO IN FOOFNS CALLS ANY NOT @ GETD*, the predicate is only applied to functions which are called by any functions in the list *FOOFNS*), or in the extreme case, the universe defaults to the entire set of things which have been noticed by MasterScope, as in the command *WHO IS @ EXPRP*.

*LIKE ATOM* [MasterScope set specification]

*ATOM* may contain *ESCapes*; it is used as a pattern to be matched, as in the editor.

For example, *WHO LIKE /R\$ IS CALLED BY ANY* would find both */RPLACA* and */RPLNODE*.

(The ESC character prints out as a \$; it is a wildcard for any number of characters.)

FIELDS OF *SET*

[MasterScope set specification]

*SET* is a set of records. This denotes the field names of those records.

For example, the command WHO USES ANY FIELDS OF BRECORD returns the list of all functions which do a fetch or replace with any of the field names declared in the record declaration of BRECORD.

KNOWN [MasterScope set specification]

The set of all functions which have been analyzed.

For example, the command `WHO IS KNOWN` prints out the list of functions which have been analyzed.

THOSE [MasterScope set specification]

The set of things printed out by the last MasterScope question.

For example, following the command

`WHO IS USED FREELY BY PARSE`

you could ask `WHO BINDS THOSE` to find out where those variables are bound.

ON PATH *PATHOPTIONS* [MasterScope set specification]

Refers to the set of functions which would be printed by the command `SHOW PATHS PATHOPTIONS`.

For example,

`IS FOO BOUND BY ANY ON PATH TO 'PARSE`

tests whether `FOO` might be bound above the function `PARSE` (that is, whether `FOO` is bound in any function that is higher up in the calling tree than `PARSE` is) . `SHOW PATHS` is explained in detail below.

## Set Specifications by Relation

A set may also be specified by giving a relation its members must have with the members of another set:

*RELATIONING SET* [MasterScope set specification]

*RELATIONING* is used here generically to mean any of the relation words in the present participle form (possibly with a modifier), e.g., `USING`, `SETTING`, `CALLING`, `BINDING`. *RELATIONING SET* specifies the set of all objects which have that relation with some element of *SET*.

For example, `CALLING X` specifies the set of functions which call the function `X`; `USING ANY IN FOOVARS FREELY` specifies the set of functions which uses freely any variable in the value of `FOOVARS`.

*RELATIONED BY SET* [MasterScope set specification]

*RELATIONED IN SET* [MasterScope set specification]

This is similar to the *RELATIONING* construction.

For example, `CALLED BY ANY IN FOOFNS` represents the set of functions which are called by any element of `FOOFNS`; `USED FREELY BY ANY CALLING ERROR` is the set of variables which are used freely by any function which also calls the function `ERROR`.



## Set Specifications by Blocktypes

*BLOCKTYPE OF FUNCTIONS*

[MasterScope set specification]

*BLOCKTYPE ON FILES*

[MasterScope set specification]

These phrases allow you to ask about BLOCKS declarations on files (see *IRM*). *BLOCKTYPE* is one of *LOCALVARS*, *SPECVARS*, *GLOBALVARS*, *ENTRIES*, *BLKFNS*, *BLKAPPLYFNS*, or *RETFNS*.

*BLOCKTYPE OF FUNCTIONS* specifies the names which are declared to be *BLOCKTYPE* in any blocks declaration which contain any of *FUNCTIONS* (a "set" of functions). The "functions" in *FUNCTIONS* can either be block names or just functions in a block.

For example,

```
WHICH ENTRIES OF ANY CALLING 'Y BIND ANY GLOBALVARS ON
'FOO.
```

*BLOCKTYPE ON FILES* specifies all names which are declared to be *BLOCKTYPE* on any of the given *FILES* (a "set" of files).

## Set Determiners

Set phrases may be preceded by a determiner, which is one of the words *THE*, *ANY*, *WHO* or *WHICH*. The question determiners (*WHO* and *WHICH*) are meaningful in only some of the commands, namely those that take the form of questions. *ANY* and *WHO* (or *WHOM*) can be used alone; they are wild-card elements, e.g., the command *WHO USES ANY FREELY*, prints out the names of all (known) functions which use any variable freely. If the determiner is omitted, *ANY* is assumed; e.g., the command *WHO CALLS ' (PRINT PRIN1 PRIN2)* prints the list of functions which call any of *PRINT*, *PRIN1*, *PRIN2*. *THE* is also allowed, e.g., *WHO USES THE RECORD FIELD FIELDX*.

## Set Types

Any set phrase has a type; that is, a set may specify either functions, variables, files, record names, record field names or property names. The type may be determined by the context within the command (e.g., in *CALLED BY ANY ON FOO*, the set *ANY ON FOO* is interpreted as meaning the functions on *FOO* since only functions can be *CALLED*), or you may give the type explicitly (e.g., *FUNCTIONS ON FIE*).

The following types are recognized: *FUNCTIONS*, *VARIABLES*, *FILES*, *PROPERTY NAMES*, *RECORDS*, *FIELDS*, *I.S.OPRS*. Also, the abbreviations *FNS*, *VARs*, *PROPNAMEs* or the singular forms *FUNCTION*, *FN*, *VARIABLE*, *VAR*, *FILE*, *PROPNAME*, *RECORD*, *FIELD* are recognized.

Note that most of these types correspond to built-in file manager types (see *IRM*).

The type is used by MasterScope in a variety of ways when interpreting the set phrase:

1. Set types are used to disambiguate possible parsings.

For example, both commands

```
WHO SETS ANY BOUND IN X OR USED BY Y
```

```
WHO SETS ANY BOUND IN X OR CALLED BY Y
```

have the same general form. However, the first case is parsed as

WHO SETS ANY (BOUND BY X OR USED BY Y)

since both BOUND BY X and USED BY Y refer to variables; while the second case is parsed as

WHO SETS ANY BOUND IN (X OR CALLED BY Y) ,

since CALLED BY Y and X must refer to functions.

Note that parentheses may be used to group phrases.

2. The type is used to determine the modifier for USE:

FOO USES WHICH RECORDS is equivalent to

FOO USES WHO AS A RECORD FIELD .

3. The interpretation of CONTAIN depends on the type of its object: the command

WHAT FUNCTIONS ARE CONTAINED IN MYFILE

prints the list of functions in MYFILE.

WHAT RECORDS ARE ON MYFILE

prints the list of records.

4. The implicit universe in which a set expression is interpreted depends on the type:

ANY VARIABLES @ GETD

is interpreted as the set of all variables which have been noticed by MasterScope (i.e., bound or used in any function which has been analyzed) that also have a definition.

ANY FUNCTIONS @ (NEQ (GETTOPVAL X) 'NOBIND)

is interpreted as the set of all functions which have been noticed (either analyzed or called by a function which has been analyzed) that also have a top-level value.

## Conjunctions of Sets

Sets may be joined by the conjunctions AND and OR or preceded by NOT to form new sets. AND is always interpreted as meaning intersection; OR as union; NOT as complement.

For example, the set CALLING X AND NOT CALLED BY Y specifies the set of all functions which call the function X but are not called by Y.

Note: MasterScope's interpretation of AND and OR follow Lisp conventions rather than the conventional English interpretation.

"Calling X and Y" would, in English, be interpreted as the intersection of (CALLING X) and (CALLING Y); but MasterScope interprets CALLING X AND Y as CALLING ('X AND 'Y), which is the null set.

Only sets may be joined with conjunctions. Joining modifiers, as in

USING X AS A RECORD FIELD OR PROPERTY NAME

is not allowed; in this case, you must type

USING X AS A RECORD FIELD OR USING X AS A PROPERTY NAME

As described above, the type of set is used to disambiguate parsings. The algorithm used is to first try to match the type of the phrases being joined and then try to join with the longest preceding phrase.

In any case, you may group phrases with parentheses to specify the manner in which conjunctions should be parsed.

## SHOW PATHS

---

In trying to work with large programs, you can lose track of the hierarchy of functions. The MasterScope `SHOW PATHS` command aids you by providing a map showing the calling structure of a set of functions. `SHOW PATHS` prints out a tree structure showing which functions call which other functions.

Loading the Browser library module modifies the `SHOW PATHS` command so the command's output is displayed as an undirected graph.

The `SHOW PATHS` command takes the form: `SHOW PATHS` followed by some combination of the following path options:

`FROM SET` [MasterScope path option]

Display the function calls from the elements of `SET`.

`TO SET` [MasterScope path option]

Display the function calls leading to elements of `SET`. If `TO` is given before `FROM` (or no `FROM` is given), the tree is inverted and a message (inverted tree) is printed to warn you that if `FN1` appears after `FN2` it is because `FN1` is called by `FN2`.

Note: When both `FROM` and `TO` are given, the first one indicates a set of functions to be displayed, while the second restricts the paths to be traced; i.e., the command `SHOW PATHS FROM X TO Y` traces the elements of the set CALLED SOMEHOW BY X AND CALLING Y SOMEHOW.

If `TO` is not given, `TO KNOWN OR NOT @ GETD` is assumed; that is, only functions which have been analyzed or which are undefined will be included.

Note that MasterScope analyzes a function while printing out the tree if that function has not previously been seen and it currently has an `EXPR` definition. Thus, any function which can be analyzed will be displayed.

`AVOIDING SET` [MasterScope path option]

Do not display any function in `SET`. `AMONG` is recognized as a synonym for `AVOIDING NOT`.

For example, `SHOW PATHS TO ERROR AVOIDING ON FILE2` does not display (or trace) any function on `FILE2`.

`NOTRACE SET` [MasterScope path option]

Do not trace from any element of `SET`. `NOTRACE` differs from `AVOIDING` in that a function which is marked `NOTRACE` is printed, but the tree beyond it is not expanded. The functions in an `AVOIDING` set are not printed at all.

For example,

`SHOW PATHS FROM ANY ON FILE1 NOTRACE ON FILE2`

displays the tree of calls emanating from `FILE1`, but does not expand any function on `FILE2`.

`SEPARATE SET`

[MasterScope path option]

Give each element of *SET* a separate tree.

Note: FROM and TO only insure that the designated functions are displayed. SEPARATE can be used to guarantee that certain functions begin new tree structures. SEPARATE functions are displayed in the same manner as overflow lines; i.e., when one of the functions indicated by SEPARATE is found, it is printed followed by a forward reference (a lowercase letter in braces) and the tree for that function is then expanded below.

`LINELENGTH N`

[MasterScope path option]

Resets LINELENGTH to *N* before displaying the tree. The linelength is used to determine when a part of the tree should "overflow" and be expanded lower.

---

## Error Messages

When you give MasterScope a command, the command is first parsed, i.e. translated to an internal representation, and then the internal representation is interpreted.

If a command cannot be parsed, e.g. if you typed

```
SHOW WHERE CALLED BY X
```

MasterScope would reply

```
Sorry, I can't parse that!
```

and generate an error.

If the command is of the correct form but cannot be interpreted (e.g., the command EDIT WHERE ANY CONTAINS ANY) MasterScope prints the message

```
Sorry, that isn't implemented!
```

and generates an error.

If the command requires some functions having been analyzed (e.g., the command WHO CALLS X) and the database is empty, MasterScope prints the message

```
Sorry, no functions have been analyzed!
```

and generates an error.

---

## Macro Expansion

As part of analysis, MasterScope expands the macro definition of called functions if they are not otherwise defined (see *IRM*). MasterScope always expands Common Lisp DEFMACRO definitions (unless it finds a template for the macro).

MasterScope Interlisp macro expansion is controlled by a variable:

`MSMACROPROPS`

[Variable]

Value is an ordered list of macro-property names that MasterScope searches to find a macro definition. Only the kinds of macros that appear on MSMACROPROPS are expanded. All others are treated as function calls and left unexpanded. Initially (MACRO).

Note: MSMACROPROPS initially contains only MACRO (not 1OMACRO, DMACRO, etc.) on the assumption that the machine-dependent macro definitions are more likely "optimizers".

If you edit a macro, MasterScope knows to reanalyze the functions which call that macro.

Note: If your macro is of the "computed-macro" style, and it calls functions which you edit, MasterScope does not notice. You must be careful to tell masterscope to REANALYZE the appropriate functions (e.g., if you edit FOOEXPANDER which is used to expand FOO macros, you have to REANALYZE ANY CALLING FOO.

## Effecting MasterScope Analysis

---

MasterScope analyzes the EXPR definition of a function, and notes in its database the relations that this function has with other functions and with variables. To perform this analysis, MasterScope uses templates which describe the behavior of functions.

For example, the information that SORT destructively modifies its first argument is contained in the template for SORT. MasterScope initially contains templates for most system functions that set variables, test their arguments, or perform destructive operations.

A template is a list structure containing any of the following atoms:

PPE [in MasterScope template]

If an expression appears in this location, there is most likely a parenthesis error.

MasterScope notes this as a call to the function ppe (lowercase). Therefore, SHOW WHERE ANY CALLS ppe prints out all possible parenthesis errors. When MasterScope finds a possible parenthesis error in the course of analyzing a function definition, rather than printing the usual ".", it prints out a "?" instead. MasterScope notes functions called with keywords they do not accept as calls to ppe.

NIL [in MasterScope template]

The expression occurring at this location is not evaluated.

SET [in MasterScope template]

A variable appearing at this place is set.

SMASH [in MasterScope template]

The value of this expression is smashed.

TEST [in MasterScope template]

Is used as a predicate (that is, the only use of the value of the expression is whether it is NIL or non-NIL).

PROP [in MasterScope template]

Is used as a property name. If the value of this expression is of the form `(QUOTE ATOM)`, MasterScope notes that *ATOM* is USED AS A PROPERTY NAME.

For example, the template for GETPROP is `(EVAL PROP . PPE)`.

Must appear at the end of a template followed by the keywords the templated function accepts.

The expression at this point is used as a functional argument.

An atom at this location is a variable which is bound.



CALL [in MasterScope template]

An atom at this location is a function which is called.

CLISP [in MasterScope template]

An atom at this location is used as a CLISP word.

! [in MasterScope template]

This atom, which can only occur as the first element of a template, allows you to specify a template for the CAR of the function form. If ! doesn't appear, the CAR of the form is treated as if it had a CALL specified for it. In other words, the templates (.. EVAL) and (! CALL .. EVAL) are equivalent.

If the next atom after a ! is NIL, this specifies that the function name should not be remembered.

For example, the template for AND is (! NIL .. TEST RETURN), which means that if you see an AND, don't remember it as being called. This keeps the MasterScope database from being cluttered by too many uninteresting relations. MasterScope also throws away relations for COND, CAR, CDR, and a couple of others.

## Special Forms

In addition to the above atoms that occur in templates, there are some special forms which are lists keyed by their CAR.

.. *TEMPLATE* [in MasterScope template]

Any part of a template may be preceded by the atom .. (two periods) which specifies that the template should be repeated an indefinite number ( $N \geq 0$ ) of times to fill out the expression.

For example, the template for COND might be

(.. (TEST .. EFFECT RETURN) )

while the template for SELECTQ is

(EVAL .. (NIL .. EFFECT RETURN) RETURN) .

(Although MasterScope "throws away" the relations for COND, it makes sense to template COND because there may be important information within the arguments of COND.)

(BOTH *TEMPLATE1* *TEMPLATE2*) [in MasterScope template]

Analyze the current expression twice, using the each of the templates in turn.

(IF *EXPRESSION* *TEMPLATE* *TEMPLATE* ) [in MasterScope template]

Evaluate *EXPRESSION* at analysis time (the variable *EXPR* is bound to the expression which corresponds to the IF), and if the result is non-NIL, use *TEMPLATE1*, otherwise *TEMPLATE2*. If *EXPRESSION* is a literal atom, it is APPLYD to *EXPR*.

For example,

```
(IF LISTP (RECORD FETCH) FETCH)
```

specifies that if the current expression is a list, then the first element is a record name and the second element a field name, otherwise it is a field name.

(@ *EXPRFORM* *TEMPLATEFORM*) [in MasterScope template]

Evaluate *EXPRFORM* giving *EXPR*, evaluate *TEMPLATEFORM* giving *TEMPLATE*. Then analyze *EXPR* with *TEMPLATE*. @ lets you compute on the fly both a template and an expression to analyze with it. The forms can use the variable *EXPR*, which is bound to the current expression.

(*MACRO* . *MACRO*) [in MasterScope template]

*MACRO* is interpreted in the same way as macros (see *IRM*) and the resulting form is analyzed. If the template is the atom *MACRO* alone, MasterScope uses the *MACRO* property of the function itself. This is useful when analyzing code which contains calls to user-defined macros. If you change a macro property (e.g., by editing it) of an atom which has template of *MACRO*, MasterScope marks any function which used that macro as needing to be reanalyzed.

Some examples of templates:

| <u>Function</u> | <u>Template</u>                                                   |
|-----------------|-------------------------------------------------------------------|
| DREVERSE        | (SMASH . PPE)                                                     |
| AND             | (! NIL TEST .. RETURN)                                            |
| MAPCAR          | (EVAL FUNCTION FUNCTION)                                          |
| COND            | (! NIL .. (IF CDR (TEST .. EFFECT RETURN)<br>(TESTRETURN . PPE))) |

Templates may be changed and new templates defined using the following functions:

(GETTEMPLATE *FN*) [Function]

Returns the current template of *FN*.

(SETTEMPLATE *FN* *TEMPLATE*) [Function]

Changes the template for the function *FN* and returns the old value. If any functions in the database are marked as calling *FN*, they are marked as needing reanalysis.

---

## Updating the MasterScope Database

MasterScope is interfaced to the editor and file manager so that it notes whenever a function has been changed, either through editing or loading in a new definition. Whenever a command is given which requires knowing the information about a specific function, if that function has been noted as being changed, the function is automatically reanalyzed before the command is interpreted. If the command requires that all the information in the database be consistent (e.g., you ask `WHO CALLS X`) then all functions which have been marked as changed are reanalyzed.

## MasterScope Entries

(MASTERSCOPE *COMMAND*—) [Function]

Top level entry to MasterScope. If *COMMAND* is *NIL*, enters into an Executive in which you may enter commands. If *COMMAND* is not *NIL*, the command is interpreted and *MASTERSCOPE* returns the value that would be printed by the command.

Note that only the question commands return meaningful values.

(CALLS *FN USEDATABASE*—) [Function]

*FN* can be a function name, a definition, or a form.

Note: *CALLS* also works on compiled code. *CALLS* returns a list of four elements:

- Functions called by *FN*
- Variables bound in *FN*
- Variables used freely in *FN*
- Variables used globally in *FN*

For the purpose of *CALLS*, variables used freely which are on *GLOBALVARS* or have a property *GLOBALVAR* value *T* are considered to be used globally. If *USEDATABASE* is *NIL* (or *FN* is not a symbol), *CALLS* performs a one-time analysis of *FN*. Otherwise (i.e., if *USEDATABASE* is non-*NIL* and *FN* a function name), *CALLS* uses the information in MasterScope's database (*FN* is analyzed first if necessary).

(CALLSCCODE *FN* —) [Function]

The subfunction of *CALLS* which analyzes compiled code. *CALLSCCODE* returns a list of elements:

- Functions called via "linked" function calls (not implemented in Interlisp-D)
- Functions called regularly
- Variables bound in *FN*
- Variables used freely
- Variables used globally

(FREEVARS *FN USEDATABASE*) [Function]

Equivalent to (CADDR (CALLS *FN USEDATABASE*)). Returns the list of variables used freely within *FN*.

(SETSYNONYM *PHRASE MEANING*—) [Function]

Defines a new synonym for MasterScope's parser. Both *OLDPHRASE* and *NEWPHRASE* are words or lists of words; anywhere *OLDPHRASE* is seen in a command, *NEWPHRASE* is substituted.

For example,

```
(SETSYNONYM 'GLOBALS ' (VARS IN GLOBALVARS OR @(GETPROP
X 'GLOBALVAR)))
```

would allow you to refer with the single word GLOBALS to the set of variables which are either in GLOBALVARS or have a GLOBALVAR property.

## Functions for Writing Routines

The following functions are provided for users who wish to write their own routines using MasterScope's database:

(PARSERELATION *RELATION*) [Function]

*RELATION* is a relation phrase; e.g., (PARSERELATION ' (USE FREELY) ).  
PARSERELATION returns an internal representation for *RELATION*. For use in conjunction with GETRELATION.

(GETRELATION *ITEM RELATION INVERTED*) [Function]

*RELATION* is an internal representation as returned by PARSERELATION (if not, GETRELATION first performs (PARSERELATION *RELATION*) ).

*ITEM* is an atom. GETRELATION returns the list of all atoms which have the given relation to *ITEM*.

For example,

(GETRELATION 'X ' (USE FREELY) )

returns the list of variables that X uses freely.

If *INVERTED* is T, the inverse relation is used; e.g.

(GETRELATION 'X ' (USE FREELY) T)

returns the list of functions which use X freely.

If *ITEM* is NIL, GETRELATION returns the list of atoms which have *RELATION* with *any* other item; i.e., it answers the question WHO *RELATIONS* ANY.

Note that GETRELATION does not check to see if *ITEM* has been analyzed, or that other functions that have been changed have been reanalyzed.

(TESTRELATION *ITEM RELATION ITEM2 INVERTED*) [Function]

Is equivalent to (MEMB *ITEM2* (GETRELATION *ITEM RELATION INVERTED*) ); that is, it tests if *ITEM* and *ITEM2* are related via *RELATION*.

If *ITEM2* is NIL, the call is equivalent to

(NOT (NULL (GETRELATION *ITEM RELATION INVERTED*) ) )

i.e., TESTRELATION tests if *ITEM* has the given *RELATION* with any other item.

(MAPRELATION *RELATION MAPFN*) [Function]

Calls the function *MAPFN* on every pair of items related via *RELATION*. If (NARGS *MAPFN*) is 1, then *MAPFN* is called on every item which has the given *RELATION* to *any* other item.

(MSNEEDUNSAVE *FNS MSG MARKCHANGEFLG*) [Function]

Used to mark functions which depend on a changed record declaration (or macro, etc.), and which must be LOADED or UNSAVED (see below). *FNS* is a list of

functions to be marked, and *MSG* is a string describing the records, macros, etc. on which they depend. If *MARKCHANGEFLG* is non-NIL, each function in the list is marked as needing reanalysis.

(UPDATEFN *FN* *EVENIFVALID* —) [Function]

Equivalent to the command `ANALYZE 'FN`; that is, `UPDATEFN` analyzes *FN* if *FN* has not been analyzed before or if it has been changed since the time it was analyzed. If *EVENIFVALID* is non-NIL, `UPDATEFN` reanalyzes *FN* even if MasterScope thinks it has a valid analysis in the database.

(UPDATECHANGED) [Function]

Performs (UPDATEFN *FN*) on every function which has been marked as changed.

(MSMARKCHANGED *NAME* *TYPE* *REASON*) [Function]

Mark that *NAME* has been changed and needs to be reanalyzed. See *MARKASCHANGED* in the *IRM*.

(DUMPDATABASE *FNLST*) [Function]

Dumps the current MasterScope database on the current output file in a LOADable form. If *FNLST* is not NIL, `DUMPDATABASE` only dumps the information for the list of functions in *FNLST*. The variable `DATABASECOMS` is initialized to

```
((E (DUMPDATABASE)))
```

Thus, you may merely perform `(MAKEFILE 'DATABASE.EXTENSION)` to save the current MasterScope database. If a MasterScope database already exists when a `DATABASE` file is loaded, the database on the file is merged with the one in memory.

Note: Functions whose definitions are different from their definition when the database was made must be *REANALYZED* if their new definitions are to be noticed.

Note: The `DataBaseFns` library module provides a more convenient way of saving databases along with the source files to which they correspond.

## Noticing Changes that Require Recompiling

When a record declaration, iterative statement operator or macro is changed, and MasterScope has noticed a use of that declaration or macro (i.e., it is used by some function known about in the database), MasterScope alerts you about those functions which might need to be recompiled (e.g., they do not currently have *EXPR* definitions). Extra functions may be noticed.

For example, if *FOO* contains `(fetch (REC X) --)`, and some declaration other than *REC* which contains *X* is changed, MasterScope still thinks that *FOO* needs to be loaded/unsaved. The functions which need recompiling are added to the list `MSNEEDUNSAVE` and a message is printed out:

**The functions *FN1*, *FN2*, ... use macros which have changed.**

**Call `UNSAVEFNS()` to load and/or unsave them.**

In this situation, the following function is useful:

(UNSAVEFNS —) [Function]

Uses LOADFNS or UNSAVEDEF to make sure that all functions in the list MSNEEDUNSAVE have EXPR definitions, and then sets MSNEEDUNSAVE to NIL.

Note: If RECOMPILEDEFAULT (see *IRM*) is set to CHANGES, UNSAVEFNS prints out

**"WARNING: you must set RECOMPILEDEFAULT to EXPRS in order to have these functions recompiled automatically."**

---

## Implementation Notes

MasterScope keeps a database of the relations noticed when functions are analyzed. The relations are intersected to form primitive relationships such that there is little or no overlap of any of the primitives.

For example, the relation SET is stored as the union of SET LOCAL and SET FREE. The BIND relation is divided into BIND AS ARG, BIND AND NOT USE, and SET LOCAL, SMASH LOCAL, etc. Splitting the relations in this manner reduces the size of the database considerably, to the point where it is reasonable to maintain a MasterScope database for a large system of functions during a normal debugging session.

Each primitive relationship is stored in a pair of hash tables, one for the forward direction and one for the reverse.

For example, there are two hash tables, USE AS PROPERTY and USED AS PROPERTY. To retrieve the information from the database, MasterScope performs unions of the hash values.

For example, to answer FOO BINDS WHO, MasterScope looks in all of the tables which make up the BIND relation. The internal representation returned by PARSERELATION is a list of dotted pairs of hash tables. To perform GETRELATION requires only mapping down that list, doing GETHASHs on the appropriate hash tables and UNIONING the result.

Hash tables are used for a variety of reasons: storage space is smaller; it is not necessary to maintain separate lists of which functions have been analyzed (a special table, DOESN'T DO ANYTHING is maintained for functions which neither call other functions nor bind or use any variables); and accessing is relatively fast. Within any of the tables, if the hash value is a list of one atom, then the atom itself, rather than the list, is stored as the hash value. This also reduces the size of the database significantly.

---

## Example

### Sample Session

The following illustrates some of the MasterScope facilities.

```
50_. ANALYZE FUNCTIONS ON RECORD
.....
NIL
51_. WHO CALLS RECFIELDLOOK
```

```

(RECFIELDLOOK ACCESSDEF ACCESSDEF2 EDITREC)
52_. EDIT WHERE ANY CALL RECFIELDLOOK
RECFIELDLOOK :
(RECFIELDLOOK (CDR Y) FIELD)
tty:
5*OK
ACCESSDEF :
(RECFIELDLOOK DECLST FIELD VAR1)
6*OK
(RECFIELDLOOK USERRECLST FIELD)
7*N VAR1
8*OK
ACCESSDEF2 :
(RECFIELDLOOK (RECORD.SUBDECS TRAN) FIELD)
tty:
(RECFIELDLOOK (RECORD.SUBDECS TRAN) FIELD)
9*N (CAR TAIL]
10*OK
EDITREC :
(RECFIELDLOOK USERRECLST (CAR EDITRECX))
11*OK
NIL
53_. WHO CALLS ERROR
..
(EDITREC)
54_. SHOW PATHS TO RECFIELDLOOK FROM ACCESSDEF
(inverted tree)

1. RECFIELDLOOK RECFIELDLOOK
2. ACCESSDEF
3. ACCESSDEF2 ACCESSDEF2
4. ACCESSDEF
5. RECORDCHAIN
ACCESSDEF
NIL
55_. WHO CALLS WHO IN /FNS
RECORDSTATEMENT -- /RPLNODE
RECORDECL1 -- /NCONC, /RPLACD, /RPLNODE
RECREDECLARE1 -- /PUTHASH
UNCLISPTRAN -- /PUTHASH, /RPLNODE2
RECORDWORD -- /RPLACA
RECORD1 -- /RPLACA, /SETTOPVAL
EDITREC -- /SETTOPVAL

```

Event 50 You direct that the functions on file RECORD be analyzed. The leading period and space specify that this line is a MasterScope command. MasterScope prints a greeting and prompts with `_`. Within the top-level Executive of MasterScope, you may issue MasterScope commands, programmer's assistant commands, (e.g., REDO, FIX), or run programs. You can exit from the MasterScope Executive by

typing OK. The function "." is defined as a Nlambda NoSpread function which interprets its argument as a MasterScope command, Executes the command and returns.

MasterScope prints a "." whenever it (re)analyzes a function, to let you know what it is happening. The feedback when MasterScope analyzes a function is controlled by the flag MSPRINTFLG: if MSPRINTFLG is the atom ".", MasterScope prints out a period. (If an error in the function is detected, "?" is printed instead.) If MSPRINTFLG is a number *N*, MasterScope prints the name of the function it is analyzing every *N*th function. If MSPRINTFLG is NIL, MasterScope won't print anything. Initial setting is ".".

Note that the function name is printed when MasterScope starts analyzing, and the comma is printed when it finishes.

- Event 51 You ask which functions call RECFIELDLOOK. MasterScope responds with the list.
- Statement 52 You ask to edit the expressions where the function RECFIELDLOOK is called. MasterScope calls EDITF on the functions it had analyzed that call RECFIELDLOOK, directing the editor to the appropriate expressions. You then edit some of those expressions. In this example, the teletype editor is used. If DEdit is enabled as the primary editor, it would be called to edit the appropriate functions.
- Statement 53 Next you ask which functions call ERROR. Since some of the functions in the database have been changed, MasterScope reanalyzes the changed definitions (and prints out .'s for each function it analyzes). MasterScope responds that EDITREC is the only analyzed function that calls ERROR.
- Statement 54 You ask to see a map of the ways in which RECFIELDLOOK is called from ACCESSDEF. A tree structure of the calls is displayed.
- Statement 55 You then ask to see which functions call which functions in the list /FNS. MasterScope responds with a structured printout of these relations.

## SHOW PATHS

The command SHOW PATHS FROM MSPARSE prints out the structure of MasterScope's parser:

```
1.MSPARSE MSINIT MSMARKINVALID
2. | MSINITH MSINITH
3. MSINTERPRET MSRECORDFILE
4. | MSPRINTWORDS
5. | PARSECOMMAND GETNEXTWORD CHECKADV
6. | | PARSERELATION {a}
7. | | PARSESET {b}
8. | | PARSEOPTIONS {c}
9. | | MERGECONJ GETNEXTWORD {5}
```



```

10. | GETNEXTWORD {5}
11. | FIXUPTYPES SUBJTYPE
12. | | OBJTYPE
13. | FIXUPCONJUNCTIONS MERGECONJ {9}
14. | MATCHSCORE
15. MSPRINTSENTENCE

overflow - a
16.PARSERELATION GETNEXTWORD {5}
17. CHECKADV

overflow - b
19.PARSESET PARSESET
20. GETNEXTWORD {5}
21. PARSERELATION {6}
22. SUBPARSE GETNEXTWORD {5}

overflow - c
23.PARSEOPTIONS GETNEXTWORD {5}
24. PARSESET {19}

```

This example shows that the function MSPARSE calls MSINIT, MSINTERPRET, and MSPRINTSENTENCE. MSINTERPRET in turn calls MSRECORDFILE, MSPRINTWORDS, PARSECOMMAND, GETNEXTWORD, FIXUPTYPES, and FIXUPCONJUNCTIONS. The numbers in braces { } after a function name are backward references: they indicate that the tree for that function was expanded on a previous line. The lowercase letters in braces are forward references: they indicate that the tree for that function will be expanded below, since there is no more room on the line. The vertical bar is used to keep the output aligned.

[This page intentionally left blank]



# MATCH

---

Match provides a fairly general pattern match facility that allows you to specify certain tests that would otherwise be clumsy to write, by giving a pattern which the datum is supposed to match.

Essentially, you write "Does the (expression) X look like (the pattern) P?"

For example, `(MATCH X WITH (& 'A -- 'B))` asks whether the second element of X is an A, and the last element a B.

## Requirements

---

DWIM must be enabled.

## Installation

---

Load `MATCH.LCOM` from the library.

## Programmer's Interface

---

`(MATCH OBJECT WITH PATTERN)`

[CLISP operator]

Matches the *OBJECT* with the *PATTERN*.

The implementation of the matching is performed by computing (once) the equivalent Lisp expression which performs the indicated operation, and substituting this for the pattern (rather than by invoking each time a general purpose capability such as that found in the AI languages FLIP or PLANNER).

For example, the translation of

```
(MATCH X WITH (& 'A -- 'B)) is:
(AND (EQ (CADR X) 'A)
 (EQ (CAR (LAST (CDDR X))) 'B))
```

Thus the pattern match facility is really a pattern match compiler, and the emphasis in its design and implementation has been more on the efficiency of object code than on generality and sophistication of its matching capabilities. The goal was to provide a facility that could and would be used even where efficiency was paramount, e.g., in inner loops. Wherever possible, already existing Lisp functions are used in the translation, e.g., the translation of `($ 'A $)` uses `MEMB`, `($ ('A $) $)` uses `ASSOC`, etc.

The syntax for pattern match expressions is `(MATCH FORM WITH PATTERN)`, where *PATTERN* is a list as described below. If *FORM* appears more than once in the translation, and it is not either a variable or an expression that is easy to (re)compute, such as `(CAR Y)`, `(CDDR Z)`, etc., a dummy variable is generated and bound to the value of *FORM* so that *FORM* is not evaluated a multiple number of times.

For example, the translation of

```
(MATCH (FOO X) WITH ($ 'A $))
```

is simply

```
(MEMB 'A (FOO X)),
```

while the translation of

```
(MATCH (FOO X) WITH ('A 'B --)) is:
```

```
[PROG ($$2)
 (RETURN
 (AND (EQ (CAR (SETQ $$2 (FOO X))) 'A)
 (EQ (CADR $$2) 'B])
```

In the interests of efficiency, the pattern match compiler assumes that all lists end in NIL, i.e., there are no LISTP checks inserted in the translation to check tails.

For example, the translation of

```
(MATCH X WITH ('A & --))
```

is

```
(AND (EQ (CAR X) (QUOTE A)) (CDR X)),
```

which matches with (A B) as well as (A . B).

Similarly, the pattern match compiler does not insert LISTP checks on elements, e.g.,

```
(MATCH X WITH (('A --) --))
```

translates as

```
(EQ (CAAR X) 'A),
```

and

```
(MATCH X WITH (($1 $1 --) --))
```

translates as

```
(CDAR X)
```

Note that you can explicitly insert LISTP checks yourself by using @, as described below, e.g.,

```
(MATCH X WITH (($1 $1 --)@LISTP --))
```

translates as

```
(CDR (LISTP (CAR X)))
```

PATLISTPCHECK

[Variable]

The insertion of LISTP checks for *ELEMENTS* is controlled by the variable PATLISTPCHECK. When PATLISTPCHECK is T, LISTP checks are inserted, e.g.,

```
(MATCH X WITH (('A --) --))
```

translates as:

```
(EQ (CAR (LISTP (CAR (LISTP X)))) 'A)
```

PATLISTPCHECK is initially NIL. Its value can be changed within a particular function by using a local CLISP declaration (see *IRM*).

PATVARDEFAULT

[Variable]

Controls the treatment of !*ATOM* patterns (see below).

If PATVARDEFAULT is ' or QUOTE, !*ATOM* is treated the same as '*ATOM*.

If PATVARDEFAULT is = or EQUAL, same as =*ATOM*.

If PATVARDEFAULT is == or EQ, same as ==*ATOM*.

If PATVARDEFAULT is \_ or SETQ, same as *ATOM*\_*&*.

PATVARDEFAULT is initially ' (quote).

PATVARDEFAULT can be changed within a particular function by using a local CLISP declaration (see *IRM*).

Note: Numbers and strings are always interpreted as though PATVARDEFAULT were =, regardless of its setting. EQ, MEMB, and ASSOC are used for comparisons involving small integers.

Note: Pattern match expressions are translated using the DWIM and CLISP facilities, using all CLISP declarations in effect (standard/fast/undoable; see *IRM*).

## Pattern Elements

A pattern consists of a list of pattern elements. Each pattern element is said to match either an element of a data structure or a segment.

For example, in the TTY editor's pattern matcher (see *IRM*), "--" matches any arbitrary segment of a list, while & or a subpattern match only one element of a list. Those patterns which may match a segment of a list are called segment patterns; those that match a single element are called element patterns.

## Element Patterns

There are several types of element patterns, best given by their syntax:

|                                         |                                                                                                                                          |
|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| \$1 or &                                | Matches an arbitrary element of a list.                                                                                                  |
| ' <i>EXPRESSION</i>                     | Matches only an element which is equal to the given expression e.g., 'A, ' (A B).                                                        |
|                                         | EQ, MEMB, and ASSOC are automatically used in the translation when the quoted expression is atomic, otherwise EQUAL, MEMBER, and SASSOC. |
| = <i>FORM</i>                           | Matches only an element which is EQUAL to the value of <i>FORM</i> ; e.g., =X, = (REVERSE Y).                                            |
| == <i>FORM</i>                          | Same as =, but uses an EQ check instead of EQUAL.                                                                                        |
| <i>ATOM</i>                             | The treatment depends on setting of PATVARDEFAULT (see above).                                                                           |
| ( <i>PATTERN1</i> ... <i>PATTERNn</i> ) | Matches a list which matches the given patterns; e.g.,                                                                                   |

(& &), (-- 'A).

*ELEMENT-PATTERN@FN* Matches an element if *ELEMENT-PATTERN* matches it, and *FN* (name of a function or a LAMBDA expression) applied to that element returns non-NIL.

For example, &@NUMBERP matches a number, and ('A -- )@FOO matches a list whose first element is A and for which FOO applied to that list is non-NIL.

For simple tests, the function-object is applied before a match is attempted with the pattern, e.g.,

((-- 'A -- )@LISTP --)

translates as

(AND (LISTP (CAR X)) (MEMB 'A (CAR X))),

not the other way around. *FN* may also be a *FORM* in terms of the variable @, e.g., &@(EQ @ 3) is equivalent to =3.

- \* Matches any arbitrary element. If the entire match succeeds, the element which matched the \* is returned as the value of the match.

Note: Normally, the pattern match compiler constructs an expression whose value is guaranteed to be non-NIL if the match succeeds and NIL if it fails.

However, if a \* appears in the pattern, the expression generated could also return NIL if the match succeeds and \* was matched to NIL.

For example,

(MATCH X WITH ('A \* --))

translates as

(AND (EQ (CAR X) 'A) (CADR X)),

so if X is equal to (A NIL B) then (MATCH X WITH ('A \* --)) returns NIL even though the match succeeded.

*~ELEMENT-PATTERN* Matches an element if the element is not (~) matched by *ELEMENT-PATTERN*, e.g., ~'A, ~=X, ~(-- 'A --).

(\*ANY\* *ELEMENT-PATTERN ELEMENT-PATTERN* ...)

Matches if any of the contained patterns match.

## Segment Patterns

\$ or -- Matches any segment of a list (including one of zero length).

The difference between \$ and -- is in the type of search they generate.

For example,

(MATCH X WITH (\$ 'A 'B \$))

translates as

```
(EQ (CADR (MEMB 'A X)) 'B)
```



whereas

```
(MATCH X WITH (-- 'A 'B $))
```

translates as:

```
[SOME X (FUNCTION (LAMBDA ($$2 $$1)
 (AND (EQ $$2 'A)
 (EQ (CADR $$1) 'B])
```

Thus, a paraphrase of (`$ 'A 'B $`) would be "Is B the element following the first A?", whereas a paraphrase of (`-- 'A 'B $`) would be "Is there any A immediately followed by a B?"

Note that the pattern using `$` results in a more efficient search than that using `--`. However, (`$ 'A 'B $`) does not match with (`X Y Z A M O A B C`), but (`-- 'A 'B $`) does.

Essentially, once a pattern following a `$` matches, the `$` never resumes searching, whereas `--` produces a translation that always continues searching until there is no possibility of success. However, if the pattern match compiler can deduce from the pattern that continuing a search after a particular failure cannot possibly succeed, then the translations for both `--` and `$` is the same.

For example, both

```
(MATCH X WITH ($ 'A $3 $))
```

and

```
(MATCH X WITH (-- 'A $3 --))
```

translate as

```
(CDDDR (MEMB (QUOTE A) X))
```

because if there are not three elements following the first A, there certainly will not be three elements following subsequent A's, so there is no reason to continue searching, even for `--`.

Similarly, (`$ 'A $ 'B $`) and (`-- 'A -- 'B --`) are equivalent.

`$2, $3, etc.` Matches a segment of the given length.

Note that `$1` is not a segment pattern.

`!ELEMENT-PATTERN` Matches any segment which *ELEMENT-PATTERN* would match as a list.

For example, if the value of `FOO` is (`A B C`), `!=FOO` matches the segment `... A B C ...` etc.

Note: Since `!` appearing in front of the last pattern specifies a match with some tail of the given expression, it also makes sense in this case for a `!` to appear in front of a pattern that can only match with an atom, e.g., (`$2 ! 'A`) means match if `CDDR` of the expression is the atom `A`.

Similarly,

```
(MATCH X WITH ($! 'A))
```

translates to

```
(EQ (CDR (LAST X)) 'A) .
```

**!ATOM** The treatment depends on setting of PATVARDEFAULT.

If PATVARDEFAULT is ' or QUOTE, same as !'ATOM (see above discussion).

If PATVARDEFAULT is = or EQUAL, same as !=ATOM.

If PATVARDEFAULT is == or EQ, same as !=ATOM.

If PATVARDEFAULT is \_ or SETQ, same as ATOM\_\$.

. The atom "." is treated *exactly* like "!". In addition, if a pattern ends in an atom, the "." is first changed to "!", e.g., (\$1 . A) and (\$1 ! A) are equivalent, even though the atom "." does not explicitly appear in the pattern.

One exception where "." is not treated like "!" is when "." preceding an assignment does not have the special interpretation that "!" has preceding an assignment (see below).

For example,

```
(MATCH X WITH ('A . FOO_ 'B))
```

translates as:

```
(AND (EQ (CAR X) 'A)
 (EQ (CDR X) 'B)
 (SETQ FOO (CDR X)))
```

but

```
(MATCH X WITH ('A ! FOO_ 'B))
```

translates as:

```
(AND (EQ (CAR X) 'A)
 (NULL (CDDR X))
 (EQ (CADR X) 'B)
 (SETQ FOO (CDR X)))
```

### SEGMENT-PATTERN@FUNCTION-OBJECT

Matches a segment if the segment-pattern matches it, and the function object applied to the corresponding segment (as a list) returns non-NIL.

For example, (\$@CDDR 'D \$) matches (A B C D E) but not (A B D E), since CDDR of (A B) is NIL.

**Note:** An @ pattern applied to a segment requires computing the corresponding structure (with LDIFF) each time the predicate is applied (except when the segment in question is a tail of the list being matched).

## Assignments

Any pattern element may be preceded by "*VARIABLE\_*", meaning that if the match succeeds (i.e., everything matches), *VARIABLE* is set to the thing that matches that pattern element.

For example, if *X* is (A B C D E), (MATCH X WITH (\$2 Y\_\$3)) sets *Y* to (C D E).

Note that assignments are not performed until the entire match has succeeded, so assignments cannot be used to specify a search for an element found earlier in the match. For example, (MATCH X WITH (Y\_\$1 =Y --)) does not match with (A A B C . . .), unless, of course, the value of *Y* was A before the match started. This type of match is achieved by using place-markers, described below.

If the variable is preceded by a *!*, the assignment is to the tail of the list as of that point in the pattern, i.e., that portion of the list matched by the remainder of the pattern.

For example, if *X* is (A B C D E), (MATCH X WITH (\$ !Y\_'C 'D \$)) sets *Y* to (C D E), i.e., CDDR of *X*. In other words, when *!* precedes an assignment, it acts as a modifier to the *\_*, and has no effect whatsoever on the pattern itself, e.g., (MATCH X WITH ('A 'B)) and (MATCH X WITH ('A !FOO\_'B)) match identically, and in the latter case, *FOO* is set to CDR of *X*.

Note: *\*\_PATTERN-ELEMENT* and *!\*\_PATTERN-ELEMENT* are acceptable, e.g.,

```
(MATCH X WITH ($ 'A *_('B --) --))
```

translates as:

```
[PROG ($$2) (RETURN
 (AND (EQ (CAADR (SETQ $$2 (MEMB 'A X))) 'B)
 (CADR $$2])
```

## Place Markers

Variables of the form *#N*, where *N* is a number, are called place markers, and are interpreted specially by the pattern match compiler. Place markers are used in a pattern to mark or refer to a particular pattern element. Functionally, they are used like ordinary variables, i.e., they can be assigned values, or used freely in forms appearing in the pattern.

For example,

```
(MATCH X WITH (#1_$1 =(ADD1 #1)))
```

matches the list (2 3).

However, they are not really variables in the sense that they are not bound, nor can a function called from within the pattern expect to be able to obtain their values. For convenience, regardless of the setting of *PATVARDEFAULT*, the first appearance of a defaulted place-marker is interpreted as though *PATVARDEFAULT* were *\_*.

Thus the above pattern could have been written as

```
(MATCH X WITH (1 =(ADD1 1))) .
```

Subsequent appearances of a place-marker are interpreted as though *PATVARDEFAULT* were *=*.

For example,

```
(MATCH X WITH (#1 #1 --))
```

is equivalent to

```
(MATCH X WITH (#1_$1 =#1 --))
```

and translates as

```
(AND (CDR X) (EQUAL (CAR X) (CADR X)))
```

Note that `(EQUAL (CAR X) (CADR X))` would incorrectly match with `(NIL)`.

## Replacements

The construct *PATTERN-ELEMENT\_FORM* specifies that if the match succeeds, the part of the data that matched is to be replaced with the value of *FORM*.

For example, if `X = (A B C D E)`, `(MATCH X WITH ($ 'C $1_Y $1))` replaces the third element of `X` with the value of `Y`. As with assignments, replacements are not performed until after it is determined that the entire match is successful.

Replacements involving segments splice the corresponding structure into the list being matched, e.g., if `X` is `(A B C D E F)` and `FOO` is `(1 2 3)`, after the pattern `('A $ _FOO 'D $)` is matched with `X`, `X` is `(A 1 2 3 D E F)`, and `FOO` is EQ to CDR of `X`, i.e., `(1 2 3 D E F)`.

Note that `($ FOO_FIE $)` is ambiguous, since it is not clear whether `FOO` or `FIE` is the pattern element, i.e., whether `_` specifies assignment or replacement.

For example, if `PATVARDEFAULT` is `=`, this pattern can be interpreted as `($ FOO_=FIE $)`, meaning search for the value of `FIE`, and if found set `FOO` to it, or `($ =FOO_FIE $)` meaning search for the value of `FOO`, and if found, store the value of `FIE` into the corresponding position. In such cases, you should disambiguate by not using the `PATVARDEFAULT` option, i.e., by specifying `'` or `=`.

Note: Replacements are normally done with `RPLACA` or `RPLACD`. You can specify that `/RPLACA` and `/RPLACD` should be used, or `FRPLACA` and `FRPLACD`, by means of `CLISP` declarations (see *IRM*).

## Reconstruction

You can specify a value for a pattern match operation other than what is returned by the match by writing `(MATCH FORM1 WITH PATTERN => FORM2)`.

For example,

```
(MATCH X WITH (FOO_$ 'A --) => (REVERSE FOO))
```

translates as:

```
[PROG ($$2)
 (RETURN
 (COND ((SETQ $$2 (MEMB 'A X))
 (SETQ FOO (LDIFF X $2))
 (REVERSE FOO))
```

Place markers in the pattern can be referred to from within *FORM*, e.g., the above could also have been written as

```
(MATCH X WITH (!#1 'A --) => (REVERSE #1)).
```

If `->` is used in place of `=>`, the expression being matched is also physically changed to the value of *FORM*.

For example,

```
(MATCH X WITH (#1 'A !#2) -> (CONS #1 #2))
```

would remove the second element from *X*, if it were equal to *A*.

In general, `(MATCH FORM1 WITH PATTERN -> FORM2)` is translated so as to compute *FORM2* if the match is successful, and then smash its value into the first node of *FORM1*. However, whenever possible, the translation does not actually require *FORM2* to be computed in its entirety, but instead the pattern match compiler uses *FORM2* as an indication of what should be done to *FORM1*.

For example,

```
(MATCH X WITH (#1 'A !#2) -> (CONS #1 #2))
```

translates as

```
(AND (EQ (CADR X) 'A) (RPLACD X (CDDR X)))
```

---

## Limitation

The pattern match facility does not contain some of the more esoteric features of other pattern match languages, such as repeated patterns, disjunctive and conjunctive patterns, recursion, etc. However, you can be confident that what facilities it does provide results in Lisp expressions comparable to those you would generate by hand.

---

## Examples

```
(MATCH X WITH (-- 'A --))
```

`--` matches any arbitrary segment. `'A` matches only an `A`, and the second `--` again matches an arbitrary segment; thus this translates to `(MEMB 'A X)`.

```
(MATCH X WITH (-- 'A))
```

Again, `--` matches an arbitrary segment; however, since there is no `--` after the `'A`, *A* must be the last element of *X*. Thus this translates to: `(EQ (CAR (LAST X)) 'A)`.

```
(MATCH X WITH ('A 'B -- 'C $3 --))
```

*CAR* of *X* must be *A*, and *CADR* must be *B*, and there must be at least three elements after the first *C*, so the translation is:

```
(AND (EQ (CAR X) 'A)
 (EQ (CADR X) 'B)
 (CDDDR (MEMB 'C (CDDR X))))
```

```
(MATCH X WITH (('A 'B) 'C Y_$1 $))
```

Since ('A 'B) does not end in \$ or --, (CDDAR X) must be NIL. The translation is:

```
(COND
 ((AND (EQ (CAAR X) 'A)
 (EQ (CADAR X) 'B)
 (NULL (CDDAR X))
 (EQ (CADR X) 'C)
 (CDDR X))
 (SETQ Y (CADDR X)) T))
```

```
(MATCH X WITH (#1 'A $ 'B 'C #1 $))
```

#1 is implicitly assigned to the first element in the list. The \$ searches for the first B following A. This B must be followed by a C, and the C by an expression equal to the first element. The translation is:

```
[PROG ($$2)
 (RETURN
 (AND (EQ (CADR X) 'A)
 (EQ [CADR (SETQ $$2 (MEMB 'B (CDDR X)) 'C)
 (CDDR $$2)
 (EQUAL (CADDR $$2) (CAR X])
```

```
(MATCH X WITH (#1 'A -- 'B 'C #1 $))
```

Similar to the pattern above, except that -- specifies a search for *any* B followed by a C followed by the first element, so the translation is:

```
[AND (EQ (CADR X) 'A)
 (SOME (CDDR X)
 (FUNCTION (LAMBDA ($$2 $$1)
 (AND (EQ $$2 'B)
 (EQ (CADR $$1) 'C)
 (CDDR $$1)
 (EQUAL (CADDR $$1) (CAR X])
```

[This page intentionally left blank]

# MATMULT

---

Two dimensional graphical transformations, such as rotations, scalings, and translations are conveniently represented as homogeneous 3-by-3 matrices, which operate on homogeneous 3-vectors. Similarly, three dimensional graphical transformations are conveniently represented as homogeneous 4-by-4 matrices, which operate on homogeneous 4-vectors. MatMult provides utilities for creating and manipulating such matrices and vectors, and takes advantage of microcode support for high-speed 3-by-3 and 4-by-4 matrix multiplication.

All matrices and vectors in MatMult are represented as Common Lisp arrays of element type single-float, so the Common Lisp array functions are sufficient to create and access individual elements of these specialized arrays. However, MatMult provides convenient wrapper functions for most common operations on these arrays.

All the following functions that return arrays accept optional array arguments. If given a result argument, these functions alter the contents of that argument rather than allocating new storage. It is an error for the optional array argument to be not of element type single-float, or to have incorrect dimensions.

## Requirements

---

MatMult should be run on an 1109 with a Weitek floating point chip set, but is also quite efficient on an 1186.

## Installation

---

Load MATMULT.LCOM from the library.

## Matrix Creation Functions

---

(MAKE-HOMOGENEOUS-3-VECTOR *X Y*) [Function]

Returns a 3-vector of element type single-float. If *X* or *Y* is provided, then the corresponding element of the vector is set appropriately, otherwise it defaults to 0.0. The third element of the vector is always initialized to 1.0.

Note: Throughout this text, "set" is used to emphasize that the value of the result element is altered and that no new storage is allocated to it.

(MAKE-HOMOGENEOUS-3-BY-3 &KEY *A00 A01 A10 A20 A21*) [Function]

Returns a 3-by-3 matrix of element type single-float. If a keyword argument is provided, the corresponding element of the matrix is set appropriately, otherwise entries default to 0.0. The (2,2) is always initialized to 1.0.

(MAKE-HOMOGENEOUS-N-BY-3 *N* &KEY *INITIAL-ELEMENT*) [Function]

Returns an N-by-3 matrix of element type single-float. If the keyword argument is provided, all the elements in the first two columns are set appropriately, otherwise they default to 0.0. The third column is always initialized to 1.0.



(MAKE-HOMOGENEOUS-4-VECTOR *X Y Z*) [Function]

Returns a 4-vector of element type single-float. If *X*, *Y* or *Z* is provided then the corresponding element of the vector is set appropriately, otherwise it defaults to 0.0. The forth element of the vector is always initialized to 1.0.

(MAKE-HOMOGENEOUS-4-BY-4 &KEY *A00 A01 A02 A03 A10 A11 A12 A13 A20 A21 A22 A23 A30 A31 A32*) [Function]

Returns a 4-by-4 matrix of element type single-float. If a keyword arguments is provided, the corresponding element of the matrix is set appropriately, otherwise entries default to 0.0. The (3,3) is always initialized to 1.0.

(MAKE-HOMOGENEOUS-N-BY-4 *N* &KEY *INITIAL-ELEMENT*) [Function]

Returns an *N*-by-4 matrix of element type single-float. If the keyword argument is provided, all the elements in the first three columns are set appropriately, otherwise they default to 0.0. The forth column is always initialized to 1.0.

(IDENTITY-3-BY-3 *RESULT*) [Function]

Returns a 3-by-3 identity matrix.

If *RESULT* is supplied, it is side effected and returned.

(That is, the storage associated with the optional result argument is reused for the result, rather than allocating new storage for the result.)

(IDENTITY-4-BY-4 *RESULT*) [Function]

Returns a 4-by-4 identity matrix. If *RESULT* is supplied, it is side effected and returned.

(ROTATE-3-BY-3 *RADIANS RESULT*) [Function]

Returns a 3-by-3 rotation matrix specified by a counter-clockwise rotation of *RADIANS* radians. If *RESULT* is supplied, it is set and returned.

(ROTATE-4-BY-4-ABOUT-X *RADIANS RESULT*) [Function]

Returns a 4-by-4 rotation matrix specified by a positive right-handed rotation of *RADIANS* radians about the X axis. If *RESULT* is supplied, it is set and returned.

(ROTATE-4-BY-4-ABOUT-Y *RADIANS RESULT*) [Function]

Returns a 4-by-4 rotation matrix specified by a positive right-handed rotation of *RADIANS* radians about the Y axis. If *RESULT* is supplied, it is set and returned.

(ROTATE-4-BY-4-ABOUT-Z *RADIANS RESULT*) [Function]

Returns a 4-by-4 rotation matrix specified by a positive right-handed rotation of *RADIANS* radians about the Z axis. If *RESULT* is supplied, it is set and returned.

(SCALE-3-BY-3 *SX SY RESULT*) [Function]

Returns a 3-by-3 homogeneous scaling transformation that scales by a factor of *SX* along the X-axis and *SY* along the Y-axis. If *RESULT* is supplied, it is set and returned.

(SCALE-4-BY-4 *SX SY SZ RESULT*) [Function]

Returns a 4-by-4 homogeneous scaling transformation that scales by a factor of *SX* along the X-axis, *SY* along the Y-axis, and *SZ* along the Z axis. If *RESULT* is supplied, it is set and returned.

(TRANSLATE-3-BY-3 *TX TY RESULT*) [Function]

Returns a 3-by-3 homogeneous translation that translates by *TX* along the X-axis and *TY* along the Y-axis. If *RESULT* is supplied, it is set and returned.

(TRANSLATE-4-BY-4 *TX TY TZ RESULT*) [Function]

Returns a 4-by-4 homogeneous translation that translates by *TX* along the X-axis, *TY* along the Y-axis and *TZ* along the Z axis. If *RESULT* is supplied, it is set and returned.

(PERSPECTIVE-4-BY-4 *PX PY PZ RESULT*) [Function]

Returns a 4-by-4 homogeneous perspective transformation defined by *PX*, *PY*, and *PZ*. If *RESULT* is supplied, it is set and returned.

## Matrix Multiplication Functions

If run on workstations equipped with the extended processor option, these functions make good use of the hardware floating-point unit. The three digits at the end of each function's name describe the dimensions of their arguments.

Note: The results of the following matrix multiplication functions are not guaranteed to be correct unless the matrix arguments are all different (Not EQ).

(MATMULT-133 *VECTOR MATRIX RESULT*) [Function]

Returns the inner product of a 3-vector, *VECTOR*, and a 3-by-3 matrix, *MATRIX*. If *RESULT* is supplied, it is set and returned.

(MATMULT-331 *MATRIX VECTOR RESULT*) [Function]

Returns the inner product of a 3-by-3 matrix, *MATRIX*, and a 3-vector, *VECTOR*. If *RESULT* is supplied, it is set and returned.

(MATMULT-333 *MATRIX-1 MATRIX-2 RESULT*) [Function]

Returns the inner product of a 3-by-3 matrix, *MATRIX-1*, and another 3-by-3 matrix, *MATRIX-2*. If *RESULT* is supplied, it is set and returned.

(MATMULT-N33 *MATRIX-1 MATRIX-2 RESULT*) [Function]

Returns the inner product of an N-by-3 matrix, *MATRIX-1*, and a 3-by-3 matrix, *MATRIX-2*. If *RESULT* is supplied, it is set and returned.

(MATMULT-144 *VECTOR MATRIX RESULT*) [Function]

Returns the inner product of a 4-vector, *VECTOR*, and a 4-by-4 matrix, *MATRIX*. If *RESULT* is supplied, it is set and returned.

(MATMULT-441 *MATRIX VECTOR RESULT*) [Function]

Returns the inner product of a 4-by-4 matrix, *MATRIX*, and a 4-vector, *VECTOR*. If *RESULT* is supplied, it is set and returned.

(MATMULT-444 *MATRIX-1 MATRIX-2 RESULT*) [Function]

Returns the inner product of a 4-by-4 matrix, *MATRIX-1*, and another 4-by-4 matrix, *MATRIX-2*. If *RESULT* is supplied, it is set and returned.

(MATMULT-N44 *MATRIX-1 MATRIX-2 RESULT*) [Function]

Returns the inner product of an N-by-4 matrix, *MATRIX-1*, and a 4-by-4 matrix, *MATRIX-2*. If *RESULT* is supplied, it is set and returned.

---

## Miscellaneous Functions

---

(PROJECT-AND-FIX-3-VECTOR *3-VECTOR 2-VECTOR*) [Function]

The homogeneous *3-VECTOR* is projected onto the X-Y plane, coerced to integer coordinates (rounding by truncation) and returned. If *2-VECTOR* is supplied, it is set and returned.

(PROJECT-AND-FIX-N-BY-3 *N-3-MATRIX N-2-MATRIX*) [Function]

The homogeneous N-by-3 matrix, *N-3-MATRIX*, is projected onto the X-Y plane row-by-row, coerced to integer coordinates (rounding by truncation) and returned. If *N-2-MATRIX* is supplied, it is set and returned.

(PROJECT-AND-FIX-4-VECTOR *4-VECTOR 2-VECTOR*) [Function]

The homogeneous 4-vector, *4-VECTOR*, is projected onto the X-Y plane, coerced to integer coordinates (rounding by truncation) and returned. If *2-VECTOR* is supplied, it is set and returned.

(PROJECT-AND-FIX-N-BY-4 *N-4-MATRIX N-2-MATRIX*) [Function]

The homogeneous N-by-4 MATRIX, *N-3-MATRIX*, is projected onto the X-Y plane row-by-row, coerced to integer coordinates (rounding by truncation) and returned. If *N-2-MATRIX* is supplied, it is set and returned.

(DEGREES-TO-RADIANS *DEGREES*) [Function]

Returns *DEGREES* converted to radians.

---

## Limitations

---

MatMult is not intended as a general matrix manipulation package; it is specialized for the 3-by-3 and 4-by-4 cases.

Use CmlFloatArray for more general floating point array facilities.

## Example

```
(* ; "Try (spiral)")

(CL:DEFUN SPIRAL (&OPTIONAL (WINDOW (CREATEW))
 &AUX
 (WIDTH (WINDOWPROP WINDOW 'WIDTH))
 (HALF-WIDTH (QUOTIENT WIDTH 2))
 (HEIGHT (WINDOWPROP WINDOW 'HEIGHT))
 (HALF-HEIGHT (QUOTIENT HEIGHT 2))
 (SCALE-FACTOR (CL:EXP (QUOTIENT
 (CL:LOG (QUOTIENT (MIN WIDTH HEIGHT) 2.0)) 1440.0))))
 (LET ((LINE-1 (MAKE-HOMOGENEOUS-3-VECTOR 1.0 0.0))
 (LINE-2 (MAKE-HOMOGENEOUS-3-VECTOR))
 (TEMP (MAKE-HOMOGENEOUS-3-VECTOR))
 (POINTS (CL:MAKE-ARRAY 2))
 (TRANSFORM (MATMULT-333 (ROTATE-3-BY-3 (DEGREES-TO-RADIANS 2.5))
 (SCALE-3-BY-3 SCALE-FACTOR SCALE-FACTOR)))
 (TRANSLATION (TRANSLATE-3-BY-3 HALF-WIDTH HALF-HEIGHT)))
 (CL:DO ((L-1 LINE-1)
 (L-2 LINE-2)
 (I 0 (CL:1+ I)))
 ((EQ I 1728))
 (MATMULT-133 L-1 TRANSFORM L-2)
 (MATMULT-133 L-2 TRANSLATION TEMP)
 (PROJECT-AND-FIX-3-VECTOR TEMP POINTS)
 (DRAWLINE HALF-WIDTH HALF-HEIGHT (CL:AREF POINTS 0)
 (CL:AREF POINTS 1)
 1
 'REPLACE WINDOW)
 (CL:ROTATEF L-1 L-2))))
```

[This page intentionally left blank]

# MINISERVE

---

MiniServe contains servers for three simple protocols: Time Service (both PUP and XNS versions) and PUP ID Service. The servers are intended to run in the background on an 1108 or 1186 on networks that lack other sources of these services.

## Requirements

---

The time must be correctly set on the machine running MiniServe (see "NS Time Service" below).

## Installation

---

Load `MINISERVE.LCOM` from the library.

Either set the variable `NS.TO.PUP.ALIST` correctly, or make sure that the variable `NS.TO.PUP.FILE` is the name of a file containing a single form which will be used to set `NS.TO.PUP.ALIST` (see "PUP ID Service" below).

Evaluate `(STARTMINISERVER)`.

## Functions

---

`(STARTMINISERVER)`

[Function]

This function has no arguments; it adds three background processes to the environment, one for each of the protocols that miniserve handles. These processes and protocols are:

|                             |                               |
|-----------------------------|-------------------------------|
| <code>\NSTIMESERVER</code>  | Provides the XNS Time Service |
| <code>\PUPTIMESERVER</code> | Provides the PUP Time Service |
| <code>\PUP.ID.SERVER</code> | Provides the PUP ID Service   |

### XNS Time Service

XNS Time Service answers requests for the time using the XNS Time Protocol.

You must already have set the correct date and time on your workstation, either via one of the installation utilities or by evaluating

`(SETTIME "dd-MMM-yy hh:mm:ss")`.

If you are not in the Pacific time zone, you should also make sure the following variables are set correctly:

`\BEGIN DST`

[Variable]

The ordinal day of the year (1= January 1, 366 = December 31) on or before which daylight saving time starts in your area. Set it to 367 if your area does not observe daylight saving time.

`\ENDDST` [Variable]

The ordinal day of the year on or before which daylight saving time ends.

`\TIMEZONECOMP` [Variable]

The number of hours west of Greenwich; e.g., Eastern standard time = 5.

## PUP Time Service

PUP Time Service is like NS Time Service, but using a PUP protocol. This service is not required by any Xerox workstation as long as XNS Time Service is available, but may be of use to other workstations.

You can disable it by evaluating

```
(MOVD 'NILL' \PUPTIMESERVER) .
```

## PUP ID Service

PUP ID Service supplies workstations with PUP host numbers, given their 48-bit XNS host numbers, so that they may communicate via PUP protocols.

`NS.TO.PUP.FILE` [Variable]

The name of a file containing a single form which will be used to set `NS.TO.PUP.ALIST`. Either this variable or `NS.TO.PUP.ALIST` must be set for the PUP ID Service to work.

`NS.TO.PUP.ALIST` [Variable]

A list which maps a workstation's XNS host number to a pup host number. Elements of this list are dotted pairs of the form:

```
((NSHOSTNUMBER A B C) . PUPNUMBER)
```

where A, B, C are the three 16-bit components of the workstation's 48-bit XNS host number (the value of the variable `\MY.NSHOSTNUMBER`), and PUPNUMBER is the corresponding PUP host number to be assigned to the workstation. PUP host numbers are integers in the range [1,254], and must be unique among hosts on a single net.

To set up this list correctly you can do the following on each workstation which will use the service (including the workstation running MiniServe):

1. Decide on a unique PUP host number for this workstation. It must be an integer in the range [1,254]. For example we'll choose PUP Host number 2.
2. Get the workstation's NS host number and add it to the PUP host number. Evaluate the following form:

```
(CONS \MY.NSHOSTNUMBER YOURPUPNUMBER)
```

Using our chosen PUP host number of 2 and an example value for `\MY.NSHOSTNUMBER` the result might be:

```
((NSHOSTNUMBER 0 43520 14312) . 2)
```

3. Back on the workstation which is about to run MINISERVE, insert the dotted pair into `NS.TO.PUP.ALIST`.

## Restarting MiniServe

---

If you need to restart MiniServe:

- Use the PSW window to kill the three processes that were started by `STARTMINISERVE`.
- Evaluate `(STARTMINISERVE)`.



[This page intentionally left blank]

# NSMAINTAIN

---

NSMaintain allows you to view and modify objects in the Clearinghouse data base from inside Lisp. Similar operations are available when chatting to a Clearinghouse service.

## Requirements

---

Xerox NS network environment with Clearinghouse server(s).

DES.LCOM.

## Installation

---

Load NSMAINTAIN.LCOM from the library. This file automatically loads DES.LCOM. DES is currently only used by the Change Password command, so its loading can be omitted if you do not need that command.

## Clearinghouse Concepts

---

The Clearinghouse maintains a distributed data base of objects, each of which has a set of properties. The objects are such things as users, groups, and network servers; the properties are such attributes as a server address or a user's mailbox location.

Clearinghouse objects are partitioned into a three-level hierarchy: each object is contained in a domain, which in turn is part of an organization. A fully qualified object name is a three-part name in the form object:domain:organization. Similarly, a domain name is a two-part name of the form domain:organization. Lisp maintains a notion of the default domain, which is typically the domain in which you and the servers in your immediate area are registered. When typing object names, you may omit the organization field or both domain and organization fields if they are the same as your default domain. Similarly, when typing a domain name, you may omit the organization field if it is the same as the default.

When printing the names of objects, the system usually elides the domain and/or organization, following the same rules. For example, for the object named "John Jones:Sales:ACME", the system would print "John Jones:" if the default domain were "Sales:ACME", or "John Jones:Sales" if the default domain were "Admin:ACME". NSMaintain, however, prints fully qualified names in certain places that do not need the compactness of the elided names, so as to reduce potential confusion. For the same reason, whenever NSMaintain prompts for an object name and you omit one or two of the fields, NSMaintain automatically echoes the defaults for you. You can change the defaults with the "Change Default Domain" command.

Any object in the Clearinghouse can have one or more aliases, which are Clearinghouse names that point directly to the object. An alias can be thought of as a "nickname", and can be used interchangeably with the "real name" for virtually all operations. For example, it is common practice to register users with their full names and provide at least one alias consisting of their last names.

Some objects in the Clearinghouse are groups, rather than individuals. Groups are described further in the section on group commands.

For more information on the Clearinghouse service, consult the *Interlisp-D Reference Manual* or the Clearinghouse documentation, which is part of the Network Systems documentation kit.

## User Interface

---

NSMaintain runs in an Exec window. To start it up, evaluate:

(NSMAINTAIN)

[Function]

Starts an NSMaintain session. It prompts with "CH:" in the current window and awaits commands from you. Command names complete automatically following one- or two-letter inputs. Type Q, for the Quit command, when you wish to finish.

Most of the commands take as input from you one or more Clearinghouse object names. For many commands, NSMaintain offers you the same name as you last used in a similar context. For example, if you use the Describe command to learn about an object that is a group and then use the List Members command, NSMaintain offers you the group name just described. To accept the name, just press the carriage return; otherwise, start typing the desired name; your type-in replaces the offered name. The alphabetic case of names is not significant; you may type in either upper or lowercase. However, the commands that create objects preserve the exact case of the name as you first type it.

Typing a null name to most commands aborts the command. If a sample name is offered, you have to backspace over it, or use Control-Q to erase the whole name. You can also usually use Control-E to abort a command.

The description of the commands below is partitioned into two parts: general user commands and administrator commands. The user commands can be used by anyone, and mostly are concerned with viewing the data base. The administrator commands allow system administrators to modify the data base; these commands cannot be used by ordinary users. However, there are two administrator commands, Add Self and Remove Self, that can be used by anybody to join or leave groups with open access.

## User Commands

Anyone can use these commands to obtain information, change passwords, and change NSMaintain's defaults.

### Obtaining Information

These commands let you examine the database.

Most of the List commands enumerate items in the database matching a particular pattern. A pattern is a Clearinghouse name optionally containing asterisks as wild cards, which match zero or more characters. Wild cards are permitted only in the first component of the name; the remaining parts must be a valid domain and organization. In a two-part name, wild cards are permitted in the domain name but not the organization. Thus, for example, "\*John\*:Sales:ACME" is a valid pattern matching objects whose name component contains the substring "John"; "Joe\*:ACME" is not a valid pattern.

Following a List command (except List Domains), you can use the Show Details command to get more information about any of the names listed.

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Describe            | Gives a description of any object registered in the Clearinghouse, what its registered name is (in case you typed an alias), and all interesting properties of the object. If the object is a group, its Owner and Friends are also listed; to see its members, use the List Members command.                                                                                                                  |
| List Domains        | Lists all domains matching a specified domain pattern; for example, "*:Xerox" to list all domains in the Xerox organization.                                                                                                                                                                                                                                                                                   |
| List Clearinghouses | Lists all Clearinghouse servers that serve a specified domain.                                                                                                                                                                                                                                                                                                                                                 |
| List Administrators | Lists the administrators for a domain.                                                                                                                                                                                                                                                                                                                                                                         |
| List Aliases        | Lists all aliases matching a given pattern. Note that none of the other List commands (except for List Objects with property any) match your pattern against an alias, so you may want to use the List Aliases command if you do not find the object you were looking for otherwise.                                                                                                                           |
| List Groups         | Lists all groups matching a given pattern.                                                                                                                                                                                                                                                                                                                                                                     |
| List True Groups    | Same as List Groups, but filters out all names that also have a "user" property. These "groups" are typically used for mail forwarding. This command requires considerably more computation than List Groups.                                                                                                                                                                                                  |
| List Servers        | Lists objects matching a given pattern and registered as a server. You are prompted for the type of server; for example, Mail, File, or Print. Type ? to see the choices.                                                                                                                                                                                                                                      |
| List Users          | Lists the names of all users matching a given pattern.                                                                                                                                                                                                                                                                                                                                                         |
| List Objects        | Lists all registered objects of of an arbitrary Clearinghouse type that match a given pattern. You are prompted for the type(a Clearinghouse property name) and pattern. To list all objects, that is, those with any property, press the carriage return to the property prompt, or supply the property "*".                                                                                                  |
| List Members        | Lists the members of a specified group.                                                                                                                                                                                                                                                                                                                                                                        |
| Show Details        | <p>Prompts you for a name, performing automatic spelling completion from the names printed in the most recent List command such as List Users, and then performs the Describe command on the name. Press the carriage return in response to the name prompt to return to the main CH: prompt.</p> <p>If there was only one name in the list, this command does a Describe on it without further prompting.</p> |
| Type Entry          | Synonym for Describe.                                                                                                                                                                                                                                                                                                                                                                                          |
| Type Members        | Synonym for List Members.                                                                                                                                                                                                                                                                                                                                                                                      |

## Miscellaneous

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Change Default Domain | <p>Changes the defaults used on type-in inside NSMaintain for domain and organization. NSMaintain asks if you also want to change the defaults globally; if you say yes, the variables <code>CH.DEFAULT.DOMAIN</code> and <code>CH.DEFAULT.ORGANIZATION</code> are changed, so that the new defaults have effect outside of NSMaintain as well.</p> <p>The defaults are never used when typing aliases in the Add Alias and Add User commands; these commands default the domain to be the same as the domain of the main object.</p> <p>Note: Unless you change the defaults globally, this command does not affect type-out and the Change Login command, which are still performed with respect to the global defaults. This may change in a future implementation.</p> |
| Change Login          | Prompts you for a new name and password, which becomes the default NS login on your machine and for NSMaintain. You can also use this to fix your password if you were incorrectly logged in before you started NSMaintain.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Change Password       | Allows you to change your password. Prompts for a user name, offering your logged-in name as default. A domain administrator can also change other users' passwords. After you type the new password, you are asked to retype the password, to ensure that you typed what you thought you had. Neither password is echoed.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Quit                  | Exit NSMaintain.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

## Administrator Commands

These commands modify the Clearinghouse database. In general, they require that you have the appropriate administrator access.

### Creating and Deleting Objects

To create or delete objects in a domain, you must be an administrator for the domain.

|               |                                                                                                                                                                                                                  |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Add Alias     | Assigns an alias for a specific object registered in the Clearinghouse database.                                                                                                                                 |
| Add User      | Creates a new user. You are prompted for the user's name (preferably a full name), a brief description (for example, the user's affiliation, office number, etc.), an initial password, and one or more aliases. |
| Change Remark | Allows you to change the remark; that is, text description, of any object in the database. NSMaintain prompts you for the object name, then for a new remark, offering the old remark as default.                |
| Remove Alias  | Removes an Alias from the database. You are prompted for the alias. This has no affect on the primary object for which the removed name is an alias.                                                             |

|                          |                                                                                                                                                                                                                                                |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Remove User              | Undoes the effect of Add User; that is, removes a user name and all its properties from the database.                                                                                                                                          |
| Remove Registered Object | Removes a specified object and all its properties from the database. The primary name and description of the object are printed first and you are asked to confirm the deletion. You can use this to remove groups and other kinds of objects. |

## Manipulating Groups

A group is a Clearinghouse object with members. Groups are most commonly used for mailing lists and access control. The members can be either individuals or other groups. In the case of groups used for access control, a member can also be a pattern in which "\*" usually replaces one or more entire fields of a three-part name.

A group has associated with it two access control lists: owners and friends. Owners can make any change to a group; they are like domain administrators for the narrow scope of the group itself. Friends are allowed to add or remove themselves from the group. For example, common interest groups typically have "open" membership, consisting of a friends list of "\*:domain", or even "\*:\*:\*" for a completely open group.

If the Owners or Friends list is empty, it defaults to the administrators of the domain. However, if the Owners list is non-empty, it overrides the administrators list. For example, if you remove yourself from the owners of a group, you can no longer modify the list, even though you are a domain administrator. The defaulting can lead to confusion, especially since the Describe command does not (and unfortunately cannot) indicate whether the owners and friends it displays are explicit or defaulted. For example, if a group previously had no explicit owners, then the Remove Owner command cannot be used, and any use of the Add Owner command implicitly removes all the domain administrators.

**Add Group** Creates a new user group. You are prompted for the group's name, a short description of the group, its initial members one at a time, its owners and its friends. NSMaintain checks all of the names except those that are patterns to ensure that you gave valid Clearinghouse names, and to resolve aliases. The Clearinghouse does not actually require that members of a group be registered Clearinghouse names, as it does not attach explicit meaning to the contents of a group until told to do so. Thus, if you type an invalid name, NSMaintain asks whether you really meant it, and keeps the name if you answer yes. Note, however, that any group used for access control must contain only registered Clearinghouse names or patterns.

If you specify any owners, you are always made an owner yourself as well, whether you explicitly said so or not, so as to avoid the anomaly of your not being able to further modify the group. You can, of course, remove yourself afterward if you really meant to.

**Add Member**

**Add Friend**

**Add Owner** Adds a member, friend or owner to a group.

**Add Self** Adds you, the currently logged in user, to a group. You must be a friend or owner of the group.

**Remove Member**

|               |                                                                                                      |
|---------------|------------------------------------------------------------------------------------------------------|
| Remove Friend |                                                                                                      |
| Remove Owner  | Removes a specified member, friend or owner from a group.                                            |
| Remove Self   | Removes you, the currently logged in user, from a group. You must be a friend or owner of the group. |

## Manipulating Domains

These commands change the list of administrators of a domain. You must be an administrator of the domain or the parent organization to do this.

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| Add Domain Administrator    | Adds a user to the set of administrators for a domain.         |
| Remove Domain Administrator | Removes a specified user from the administrators for a domain. |

## Errors

NSMaintain always ends each command with some sort of feedback about the completion of the operation. In information commands, the feedback is, of course, the requested information. In commands that change the database, NSMaintain usually prints "done". If a command fails, NSMaintain prints a terse error message. Listed here are some of the more common ones:

|                          |                                                                                                                                                                                                                                                                                             |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NoSuchObject             | You asked about a name that does not exist in the Clearinghouse database. Check that the spelling and the domain are correct.                                                                                                                                                               |
| IllegalOrganization      |                                                                                                                                                                                                                                                                                             |
| IllegalDomain            |                                                                                                                                                                                                                                                                                             |
| IllegalObject            | The name you gave is not legal as a Clearinghouse name. Since NSMaintain already checks for incorrect use of asterisks, this usually means the name is too long. (The name component must be no more than 40 characters long; domains and organizations are limited to 20 characters each.) |
| Missing                  | The name you specified for a group, such as in the AddMember command, is not a group.                                                                                                                                                                                                       |
| CredentialsInvalid       |                                                                                                                                                                                                                                                                                             |
| VeriferInvalid           | You are logged in incorrectly; that is, either your name or your password is incorrect. You can use the Change Login command to log in correctly.                                                                                                                                           |
| AccessRightsInsufficient | You do not have the authority to make the change you requested. You can find out who does have the authority by using the command Describe for changing a group, or List Domain Administrators for all other changes.                                                                       |
| NoChange                 | The change you requested would have no effect; for example, you added to a group a name that was already a member, or requested to remove a name that was not there.                                                                                                                        |
| TooBusy                  | The Clearinghouse contacted by NSMaintain was too busy to field the request. Lisp's present Clearinghouse implementation, unfortunately, does not handle this error, so passes it along to you. If you repeat the operation it may succeed. If this error persists for a long time, you may |

want to evaluate (START.CLEARINGHOUSE T) to completely clear the Clearinghouse cache; the system may then succeed in locating a more responsive server.

## Examples

In the example session that follows, all user input is in boldface; everything else is typed by the system. To avoid clutter, carriage returns typed by the user are not shown. In many cases, a simple carriage return accepts the default input typed by the system, or completes a partially typed name. For clarity, most of the user input is in uppercase, although lowercase is equally acceptable. Commentary is in italics.

64> (NSMAINTAIN)

[Default login: Arthur Dent:Research:ACME;  
Default domain: Research:ACME]

*NSMaintain shows me the defaults.  
(My password has not, however, been verified.)*

CH: Describe name: **EDISON**:Research:ACME ...

Thomas A. Edison:Research:ACME is a User (Electronics Div., Rm 2732)  
Aliases: Edison:, Wizard: *Domain and organization are elided here*  
Mailboxes: [Time: 8-Aug-86 17:30:54; Mail.Service: (Snail:)]  
Userdata: [Last.Name.Index: 10; File.Service: Phylum:]

*The Userdata property is used by Viewpoint*

CH: List Groups by pattern: \*:Research:ACME ... AllResearch, Consultants,  
ED, LispImplementors, LispInterest, NetAdministration, Skiers, Staff,  
WireBusters

CH: Show Details of previously listed names

name: **Consultants** *"C<cr>" is all that I typed*

Consultants:Research:ACME is a User Group (Part-time personnel)  
Owners: Staff:  
Friends: NetAdministration:

name: **SKiers**

Skiers:Research:ACME is a User Group (Snow sport enthusiasts)  
Owners: UserAdministration:All Areas, Perry White:  
Friends: **::\*** *Anyone in organization ACME can join*

name:

CH: List Members of group: Skiers:Research:ACME ...  
Alexander G. Bell:Telcom, Christopher Craft:, Staff:

CH: Add Self to group: Skiers:Research:ACME ... done  
*Skiers was offered as default, being the last  
group I mentioned—I had only to type a cr.*



CH: Add Self to group: Skiiers:Research:ACME ... failed: NoChange  
*I.e., I'm already a member*

CH: List Servers of type File  
by pattern: **\*:Development:ACME** ... Arrow, Quiver

CH: List Users by pattern: Ed\*:Research:ACME ... (none)  
*There are no users whose full name starts "Ed"*

CH: List **AL**iasess by pattern: **Ed\***:Research:ACME ... Edison, Educators  
*But there are some aliases (not necessarily all users)*

CH: List objects having property WORKSTATION  
by pattern: **\*M\***:Research:ACME ... Archimedes, Camero, Cardamom, Homestead,  
MayDay, Mendel, Ramanujan, SatanicMechanic, TheTajMahal

CH: Show Details of previously listed names  
name: Archimedes

Archimedes:Research:ACME is a Workstation (1186 in Rm. 2732)  
Address.List: (6285#0.125101.20200#0)  
Authentication.Level: [Simple: true; Strong: false]

CH: Change Default Domain (for name entry) to be: Development:ACME  
Set this default globally as well (i.e. for use outside Maintain)? N

CH: Add Alias for object: Newton:Development:ACME  
Alias: Isaac:Development:ACME ... done

CH: Describe name: **Newton**:Development:ACME ...

S. Isaac Newton:Development:ACME is a User (Apple Tester, Rm. 34)  
Aliases: Isaac:Development, **SIN**:Development  
Userdata: [Last.Name.Index: 0; File.Service: Arrow:Development]

CH: Remove Alias alias: SIN:Development:ACME ... done, alias was removed  
from S. Isaac Newton:Development

CH: Add User  
New user's name: Charles S. Brown:Development:ACME ...  
Remark (terminate with CR): Test Team captain  
Alias: **Chuck**:Development:ACME  
Alias: **Brown**:Development:ACME  
Alias: **CSB**:Development:ACME  
Alias: xxx *Bare <cr> was typed to end the list*  
Initial password: \*\*\*\*\* (retype password) \*\*\*\*\*... done

*Chuck can later use Change Password to  
set a password of his own choosing.*

CH: Add Group  
New group name: Entomologists:Development:ACME ...  
Remark (terminate with CR): **Seekers of bugs**

Enter names of members, owners and friends, one per line, terminated with a  
blank line.

Member: **brown**:Development:ACME = Charles S. Brown:Development:ACME  
*NSMaintain resolves alias, so that member  
names are in canonical form*

Member: **F. Kafka**:Development:ACME  
Member: xxx

(If you enter no owners, the group will be owned by the administrators of  
Development:ACME.)

Owner: brown:Development:ACME = Charles S. Brown:Development:ACME  
Owner: xxx

Friend: \*:Development:ACME  
Friend: **\*:Research**:ACME  
Friend: xxx

Adding members... done  
Adding owners... (including Arthur Dent:Research:ACME) done  
*I'm an owner, too (else I couldn't modify the group)*  
Adding friends... done

CH: Remove User: **BILBO**:Development:ACME ...  
Bilbo Baggins:Development:ACME (Furry ring finder)  
Confirm deletion (y or n): Y  
done

CH: Quit [confirm] *Back to the exec now.*

[This page intentionally left blank]

---

**POSTSCRIPTSTREAM**

---

By: Matt Heffron (mheffron@orion.cf.uci.edu)

**INTRODUCTION**

The PostScript package defines a set of imageops for printers which understand the PostScript page description language by Adobe. At Beckman we have successfully used TEdit, Sketch, LISTFILES, and HARDCOPYW to an Apple LaserWriter and an AST TurboLaser PS. The PostScript imagestream driver installs itself when it is loaded. All symbols in the PostScript driver are located in the INTERLISP: package.

**VARIABLES**

POSTSCRIPT.FONT.ALIST [InitVariable]

POSTSCRIPT.FONT.ALIST is an ALIST mapping Xerox Lisp font names into the root names of PostScript font files. It is also used for font family coercions. The default value should be acceptable for any of the fonts which are built into the Apple Laserwriter.

POSTSCRIPTFONTDIRECTORIES [InitVariable]

POSTSCRIPTFONTDIRECTORIES is the list of directories where the PostScript .PSCFONT font files can be found. The default value is: ("{DSK}/usr/local/lde/fonts/postscript/") on a Sun or IBM workstation and ("{DSK}<LISPFILES>FONTS>PSC>") for other cases .

POSTSCRIPT.DEFAULT.PAGEREGION [InitVariable]

POSTSCRIPT.DEFAULT.PAGEREGION indicates the area of the page to use for text file listings (i.e. LISTFILES). It is in units of 100'ths of points. The default value is: (4800 4800 52800 70800), which gives left and bottom margins of 0.75 inch and top and right margins of 0.5 inch on 8.5 x 11 paper.

POSTSCRIPT.PAGEREGIONS [InitVariable]

POSTSCRIPT.PAGEREGIONS is an ALIST mapping pagetypes into paper size and actual imageable area on the page. By default, it knows about LETTER, LEGAL, and NOTE pagetypes, and the corresponding sizes and imageable areas for the Apple Laserwriter. Others can be defined by the user by adding the appropriate entries onto this ALIST.

POSTSCRIPT.PAGETYPE [InitVariable]

POSTSCRIPT.PAGETYPE is used by OPENIMAGESTREAM to lookup the paper size and actual imageable area of the page in POSTSCRIPT.PAGEREGIONS to determine the initial margins. This value can be overridden with the PAGETYPE or PAPERTYPE options in the OPENIMAGESTREAM call. The name of the type of page selected is NOT passed through to the PostScript output.

POSTSCRIPT.MAX.WILD.FONTSIZE [InitVariable]

POSTSCRIPT.MAX.WILD.FONTSIZE indicates the maximum point size that should be returned from FONTSAVAILABLE when the SIZE argument is wild (i.e. \*). All integer point sizes from 1 to POSTSCRIPT.MAX.WILD.FONTSIZE will be indicated as available. The default value is: 72.

POSTSCRIPT.PREFER.LANDSCAPE [InitVariable]

POSTSCRIPT.PREFER.LANDSCAPE indicates if the OPENIMAGESTREAM method should default the orientation of output files to LANDSCAPE. It can have one of three values: NIL, T, or ASK. NIL means prefer portrait orientation output, T means prefer landscape, and ASK says to bring up a menu to ask the preferred orientation if it wasn't explicitly indicated in the OPENIMAGESTREAM call (with the ROTATION option). The default value is: NIL. An item (PS Orientation) is added to the Background Menu to let you change the value of this variable.

POSTSCRIPT.TEXTFILE.LANDSCAPE [InitVariable]

POSTSCRIPT.TEXTFILE.LANDSCAPE indicates if the printing of TEXT files (e.g. LISTFILES, ...) should force the orientation of output files to LANDSCAPE. When it is non-NIL the orientation of output files is forced to LANDSCAPE. (There is no ASK option here.) The default value is: NIL.

POSTSCRIPT.BITMAP.SCALE [InitVariable]

POSTSCRIPT.BITMAP.SCALE specifies an independent scale factor for display of bitmap images (e.g. window hardcopies). Values less than 1 will reduce the image size. (I.e. a value of 0.5 will give a half size bitmap image.) The position of the scaled bitmap will still have the SAME lower-left corner (i.e. the scaled bitmap is not centered in the region of the full size bitmap image). The default value is: 1.

#### HINT

Setting POSTSCRIPT.BITMAP.SCALE to 0.96, instead of 1, will give cleaner BITMAP images on a 300 dpi printer. (This corrects for the 72 ppi imagestream vs. the 75 dpi printer, using 4x4 device dots per bitmap pixel.) Also, values of 0.24, 0.48 and 0.72, instead of 0.25, 0.5 and 0.75, will also give cleaner images for reduced size output. In general, use integer multiples of 0.24 for a 300 dpi printer.

POSTSCRIPT.TEXTURE.SCALE [InitVariable]

POSTSCRIPT.TEXTURE.SCALE specifies an independent scale for the display of bitmap textures. The value represents the number of device space units per texture unit (bitmap bit). The default value is 4, which represents each bit of the texture as a 4x4 block, so that textures are approximately the same resolution as on the screen (for 300 dpi output devices, such as the Apple Laserwriter).

The PostScript package extends the allowed representations of a texture, beyond 16-bit FIXP and 16x16 bitmap, to ANY square bitmap. (If the bitmap is not square, its longer edge is truncated from the top or right to make it square.) Use this feature with caution, as large bitmap textures, or sizes other

than multiples of 16 bits square, require large amounts of storage in the PostScript interpreter (in the printer controller), and can cause limitcheck errors when actually printing.

Anywhere that a texture or color can be used on an imagestream or in the specification of a BRUSH, you can instead give a FLOATP between 0.0 and 1.0 (inclusive) to represent a PostScript halftone gray shade. (0.0 is black and 1.0 is white. Specifically, the value sets the brightness of the shade.) The value you specify will not be range checked, and will be passed directly through to the PostScript setgray operator. (E.g. you can pass 0.33 as the color to DRAWLINE to get a dark gray line with approximately 67% of the pixels in the line black.)

POSTSCRIPT.IMAGESIZEFACTOR

[InitVariable]

POSTSCRIPT.IMAGESIZEFACTOR specifies an independent factor to change the overall size of the printed image. This re-sizing affects the entire printed output (specifically, it superimposes its effects upon those of POSTSCRIPT.BITMAP.SCALE and POSTSCRIPT.TEXTURE.SCALE). Values greater than 1 enlarge the printed image, and values less than 1 reduce it. An invalid POSTSCRIPT.IMAGESIZEFACTOR (i.e. not a positive, non-zero number) will use a value of 1. The BITMAPSCALE function for the POSTSCRIPT printer type does NOT consider the POSTSCRIPT.IMAGESIZEFACTOR when determining the scale factor for a bitmap.

## MISCELLANEOUS

The SCALE of a PostScript imagestream is 100. This is to allow enough resolution in the width information for fonts to enable TEdit to correctly fill and justify text.

The first time any PostScript imagestream is created (even if only to hardcopy a bitmap or window) the DEFAULTFONT is instantiated (unless a FONTS option was given to the OPENIMAGESTREAM, in which case the initial font for the imagestream will be set to that font, or to the CAR if a list).

The PostScript imagestream method for FILLPOLYGON uses the global variable FILL.WRULE as the default value for the WINDINGNUMBER argument. (This is the same variable which is used by the DISPLAY imagestream method for FILLPOLYGON.)

The PostScript imagestream method for OPENIMAGESTREAM (and, therefore, SEND.FILE.TO.PRINTER), supports an IMAGESIZEFACTOR option to change the size of the printed image. The IMAGESIZEFACTOR re-sizing is combined with the POSTSCRIPT.IMAGESIZEFACTOR to produce an overall re-sizing of the printed image. A HEADING option is also supported to give a running header on each page of output. The value of the HEADING option is printed at the top left of the page, followed by "Page " and the appropriate page number. They are printed in the DEFAULTFONT (unless a FONTS option was given to the OPENIMAGESTREAM, in which case it will be that font, or to the CAR if a list).

The PostScript package is contained in the files: POSTSCRIPTSTREAM.LCOM & PS-SEND.LCOM, with the source in the files: POSTSCRIPTSTREAM & PS-SEND. The module PS-SEND.LCOM is required and will be loaded automatically when POSTSCRIPTSTREAM.LCOM is loaded. It contains the function which is called by SEND.FILE.TO.PRINTER to actually transmit the file to the printer. It is, by its nature, quite site specific, so it is in a separate file to make modifying it for any site relatively

simple. System record declarations required to compile POSTSCRIPTSTREAM can be found in EXPORTS.ALL.

I'm pretty sure that the output generated by the PostScript imageops fully conforms to the Adobe Systems Document Structuring Conventions, Version 2.0, January 31, 1987.

### Including Other PostScript Operations

If you wish to insert your own specific PostScript operations into a PostScript imagestream, you can do so with the following functions:

(POSTSCRIPT.OUTSTR *STREAM STRING*) [Function]

POSTSCRIPT.OUTSTR outputs a string or value to the imagestream. *STREAM* must be an open PostScript imagestream. *STRING* is the value to output (STRINGP and LITATOM are most efficient, but any value can be output (its PRIN1 pname is used)).

(POSTSCRIPT.PUTCOMMAND *STREAM STRING ... STRING*) [NoSpread Function]

POSTSCRIPT.PUTCOMMAND is more general for sequences of commands and values. It calls POSTSCRIPT.OUTSTR repeatedly to output each of the *STRING* arguments to *STREAM*.

(\POSTSCRIPT.OUTCHARFN *STREAM CHAR*) [Function]

\POSTSCRIPT.OUTCHARFN is used to output the characters forming the text of a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream to output to, and *CHAR* is the CHARCODE of the character to output. The / (slash), ( and ) (parenthesis) characters will be quoted with /, and characters with ASCII values less than 32 (space) or greater than 126 (tilde) will be output as *nnn* (in octal). \POSTSCRIPT.OUTCHARFN will output the ( character to open the string, if necessary. Use POSTSCRIPT.CLOSESTRING (below) to close the string.

(POSTSCRIPT.CLOSESTRING *STREAM*) [Function]

POSTSCRIPT.CLOSESTRING closes a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream. It is important to use POSTSCRIPT.CLOSESTRING to output the ) character to close the string, because it also clears the stream state flag that indicates that a string is in progress (otherwise, the next POSTSCRIPT.PUTCOMMAND would output the commands to close the string and show it).

### Warning

Do not attempt to create a PostScript font larger than about 600 points, as much of Interlisp's font information is stored in SMALLP integers, and too large a font would overflow the font's height, or the width for any of the wider characters. (I know that 600 points is a ridiculously large limit (about 8.3 inches), but I thought I'd better mention it, or someone might try it!)

### Changes from the Initial Medley Release

This second Medley release of the PostScript imagestream driver includes some performance enhancements when writing bitmaps to the output, some SUN-specific code (from Will Snow of envos), implementation of the SCALEDIBITBLT, DSPROTATE, and DSPTRANSLATE operations, and a lot of performance enhancements (many thanks to Tom Lipkis of Savoir).

### **Changes from the Lyric Release**

The Medley release of this PostScript imagestream driver changed the default value of POSTSCRIPT.TEXTFILE.LANDSCAPE from T to NIL. It also added the support for the HEADING option.

### **Known Problems/Limitations**

The output generated for a PostScript imagestream is rather brute force. It isn't particularly careful to generate the smallest output file for a given sequence of operations. Specifically, it often generates extra end-of-lines between PostScript operator sequences (this has no effect on the printed output, only on the file size).

Using BITMAPs or Functions as BRUSH arguments to the curve drawing functions is not supported, nor is using a non-ROUND BRUSH with DRAWCIRCLE or DRAWELLIPSE.

The implementation of DSPROTATE accepts ROTATION argument values of 0 and 90 (any non-NIL, non-zero value is converted to 90). A value of 0 converts the page orientation to Portrait, and 90 converts the page orientation to Landscape. These conversions perform the translations necessary to keep the clipping region on the page. (This may or may not be the right thing to do, but since DSPROTATE is undocumented in what it should do, this is what the PostScript driver does).

There is no support for NS character sets other than 0, and there is no translation of the character code values from NS encoding to PostScript encoding.

There is no support for color.

\POSTSCRIPT.OUTCHARFN is pretty wimpy in its handling of TAB characters. It just moves to the next multiple of (eight times the average character width of the current font) from the current left margin.

I haven't yet documented how to build the .PSCFONT files from .AFM files for new fonts that become available.



# PRESS

---

Xerox Lisp includes utilities for generating hardcopy in "Press" format. This is a file format for communicating documents to Xerox laser xerographic printers of the Press, Spruce or Fullpress class, which are known by the names Dover, Penguin, Raven and Spruce.

Note: Press, along with other classes of printers, is described fully in two sections in the *IRM*, "Hardcopy Facilities" and "Fonts."

The reason for putting this document into the *Lisp Library Modules Manual* is that support of these printers is no longer an integral part of the system.

## Requirements

---

FONTS.WIDTHS

PUPPRINT.LCOM

A Press print server

## Installation

---

Load PRESS.LCOM from the library.

Set the variable PRESSFONTWIDTHFILES (see below).

## User Interface

---

You can use the usual means:

- HARDCOPY in background (right-button) menu
- HARDCOPY in FileBrowser menu
- Functions typed into the exec window (see below)

## Functions

---

|                      |            |
|----------------------|------------|
| DEFAULTPRINTINGHOST  | [Variable] |
| PRESSFONTWIDTHFILES  | [Variable] |
| SEND.FILE.TO.PRINTER | [Function] |
| LISTFILES            | [Function] |

(For details about all the above, see the *IRM*.)

## Limitations

---

PRESS does not support NS characters or NS fonts.

[This page intentionally left blank]

# READNUMBER

---

ReadNumber contains functions for implementing a calculator-type menu for entering numbers with the mouse.

## Installation

---

Load READNUMBER.LCOM from the library.

## Functions

---

ReadNumber functions are called either from the Executive window or programmatically from another process.

The numbers captured by ReadNumber are passed to whatever process currently has the TTY.

### Create a Key Pad

(RNUMBER MSG POSITION MSGFONT DIGITFONT INCLUDEABORTFLG  
FLOATINGPTFLG POSITIVEONLYFLG ACCEPTTYPEINFLG ) [Function]

Brings up a menu that looks like a ten-key calculator pad. Your selections, made by pressing the left mouse button when the cursor is on a digit, are accumulated in a displayed total. The key pad includes a backspace key (BS), a clear key (CLR), and a +/- key (-). When OK is selected, the total is returned.

If *MSG* is given, it is displayed at the top of the menu.

If *POSITION* is given, the menu is put there; otherwise it is put at the cursor.

If *MSGFONT* is given, *MSG* is printed in it. If *MSGFONT* is NIL, DEFAULTFONT is used.

If *DIGITFONT* is given, the labels on the keys is printed in that font. If *DIGITFONT* is NIL, BOLDFONT is used.

If *INCLUDEABORTFLG* is non-NIL, the menu also includes an abort key (abt). If the abort key is pressed, RNUMBER returns NIL.

Note: If this option is set, you will not be able to use the backspace to correct mistakes. You will have to use CLEAR and begin the number again.

If *FLOATINGPTFLG* is non-NIL, the menu includes a decimal point, and the value returned may be a floating point number.

If *POSITIVEONLYFLG* is non-NIL, the menu does not include a +/- key (-) and you can only input positive numbers (but see ACCEPTTYPEINFLG ).

If *ACCEPTTYPEINFLG* is non-NIL, the menu also responds to user-typed input (i.e., numbers typed in on the keyboard, rather than selected with the mouse). In this mode, carriage return corresponds to OK.

Note: The decimal point (.) and the minus sign (-) are also accepted, even though they are not options in the key pad menu.

If you close the key pad window, the action taken by RNUMBER depends upon the value of *INCLUDEABORTFLG*. If *INCLUDEABORTFLG* is NIL, RNUMBER generates an error (i.e., calls (ERROR!)). If *INCLUDEABORTFLG* is non-NIL, RNUMBER returns NIL (the same thing it does if the abort key is pressed).

## Create a Key Pad for Repeated Use

For some applications, it may be beneficial to avoid the creation of the key pad menu window each time a number is asked for. The following functions allow you to create a key pad menu window and use it repeatedly to get values from you.

Note: When used in this manner, a key pad menu window can only be used by one process at a time.

(CREATE .NUMBERPAD .READER MSG WPOSITION MSGFONT DIGITFONT  
INCLUDEABORTFLG FLOATINGPTFLG POSITIVEONLYFLG) [Function]

Creates a window suitable for use by NUMBERPAD .READ (see below). Its arguments are the same as for the function RNUMBER.

(NUMBERPAD .READ NUMBERPAD/READER ACCEPTTYPEINFLG) [Function]

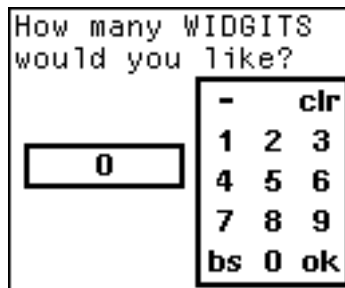
NUMBERPAD/READER should be a window returned by the function CREATE .NUMBERPAD .READER (see above). NUMBERPAD .READ uses the window in the same manner as the function RNUMBER.

## Examples

---

(RNUMBER "How many WIDGITS would you like?")

results in the following pop-up menu:



```
(RNUMBER "How far to the left?") NIL '(CLASSIC 12) '(MODERN 14) T T)
```

results in the following pop-up menu:

| How far to the left?           |   |     |     |
|--------------------------------|---|-----|-----|
| <input type="text" value="0"/> | - | AB  | CLE |
|                                |   | ORI | AR  |
|                                | 1 | 2   | 3   |
|                                | 4 | 5   | 6   |
|                                | 7 | 8   | 9   |
|                                | . | 0   | ok  |

## Limitations

If you choose both `FLOATNGPOINTFLG` and `INCLUDEABORTFLG`, there is no room for the backspace key (see the illustration above). Correct your input by selecting `CLEAR` and starting over. However, if `ACCEPTTYPEINFLG` is `T`, you can use the keyboard's backspace key.

[This page intentionally left blank]

The 1108 and 1186 each support two RS232 ports. One of these ports is configured as Data Terminal Equipment, and is intended to be connected to modems or terminal ports on other computers. (On the 1108, this port is available only with the addition of the E30 option.) The other port is configured as Data Communication Equipment, and is meant to drive printers or terminals. In this document, the DTE port is called the RS232 port, and the DCE port is called the TTY port.

Lisp provides a stream-oriented interface to the RS232 hardware. Users' programs can open streams to the hosts {RS232} or {TTY}, and perform input or output using standard Lisp I/O functions, such as READ-BYTE, READ-CHAR, etc.

Programs may use RS232 streams or TTY streams with the same programmatic interface. However, the RS232 port is preferred over the TTY if the application expects to handle large amounts of input data. In the 1108 and 1186, data entering the RS232 port is buffered independently of Lisp by the I/O processor (IOP). In addition, the RS232 software provides an additional layer of character buffering, freeing user programs from having to monitor the RS232 hardware frequently.

No independent buffering is provided for data entering the TTY port. As a result, Lisp cannot guarantee to catch all characters received on this port. For this reason, the TTY port should be used primarily to drive output devices such as printers.

## Requirements

---

### Hardware

To connect to a modem or another device, you need an RS232 or TTY cable (see the wiring diagrams in the Introduction of this manual).

For the 1108 only, you need the E-30 upgrade kit.

For the RS232 port to operate correctly, the remote device must assert the standard RS232 signals DSR and CTS. This requirement means that the appropriate pins on the 1186 be driven high, either properly by connecting the 1186 side to the corresponding device side, or permanently by jumpering the pins.

For the TTY port to operate correctly, the remote device must assert DTR and RTS. This requirement means that the appropriate pins on the 1186 be driven high, either properly by connecting the 1186 side to the corresponding device side, or permanently by jumpering the pins.

### Software

You need the following .LCOM files in order to run this module successfully:

DLRS232C or DLTTY (one of these is required)

RS232MENU

RS232CHAT or TTYCHAT, and KERMIT (these are optional).

(See also the file dependencies enumerated in the Introduction of this manual.)

## Installation

---

Load the required .LCOM modules from the library.

Run RS232C.INIT or TTY.INIT to set parameters.

## Using the RS232 Port

---

Support for the RS232 port is contained in the file DLRS232C.LCOM. Before using the RS232 port, it is necessary to initialize the RS232 hardware. The function RS232C.INIT is provided for this purpose:

```
(RS232C.INIT BAUDRATE BITSPERSERIALCHAR PARITY NOOFSTOPBITS
FLOWCONTROL)
```

 [Function]

The arguments correspond to the port parameters (see below).

Alternatively, the *BAUDRATE* argument can be an instance of the RS232C.INIT record. If *BAUDRATE* is NIL, the value of the global variable RS232C.DEFAULT.INIT.INFO is used in its place. This provides a means of automatically initializing the RS232 hardware without user intervention.

RS232C.DEFAULT.INIT.INFO [Variable]

This variable controls default initialization of the RS232 port. Its value may be set in the site INIT.LISP file, or in your INIT.LISP file. If RS232C.DEFAULT.INIT.INFO is not set when the RS232 module is loaded, its fields will be set to the following default values:

BaudRate: 1200

BitsPerSerialChar: 8

Parity: NONE

NoOfStopBits: 1

FlowControl: XOnXOff

Programs may use the Lisp function OPENSTREAM as an alternative to calling RS232C.INIT directly, with the parameters bundled up into the PARAMETERS argument.

For example, (RS232C.INIT 9600 8) can also be achieved by:

```
(OPENSTREAM '{RS232} 'INPUT NIL '((BaudRate 9600)
(BitsPerSerialChar 8))
```

(RS232C.SET.PARAMETERS *PARAMETERLIST*) [Function]

This function allows applications to change the settings of the RS232 hardware while the RS232 port is in use.

*PARAMETERLIST* is an association list of parameter names and values. The following example sets the baud rate to 9,600 baud, and the character length to eight bits:

```
(RS232C.SET.PARAMETERS '((BaudRate . 9600) (BitsPerSerialChar
. 8)))
```



The following is a list of legal parameter names and values:

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BaudRate          | 1186: 50, 75, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200.<br><br>1108: all of the above, and 28800, 38400, 48000, 56000, 57600.                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| BitsPerSerialChar | 5, 6, 7, 8 bits of data. If 5 or 6 bits of data are sent, they should be DATA, not CHARACTER.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Parity            | NONE, ODD, EVEN (1 parity bit).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| NoOfStopBits      | 1, 1.5, 2 (i.e., the stop bit may have a period equal to 1, 1.5 or 2 bit-widths).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| FlowControl       | NIL, NONE, XOnXOff, list.<br><br>NIL and NONE: no flow control.<br><br>XOnXOff: flow control using Xon and Xoff characters. For applications requiring XOn and XOff characters other than ^Q and ^S respectively, this parameter may be supplied as a list in the form: (1 <XOn> <XOff>), where <XOn> and <XOff> represent the character codes (ASCII 1 - 127) of the characters which are to be treated as the XOn and XOff characters. The leading "1" signifies that flow control should be enabled; a leading "0" will program the RS232 port with the appropriate XOn and XOff characters, but leave flow control disabled. |
| ModemControl      | This parameter should be a list of modem control signals to be enabled, such as DSR, CTS, DTR, DTR, RI and HS.<br><br>The functions RS232MODEMCONTROL, RS232MODEMSTATUSP, and RS232MODIFYMODEMCONTROL provide finer control over the settings of modem signals (see below).                                                                                                                                                                                                                                                                                                                                                      |
| DTR               | This parameter enables or disables the data terminal ready signal; it may be specified as T or NIL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| RTS               | This parameter enables or disables the request to send signal; it may be specified as T or NIL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

(RS232C.GET.PARAMETERS *PARAMETERLIST*) [Function]

The current settings for the RS232 port may be obtained at any time by calling this function.

*PARAMETERLIST* should be a list of parameter names.  
RS232C.GET.PARAMETERS returns an association list of parameter names and values, in a format acceptable to RS232C.SET.PARAMETERS.

(RS232C.SHUTDOWN) [Function]

The RS232 port is turned off by calling this function. It disables the RS232 port and closes any open streams on the devices.

## Using RS232 Streams

Programs may open streams to the RS232 port by calling `OPENSTREAM` with the file name {RS232}. The *ACCESS* argument to `OPENSTREAM` controls whether an INPUT or OUTPUT stream is returned. RS232 streams are unidirectional; to obtain a second stream open for the opposite access, call the function `RS232C.OTHER.STREAM`.

Only one pair of RS232 streams may be open at a time; an error will result if you attempt to open more.

(`RS232C.OTHER.STREAM` *STREAM*) [Function]

*STREAM* should be an RS232 stream. If *STREAM* is open for INPUT, an RS232 stream open for OUTPUT is returned; conversely, if *STREAM* is open for OUTPUT, an RS232 stream open for INPUT is returned.

The following Lisp functions are defined to work on RS232 streams open for the appropriate access: `BIN`, `BOUT`, `READP`, `OPENP`, `CLOSEP`, and `FORCEOUTPUT`.

(`RS232C.CLOSE-STREAM` *DIRECTION*) [Function]

This function closes one or both RS232 streams, so you don't need to have access to the streams to close them.

*DIRECTION* can be one of INPUT, OUTPUT, BOTH, or NIL. The function closes the RS232 stream open in *DIRECTION* mode; if *DIRECTION* is BOTH or NIL the input and output streams will be closed, if they exist.

RS232 streams are buffered. Input and output are performed in units of packets of data. The I/O processor collects incoming data into a packet, and makes that packet available to Lisp when one of the following conditions is true:

The packet is filled.

The frame time-out has expired.

The frame time-out is the number of hundredths of a second that are allowed to occur between the reception of single characters. This parameter is automatically set by the RS232 module. Its value depends on the baud rate of the RS232 port. If the value is set too large, interactive applications such as Chat will suffer from uneven typeout; if the value is too small, a larger number of shorter packets may be exchanged between Lisp and the I/O processor, resulting in increased processing overhead.

Lisp buffers data for output in packets of up to 578 characters. Output packets are sent to the RS232 port when one of the following conditions is true:

The current output packet is full.

The user program calls the `FORCEOUTPUT` function to force the current output packet to be sent.

The output stream is closed.

Applications that generate a large amount of output slowly may wish to reduce the size of outgoing packets. Although this will require additional processing overhead, it will cause output to occur more frequently without the program explicitly calling `FORCEOUTPUT`.

(RS232C.OUTPUT.PACKET.LENGTH *NEWVALUE*) [Function]

This function returns the current setting of the variable that controls the maximum size of output packets. If *NEWVALUE* is supplied, the setting is changed to *NEWVALUE*. *NEWVALUE* may be a number between 1 and 578.

Specifying a value is a matter of how long you are willing to wait at input time before you are assured of seeing incoming data. You can do a straightforward division to set the length (e.g., at 9600 baud, you get about 1 char/millisecond, so the maximum delay is 578ms; if you can tolerate only 1/4 second, then set it to 250 or so).

(RS232C.READP.EVENT *STREAM*) [Function]

Many RS232 applications are time-dependent. File transfer protocols such as Kermit and Modem depend on one or both sides of a file transfer detecting connection problems by means of time-outs. This function allows user programs to detect time-out conditions efficiently.

*STREAM* should be an RS232 stream open for input. This function returns an event that a program may wait upon for input data to become available on the stream. The following example illustrates how a program could wait up to 10 seconds for a character to become available:

```
[LAMBDA (STREAM)
 (LET ((EVENT (RS232C.READP.EVENT STREAM))
 (TIMER (SETUPTIMER 10000)))
 (until (OR (READP STREAM) (TIMEREXPIRED? TIMER))
 do (AWAIT.EVENT EVENT TIMER T)
 finally (RETURN (COND ((READP STREAM) (BIN STREAM))
```

## Using Modems

The following functions are useful for controlling modems:

(RS232SENDBREAK *EXTRALONG?*) [Function]

This function sends the out-of-band BREAK signal, for a period of 0.25 seconds; if *EXTRALONG?* is non-NIL, then the period is extended to 3.5 seconds.

(RS232MODEMCONTROL *SIGNALSONLST*) [Function]

This function is a Lambda-NoSpread function that sets the modem control lines to be “on” for the signals named in the list *SIGNALSONLST*; it returns the former setting of the lines. If *SIGNALSONLST* is not supplied (which is not the same as supplying NIL), then the control lines remain unchanged. The entries in *SIGNALSONLST* are symbol names for standard modem control lines; currently usable signal names are DTR and RTS.

(RS232MODIFYMODEMCONTROL *SIGNALSONLST SIGNALSOFFLST*) [Function]

Changes only those modem control lines specified in the union of the two arguments; those in *SIGNALSONLST* are set to be on, and those in *SIGNALSOFFLST* are set off. Returns the former state just as (RS232MODEMCONTROL) does.

(RS232MODEMSTATUSP *SPEC*) [Function]

Returns non-null if the reading of the modem status lines is consistent with the boolean form specified by *SPEC*; modem status signals currently supported are

DSR, RI, and RLSD. SPEC may be any AND/OR/NOT combination over these signal names.

Example:

```
(RS232MODEMSTATUSP ' (AND CTS (NOT RLSD))) .
```

```
(RS232MODEMHANGUP)
```

[Function]

This function takes whatever steps are appropriate to cause the modem to hang up. Generally, this means turning the DTR signal down for about three seconds, or until the DSR signal has gone down.

## Error Condition Reporting

The RS232 port detects parity errors, character framing errors, lost characters, and a number of other unusual conditions. As the I/O processor delivers each input packet to Lisp, it reports when the packet was received without error. If an error did occur while the packet was being received, Lisp will report this fact by writing a message to RS232C.ERROR.STREAM.

```
RS232C.ERROR.STREAM
```

[Variable]

RS232 error conditions are reported on this stream. This stream is initially the PROMPTWINDOW.

```
(RS232C.REPORT.STATUS NEWVALUE)
```

[Function]

There are circumstances in which the RS232 hardware believes it has encountered an error, when in fact it has not. A frequent cause is an incorrect parity setting in the RS232 port. Continually reporting RS232 errors is likely to slow RS232 processing severely. In cases where error reporting is not important, it is possible to disable error reports with the RS232C.REPORT.STATUS function. This function returns the current setting of status reporting, which may be one of the following:

T Errors are reported on both input and output.

NIL Errors are never reported.

OUTPUT Errors are reported on output only.

INPUT Errors are reported on input only.

In addition, if *NEWVALUE* is supplied, the current setting of status reporting is changed to *NEWVALUE*.

## RS232TRACE

To help in debugging RS232 applications, it is possible to trace the data that is being sent out via the port. RS232 packets are traced using:

```
(RS232C.TRACE MODE)
```

[Function]

*MODE* is one of PEEK, T, or NIL. If *MODE* is either T or PEEK, RS232C.TRACE opens a trace window in the mode selected. T indicates a full trace, with every byte being shown. PEEK is a less verbose trace, with every incoming packet shown as a "+" and every outgoing packet shown as a "!". NIL turns off the

tracing. Clicking the left mouse button in the trace window will cycle between the modes.



## RS232CHAT

The RS232CHAT module is a facility that permits the Chat library module to communicate over the RS232 port. Once it is loaded, you may chat to the host named RS232 to open a connection to the RS232 port.

## RS232CMENU

RS232CMENU is a utility that provides a menu interface to controlling the settings of the RS232 port. When loaded along with RS232CHAT, this utility may be invoked by choosing the "Set Line Parameters" entry in the options menu that appears if you select "Options" in the middle-button Chat menu.

| RS232 Port Settings |         |            |            |                   |         |
|---------------------|---------|------------|------------|-------------------|---------|
| Apply!              | Abort!  | Show!      | SendBreak! | SendLongBreak!    | Hangup! |
| Baud Rate:          | 9600    | Parity:    | None       | Character Length: | Seven   |
| Flow Control:       | XOnXOff | Stop Bits: | One        | Report Errors:    | Always  |

The following commands are available in the RS232 menu:

- |               |                                                                                                                                                                                      |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Apply         | This menu item changes the RS232 port settings according to the values of the fields in the rest of the menu. No changes are made to the RS232 hardware until this command is given. |
| Abort         | Closes the RS232 menu and aborts any changes that could have been made.                                                                                                              |
| SendBreak     | Sends a normal (0.25 second) break signal. Requires confirmation, as this command could cause the modem connection to be broken.                                                     |
| SendLongBreak | Sends a long (3.5 second) break signal. As above, this requires confirmation.                                                                                                        |
| Hangup        | Tries to hang up the modem (by dropping DTR). Requires confirmation.                                                                                                                 |

The following fields are multiple choice items; when their labels are selected with the mouse, a menu of possible values for the field appears. Choosing a value from the menu will change the field; clicking off the menu will leave the field unchanged.

|                  |                                                             |
|------------------|-------------------------------------------------------------|
| Baud Rate        | Changes the BaudRate of the RS232 port.                     |
| Parity           | Changes the Parity setting of the RS232 port.               |
| Character Length | Changes the BitsPerSerialChar setting of the RS232 port.    |
| Flow Control     | Changes the FlowControl setting of the RS232 port.          |
| Stop Bits        | Changes the NoOfStopBits setting of the RS232 port.         |
| Report Errors    | Changes the RS232C.REPORT.STATUS setting of the RS232 port. |

## File Transfer Using RS232

Files may be transferred using RS232CHAT and either the Kermit or Modem protocols.

## Using the TTY Port

---

Support for the TTY port is contained on the file DLTTY.LCOM. The TTY port is designed to support low-speed communications with RS232 devices. The I/O processor offers no low-level support for input buffering. As a result, it is quite possible for newly received input characters to overwrite previously received but unread characters in the input hardware. The TTY port provides exactly one character's worth of buffering; each character must be read by Lisp before the next character is completely received.

**Note:** No hardware flow control is provided on the TTY port. The Lisp TTY port service routines will obey received flow control commands, but will not generate flow control commands in response to increased input data rate.

(TTY.INIT *BAUDRATE BITSPERSERIALCHAR PARITY NOOFSTOPBITS*  
*FLOWCONTROL*) [Function]

This function is very similar to the function RS232C.INIT.

Before using the TTY port, the TTY hardware must be programmed with the proper characteristics for your application.

Alternatively, the *BAUDRATE* argument can be an instance of the RS232C.INIT record. If *BAUDRATE* is NIL, the value of the global variable TTY.DEFAULT.INIT.INFO is used in its place. This provides a means of automatically initializing the TTY port hardware without user intervention.

TTY.DEFAULT.INIT.INFO [Variable]

This variable controls default initialization of the TTY port. Its value may be set in the site INIT.LISP file, or in your INIT.LISP file. If TTY.DEFAULT.INIT.INFO is not set when the TTY package is loaded, its fields will be set to the following default values:

```

BaudRate: 1200
BitsPerSerialChar: 8
Parity: NONE
NoOfStopBits: 1
FlowControl: XOnXOff

```

Programs may use OPENSTREAM as an alternative to calling TTY . INIT directly, with the parameters bundled up into the PARAMETERS argument as shown below.

For example:

```

(OPENSTREAM ' {TTY} ' BOTH NIL ' ((BaudRate 9600)
(BitsPerSerialChar 8))

```

(TTY . SET . PARAMETERS *PARAMETERLIST*)

[Function]

Applications may change the settings of the TTY hardware while the TTY port is in use.

*PARAMETERLIST* is an association list of parameter names and values. For example, let's set the baud rate to 9600 baud and the character length to eight bits:

```

(TTY . SET . PARAMETERS ' ((BaudRate . 9600) (BitsPerSerialChar .
8)))

```

The following is a list of legal parameter names and values:

|                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BaudRate          | 50, 75, 110, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600, 19200.                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| BitsPerSerialChar | 5, 6, 7, 8. If 5 or 6 bits of data are sent, they should be DATA, not CHARACTERS.                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Parity            | NONE, ODD, EVEN ( 1 parity bit).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| NoOfStopBits      | 1, 2 (This parameter should be 1 except at 110 baud).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| FlowControl       | NIL, XOnXOff. For applications requiring XOn and XOff characters other than ^Q and ^S respectively, this parameter may be supplied as a list in the form: (1 <XOn> <XOff >), where <XOn> and <XOff> are replaced by the character values of the XOn and XOff characters. The leading 1 signifies that flow control should be enabled; a leading 0 will program the TTY port with the appropriate XOn and XOff characters, but leave flow control disabled.<br><br>Note: XOnXOff flow control is known to be reliable only up to 4800 baud. |
| DSR               | This parameter enables or disables the data set ready signal; it may be specified as T or NIL.                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| CTS               | This parameter enables or disables the clear to send signal; it may be specified as T or NIL.                                                                                                                                                                                                                                                                                                                                                                                                                                              |

(TTY.GET.PARAMETERS *PARAMETERLIST*)

[Function]

The current settings for the TTY port may be obtained at any time by calling this function.

*PARAMETERLIST* should be a list of parameter names.

TTY.GET.PARAMETERS returns an association list of parameter names and values, in a format acceptable to TTY.SET.PARAMETERS.

(TTY.SHUTDOWN)

[Function]

This function turns off (disables) the TTY port and closes any open streams on the device.

## Using TTY Streams

Programs may open streams to the TTY port by calling OPENSTREAM with the file name {TTY}. The ACCESS argument to OPENSTREAM may be INPUT, OUTPUT, or BOTH.

Unlike RS232 streams, TTY port streams are not buffered, and a single stream may be used for both input and output. The generic Lisp input/output functions BIN, BOUT, READP, OPENP, and CLOSEP may be used on TTY port streams.

## TTYCHAT

TTYCHAT is a module that enables the Chat library module to communicate over the TTY port. No user-callable functions or user-settable variables are available in the TTYCHAT module. Once it is loaded, you may chat to the host named TTY to open a connection to the TTY port. TTYCHAT is contained on the file TTYCHAT.LCOM.

Because the TTY port does no low-level input buffering, it is quite likely that many input characters will be lost while chatting at 1200 baud or higher. This package should only be used for non-critical applications, such as testing connections between the TTY port and low-speed printers.

## RS232CMENU

RS232CMENU is compatible with the TTY port as well. Certain RS232 port commands, such as SendBreak! are not available with the TTY port, and hence do not appear in the menu.

---

## Examples

### Testing the Connection Between Two Xerox Lisp Machines

To test for a working RS232 connection between Machine A (a Xerox 1108) and Machine B (a Xerox 1100, 1108, or 1186) by moving a file between them, proceed as follows:

1. Load RS232CHAT.LCOM on both machines.
2. Call RS232C.INIT to set up parameters.
3. Do RS232CHAT on both machines. You will be prompted for a window.

Whatever you type on machine A should be echoed on machine B and vice versa.



---

## Testing the Connection Between Xerox Lisp Machines and a VAX Running VMS

VAX side: Set baud rate at which files will be transferred on the VAX side using the VMS command SET. For example, to set to 1200 baud, type:

```
SET TERM TTA1:/SPEED=1200/PERM
```

1108 side: Load RS232CHAT.LCOM

```
Initialize: (RS232C.INIT 1200 8 NIL 1 NIL'RS232C)
```

```
Then call (RS232CHAT)
```

You should be able to use the Chat window like a VAX teletype terminal.

By matching the baud rate on the VAX ( through SET TERM) with that on the 1108 (through RS232C.INIT), you can use any speed up to 9600 baud.

[This page intentionally left blank]

# SAMEDIR

---

SameDir modifies MAKEFILE to guard against inadvertently writing out a file onto a directory other than the one it came from.

SameDir adds the form (CHECKSAMEDIR) to MAKEFILEFORMS. It compares the (HOST&DIRECTORYFIELD OLDFILE) against (DIRECTORYNAME T T) to see whether the connected directory matches the old file's source.

## Installation

---

Load SAMEDIR.LCOM from the library.

## User Interface

---

If you do a MAKEFILE and you are connected to a directory that is not listed in the FILEDATES property of the file, and the file has a FILEDATES property at all (i.e., this isn't a brand new file), the system will prompt you with:

**You haven't loaded or written TORTOISE in your connected directory {server}<user> should I write it out anyway?**

Your options are reply with Y, N, C, or O:

- Y Yes, do the MAKEFILE
- N No, abort the MAKEFILE
- C Connect to other directory: allows you to type in another path.
- O Oops! Connect to the best guess; i.e., the directory where the file was last loaded or written. This option requires confirmation, in case you don't like the directory that the system prompts you with.

The default answer to the question is Y (do the MAKEFILE).

When comparing directory names, SameDir ignores case differences between the old and new directory names.

MIGRATIONS

[Variable]

For those who regularly LOADFROM files on one directory and MAKEFILE elsewhere, the variable MIGRATIONS can be set to keep SameDir from asking too often. It is an association list containing pairs of (OLDDIR . NEWDIR), which specifies which migrations are allowable.

For example, if it is legitimate to LOADFROM a file on {MYHOST}<PUBLIC> and then do a MAKEFILE to {MYHOST}<TEST>, then adding ({MYHOST}<PUBLIC> . {MYHOST}<TEST>) to MIGRATIONS will prevent MAKEFILE from complaining about such movement.

## Limitations

---

For UNIX hosts using the PUP FTP protocol, there is sometimes an inconsistency between the directory name in the full file name and the directory name in `DIRECTORYNAME`. SameDir may have trouble in that case detecting that the directories are the same.

[This page intentionally left blank]

# SPY

---

Spy is a tool to help you make programs run faster by giving you a picture of where the program is spending its time.

## Description

---

Spy has two parts: a sampler and a displayer. The sampler runs while your program is running, and it monitors what your program is doing. The displayer displays the data gathered by the sampler.

The sampler periodically interrupts the running program to account the functions in the current call stack. This allows Spy to remember not only (proportionally) how long is spent in each individual function, but also how long each function is seen on the call stack. The sampler data structures minimize interference with the normal running of the program — there is little noticeable performance degradation. Spy doesn't log every call and return (it only samples), so you can run it even over long computations without fear of overflowing storage limits.

The displayer uses the Grapher module to display the data gathered by the sampler. In the graph, the height of each node is adjusted to be proportional to the amount of time. Just as MasterScope and Browser give an interactive picture of the static structure of the program, Spy gives an interactive picture of the dynamic structure. The displayer is interactive as well as graphic. That is, you can look at the data in a variety of ways, since it seems there is no one picture that says it all. Since the displayer knows the whole call graph, it can show the entire tree structure, with separate calls to a function accounted separately, or merge separate calls to the same functions. Since the sampler records the entire calling stack when it samples, it can account for both individual and cumulative time. When the sampler runs, if a function is on the top of the stack, it adds to its individual total; if the function is on the stack at all, the sampler adds to the cumulative total.

When there are several calls to the same function within the graph, the displayer can either merge the nodes (show the total time for the function in one node) or not. If a node is merged, then one of the boxes in the graph will have all of the time for that function accounted to it, and the rest will be left as ghost boxes. Spy has a variety of ways of controlling which nodes will be merged.

## Requirements

---

GRAPHER  
READNUMBER  
IMAGEOBJ

## Installation

---

Load `SPY.LCOM` and the required `.LCOM` modules from the library.

## User Interface

(SPY.BUTTON *POS*)

[Function]

This function puts up a little window, which you can use to turn Spy on and off. If *POS* is NIL, you can drag the window with the mouse. If *POS* is specified (in the format *xxx . yyy*) then the window is placed at those coordinates.

When Spy isn't watching, it looks like this:



Left-clicking (pressing the left mouse button) on it once turns on sampling, and the window looks like this:



Clicking on it again turns off sampling, and displays the results (SPY.TREE 10). This is the simplest way of spying on operations. (See SPY.TREE below.)

Note: You cannot turn off sampling if the mouse process is locked out.

(SPY.START)

[Function]

Reinitializes the internal Spy data structures, and turns on sampling.

(SPY.END)

[Function]

Turns off sampling, and cleans up the data structures in preparation for the display phase performed by SPY.TREE.

(SPY.TOGGLE)

[Function]

If Spying is off, turn it on with (SPY.START). If it is on, turn it off with (SPY.END) and then show the results with (SPY.TREE 10).

It is reasonable to use this with an interrupt character; e.g.,

```
(INTERRUPTCHAR (CHARCODE ↑C) ' (SPY.TOGGLE) T)
```

enables Control-C (or any other character you specify) as an interrupt which turns spying on and off, the same as clicking on the Spy button. (If the Spy button is visible, it responds to the interrupt.) Then, a Control-C turns on spying, and another one turns it off.

(WITH.SPY FORM)

[Macro]

Calls (SPY.START), evaluates *FORM*, calls (SPY.END), and then returns the value of *FORM*.

For example,

```
(WITH.SPY (LOAD 'FOO))
```

is basically

```
(PROGN (SPY.START) (PROG1 (LOAD 'FOO) (SPY.END]) .
```

```
(SPY.TREE THRESHOLD INDIVIDUALP MERGETYPE DEPTHLIMIT) [Function]
```

SPY.TREE displays the results of the last SPY sampling in a Grapher window. There are a number of parameters that control the display, which you can either set when you call SPY.TREE, or set interactively with the menu. You normally just use (SPY.TREE) and the menus.

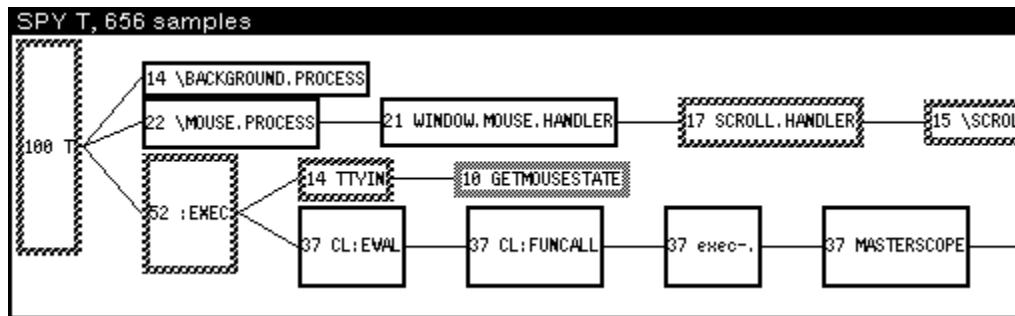
*THRESHOLD* is a percentage (defaults to 0). If a function's contribution to the total elapsed time is lower than the threshold percentage, it is not displayed.

*INDIVIDUALP* is either NIL or T. This controls whether cumulative or individual percentages are displayed. The default is cumulative, in which case a function is charged for the time spent in that function and all subfunctions; if individual, a function is only charged for the time spent in that function alone.

*MERGETYPE* is one of (NONE, ALL, DEFAULT). This controls accounting for functions that appear in several places in the calling tree. Mergetype ALL indicates the total time spent for all calls to the same function, regardless of where it appears. Mergetype NONE indicates the times separately for each instance of the function. Mergetype DEFAULT is the same as NONE except for recursive functions.

*DEPTHLIMIT* is a number (defaults to NIL = arbitrary depth; not completely debugged for other values).

You will get a prompt to open a window, and then a graph will appear in it, something like this:



In this example, 100% of the time was spent under the top frame, T. That time was then divided up among three processes: \BACKGROUND.PROCESS, \MOUSE.PROCESS, :EXEC. The numbers to the left of the label are the percentages. The height of the box is proportional to the percentage (except that it is always made big enough to hold the label). The width isn't significant; it is just wide enough to hold the name of the function

You can point at any of the nodes.



## Right Button Operation

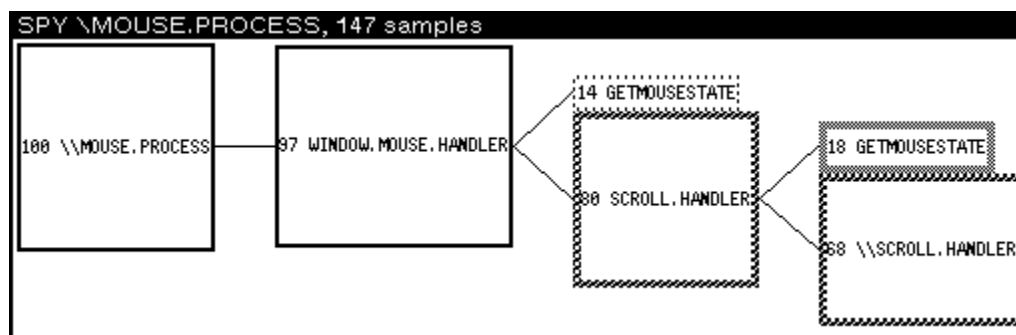
If you right-click on a node, the window title will change to show the function name, and the individual and cumulative percentages. The first number is the individual total and the second is the cumulative. If the right-click is not on a node, the title will change to show the name of the top frame and the total number of samples.

## Left/Middle Button Operations — on a Node

If you left-click, or middle-click on a node, you will get a menu:

```
NewSubTree
SubTree
Delete
Merge
Edit
InspectCode
```

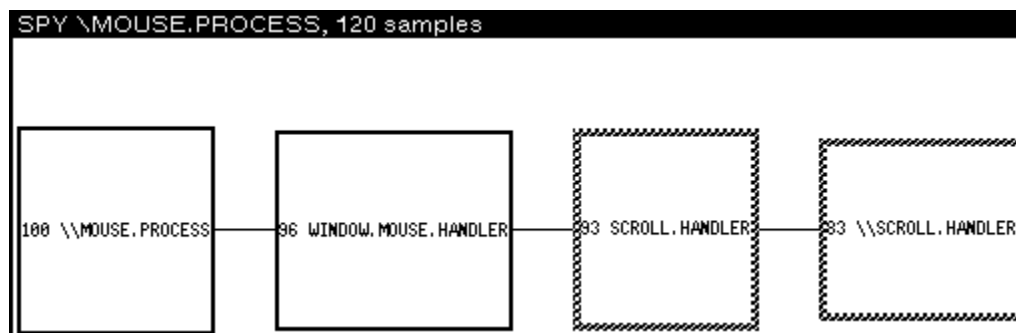
**NewSubTree** Creates another Spy window that includes data only from this node and its descendents (with the tree rooted at the selected node). Suppose that you were only interested in the actions that you had invoked with the mouse. You can left-click `\MOUSE.PROCESS` and select the `NewSubTree` option. You then get a picture like this:



**SubTree** Behaves just like `NewSubTree` except that Spy will reuse the same window.

**Delete** Removes the selected item (and its subbranches) from consideration in this window, and redisplay.

For example, if you don't want to consider `GETMOUSESTATE` in the graph at all, you can delete the `GETMOUSESTATE` box and get:



The percentage for the `SCROLL.HANDLER` changed from 80% to 93% when `GETMOUSESTATE` was deleted.

|             |                                                                    |
|-------------|--------------------------------------------------------------------|
| Merge       | Allows you to merge a node with its caller everywhere in the tree. |
| Edit        | Invokes <code>SEdit</code> for the function in the node.           |
| InspectCode | Shows the compiled code for the function in the node.              |

## Left/Middle Button Operation — Not on a Node

If you press the left, or middle button while not on a node, you get a menu that will let you view or change the parameters for the Spy window:

|                                 |                                                                                                                                                                                                                                                                                      |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Legend                          | Displays SPY border interpretation.                                                                                                                                                                                                                                                  |
| Inspect                         | Allows for the inspection of the current parameter settings for the Spy window.                                                                                                                                                                                                      |
| SetThreshold                    | Sets the threshold for displaying a node. Any node whose percentage is below the threshold won't show up (unless it is needed to connect the graph together). You can set the initial threshold via the threshold argument to <code>SPY.TREE</code> ; otherwise it defaults to zero. |
| Individual/Cumulative           | This is used to toggle between the display of individual and cumulative times. The initial default is cumulative; you can set it to Individual by supplying <code>T</code> as the <code>INDIVIDUALP</code> argument to <code>SPY.TREE</code> (see below).                            |
| MergeNone/MergeAll/MergeDefault | This controls merging of nodes (see below).                                                                                                                                                                                                                                          |

## Ctrl-Left/Ctrl-Middle Button Operations

If you press the left, or middle button while the control key is down, you will get the same menu, but the action will be deferred until you next use the left/middle button. For example, you can delete several nodes and then do one update.

## Merged Nodes

If two nodes are merged, then the merged node will include the time (and descendents) of the other. The display of a merged node is different; it is shown with a thick gray border; e.g. ,

Includes other branches

The time in a merged node is the sum of the times for all occurrences. Other calls to the same function may show up as ghost boxes; e.g.

```
:Shown elsewhere:
```

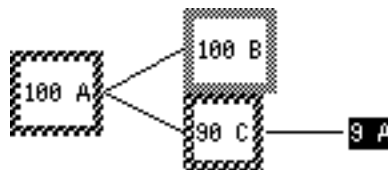
The case where a function is merged with a recursive call to itself is handled specially: the head of the recursion is marked with a wider checkered border:

```
Head of recursive chain
```

and the tail of the recursion is shown in reverse video:

```
End of recursive chain
```

In the recursive case, you can find a situation like this:



In this case, A calls B and C, and C calls A. All of the time is really spent in B, although only 10% is due to the call from the top-level, and 90% is under a call to C, which called A, which called B. In this situation, C is also recursive, of course, and also has the recursive border. If you find this display confusing, try the MERGENONE option and see if you get a clearer picture.

MERGEDEFAULT means to merge any function that is not in SPY.NOMERGEFNS, initially set to (SI : : \*UNWIND-PROTECT\* CL:EVAL \EVAL-PROGN \INTERPRET-ARGUMENTS \INTERPRETER \INTERPRETER1 ERRORSET \EVAL \EVALFORM APPLY \PROGV EVAL).

MERGENONE means not to merge at all.

MERGEALL means to merge any two nodes for the same function.

The default for Individual mode is MERGEALL. The default for Cumulative mode is MERGEDEFAULT.

## Individual and Cumulative Modes

Spy initially comes up with the height of the boxes showing the amount of time the function was on the current call stack. This is called cumulative mode, since each function gets the time that both it and the functions it calls account for. There is another kind of display, called Individual, in which the boxes are proportional to the amount of time the function was on the top of the stack.

One thing to watch for: when you switch between Individual and Cumulative modes, the threshold stays the same. Sometimes the threshold for Individual needs to be higher; otherwise, functions will tend to disappear in the Individual tree. Also, switching to Individual mode also changes to MERGEALL, while switching to Cumulative changes you to MERGEDEFAULT.

(SPY.LEGEND) [Function]

If you forget what the different shadings and borders mean, this function brings up a window that shows what they mean; i.e., it shows the interpretation of SPY.BORDERS or the other internal controls.

SPY.FREQUENCY [Variable]

How many times per second to sample? Initially set to 10. (Maximum 60).

SPY.NOMERGEFNS [Variable]

Functions on this list won't get merged under MergeDefault. Includes (SI::\*UNWIND-PROTECT\* CL:EVAL \\EVAL-PROGN \\INTERPRET-ARGUMENTS \\INTERPRETER \\INTERPRETER1 ERRORSET \\EVAL \\EVALFORM APPLY \\PROGV EVAL). You may need to add more.

SPY.TREE [Variable]

This variable (same name as the function) is used to hold the data from the last sampling. You can save it and restore it using UGLYVARS (see *IRM*).

SPY.BORDERS [Variable]

Used to control the border display on a tree. This is a list of (*NODETYPE DESCRIPTION BORDERWIDTH TEXTURE INTERIORTEXTURE*).

SPY.FONT [Variable]

Font used to display node labels. Initially GACHA 10.

SPY.MAXLINES [Variable]

Maximum height of a node in the graph, measured in multiples of the font height of SPY.FONT.

## Limitations

Spy doesn't know anything about the interpreter or the internal workings of Lisp. Internal functions that are not REALFRAMEP and don't normally show up on BT backtraces (but do on BT!) will be shown in Spy. This includes things like \INTERPRETER1, which will appear underneath any interpreted function call. Thus Spy does not distinguish between frames that are interesting or not interesting to the user.

[This page intentionally left blank]

# **SYSEDIT**

---

There are many system-internal data type declarations that don't usually appear in the sysout. For example, most people don't use the internal fields of strings or streams, so their definitions aren't included in the sysout. However, there is a collection of definitions that are used by people who are working on system-level code, definitions that are widely relied upon by more than one system source file.

SysEdit is provided for users who work with the system sources or who write system-level code — it brings in the frequently-used definitions.

## **Requirements**

---

MASTERSCOPE  
EXPORTS.ALL  
CMLARRAY-SUPPORT

## **Installation**

---

Lisp programming environment.

## **Definitions**

---

The declarations and definitions are provided in the source listings of SysEdit and in the files it loads.

## **Limitations**

---

SEdit internal definitions are not included, nor are declarations that are used only within a single system source file.

[This page intentionally left blank]

# TABLEBROWSER

---

TableBrowser implements a simple mechanism for building applications that browse certain kinds of tabular data. It supplies a set of basic functions that maintain the window, allowing scrolling and selection of items; the application defines the items, how they print, and any higher-level operations to perform on them. FileBrowser is an example of an application built upon TableBrowser.

## Installation

---

Load TABLEBROWSER.LCOM from the library.

During program development, you also need to load the declarations file TABLEBROWSERDECLS; this file is not needed when running a compiled application. The file manager coms for the typical application file should thus include the two commands

```
(FILES (SYSLOAD) TABLEBROWSER)
(DECLARE: EVAL@COMPILE DONTCOPY
 (FILES (SOURCE) TABLEBROWSERDECLS))
```

## User Interface

---

An instance of a TableBrowser application is a window displaying a browser. The browser consists of an ordered set of items. Each item contains an item of application data and some bookkeeping information. TableBrowser maintains the display, supplying generic scrolling, painting, reshaping and selection mechanisms; the application supplies TableBrowser with a set of methods for displaying the contents of an item, what to do when an item is copy selected, etc.

Review the user interface of the FileBrowser module to get an idea of the selection mechanism and the sort of functionality available from TableBrowser. Briefly, items are selected by clicking in the browser window — left to select a single item, middle to add an item to the current selection, control-middle to remove an item, right to extend the selection, control-right to extend the selection, including deleted items.

The typical application creates a browser window, fills it with items, and attaches one or more menus to the window. The menu contains commands that may perform application-specific operations, usually on the set of currently selected items.

## Records

---

There are two important records, TABLEITEM and TABLEBROWSER. These records are declared in the file TABLEBROWSERDECLS.

### TABLEITEM Record

Each piece of application data (a line in the browser) is encapsulated in an instance of the datatype TABLEITEM, whose fields are as follows. The first four fields are supplied by the application; the others are maintained by TableBrowser and should be considered read-only from the point of view of the application:



|                                                                                                                                                                                                                      |                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| TIDATA                                                                                                                                                                                                               | [Record field] |
| An arbitrary pointer to the application data.                                                                                                                                                                        |                |
| TIUNSELECTABLE                                                                                                                                                                                                       | [Record field] |
| If T, the user is not permitted to select this item.                                                                                                                                                                 |                |
| TIUNCOPYSELECTABLE                                                                                                                                                                                                   | [Record field] |
| If T, the user is not permitted to copy-select this item.                                                                                                                                                            |                |
| TIUNDELETABLE                                                                                                                                                                                                        | [Record field] |
| If T, the user is not permitted to delete this item. This field is not currently supported by TableBrowser, though the application may use it itself for this purpose.                                               |                |
| TI#                                                                                                                                                                                                                  | [Record field] |
| An integer denoting the position of this among the set of items in the browser. This field is maintained automatically by TableBrowser as items are removed or new items are inserted. The first item is numbered 1. |                |
| TISELECTED                                                                                                                                                                                                           | [Record field] |
| T if the item is currently selected; NIL otherwise.                                                                                                                                                                  |                |
| TIDELETED                                                                                                                                                                                                            | [Record field] |
| T if the item is currently marked for deletion; NIL otherwise.                                                                                                                                                       |                |

### TABLEBROWSER Record

For each browser, there is an instance of the datatype TABLEBROWSER, whose fields are described below. TABLEBROWSERS are created by TB.MAKE.BROWSER (see the section "Functions," below). The first six fields listed here are "methods"— application functions called from the TableBrowser code when some event occurs or in order to carry out some operation. The application function is called with the browser object itself as the first argument, and possibly other arguments. Only the first method, TBPRINTFN, is required; the others may be NIL. For most applications, the fields listed here are set in the call to TB.MAKE.BROWSER and then never changed.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| TBPRINTFN                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | [Record field] |
| Application function invoked to print a single item in the browser. The three arguments are the browser, the item being painted, and the window in which the item is to be painted. At the time the method is called, TableBrowser has set the x,y position of the window to the left edge of the line where this item starts. It has done nothing to the line, including clear it; the application must clear the line before printing if it so desires. Somewhat smoother, but more complex, repainting is possible if the application clears only the areas that it is not otherwise overwriting. After the printing is finished, TableBrowser supplies a selection mark and deletion line if appropriate. TableBrowser worries about scrolling, reshaping, inserting new items, etc., and calls the printing function for each line that it has determined needs to be (re)displayed as the result of operations performed on the browser. |                |

TBCOPYFN

[Record field]

Application function invoked when an item in the browser is copy-selected. Copy selection occurs when the Copy or Shift key is held down and the mouse is used to select a line in the browser. While the Copy key is down, TableBrowser highlights the item under the cursor with a dotted underline; when the copy key is released, the highlighting is removed, and the copy function is called with two arguments: the browser and the item selected. The Copy function typically calls BKSYSBUF on some string or structure representing the item selected. Copy selection can be aborted by moving the mouse outside the window with the mouse button still down. If an item's TIUNCOPYSELECTABLE field is true, copy selection is ignored while the mouse is inside the item. If TBCOPYFN is NIL, then copy selection is ignored for the entire browser.

TBCLOSEFN

[Record field]

Application function invoked when the user tries to close or shrink the browser window. The three arguments are the browser, the window, and a flag whose value is one of the symbols CLOSE or SHRINK. If the function returns the symbol DON'T, then the close or shrink operation is aborted; if it returns NIL, it proceeds; otherwise, the value is a function to run as a separate cleanup process. In this last case, the window remains open and the value returned from the TBCLOSEFN application is called in a new process with the same three arguments (browser, window, flag). When this function finishes, it should call the function TB.FINISH.CLOSE (see the section "Functions," below) to complete the close or shrink operation, assuming it wishes it to proceed.

TBAFTERCLOSEFN

[Record field]

Application function invoked when the browser window is about to be discarded. The two arguments are the browser and the window. This function is called *after* the TBCLOSEFN, if any, has permitted the window to close; note that it is not called when a window is merely shrunk. The application might, for example, want to remove the browser from its own structures, snap circular links, etc. The return value is ignored.

TBTITLEEVENTFN

[Record field]

Application function invoked when the middle mouse button is pressed while the cursor is in the title bar of the browser (clicking in the body of the window is for selection and is handled by TableBrowser itself). The two arguments are the window and the browser. Note that this is the only method that does not take the browser as the first argument; this is to match the form of window system button event functions.

TBFONTCHANGEFN

[Record field]

Application function invoked when the font of the window is changed (by TB.SET.FONT). The two arguments are the browser and the window. The application function might, for example, want to change cached information about the size of the font.

TBLINETHICKNESS

[Record field]

The thickness of the horizontal lines drawn through deleted items. This defaults to the value of TB.DELETEDLINEHEIGHT, initially 1. For example, setting this field to the height of an item would result in items being completely blacked out when they are deleted.

TBHEADINGWINDOW

[Record field]

An optional auxiliary window that is to be horizontally scrolled in parallel with the main window. The `WIDTH` of the window's `EXTENT` property is maintained in synch with that of the main window, and whenever the main window is horizontally scrolled, the heading window is scrolled by the same amount. You still need to create this auxiliary window, attach it where you want it and supply it with a `REPAINTFN`. (This is how `FileBrowser` implements its header line identifying the columns of attributes.)

TBUSERDATA

[Record field]

An arbitrary pointer to application-dependent data.

The following fields describe the size and shape of items. The values are usually derived from information supplied when a browser is created (see `TB.MAKE.BROWSER`) and then never changed. If an application wishes to change any of these fields once a browser has been created, it should call `TB.SET.FONT` afterwards to notify `TableBrowser` of the change.

TBFONT

[Record field]

A font descriptor, the default font in which items are painted.

TBFONTHEIGHT

[Record field]

The height of the font.

TB#LINESPERITEM

[Record field]

The number of lines (in units of the font's height) occupied by each item. `TableBrowser` requires that each item occupy the same number of lines. The default is 1. For multi-line items, the window is positioned at the first line when its print function is called, selection markers point at the first line, and deletion lines are drawn only through the first line.

TBITEMHEIGHT

[Record field]

The total height of an item. This is normally the font height times the number of lines per item, but an application can set it explicitly, independent of the font, instead of specifying the number of lines per item.

TBBASELINE

[Record field]

The distance of an item's baseline above the bottom of the item. This field has been renamed from `TBFONTDESCENT`. This field is used for two purposes:

- When the browser's `PRINTFN` is called, the y-position of the window is set to be at the baseline.
- Selection marks and deletion lines are centered between the baseline and the top of the item.

The following fields are maintained by `TableBrowser`, but may be of use to application code; they should be considered read-only.

TBWINDOW

[Record field]

A pointer to the window containing the browser.

TBLOCK

[Record field]

A monitor lock acquired by the TableBrowser when performing operations on the browser. Application code may want to hold this lock while performing a series of TableBrowser operations that it wishes to have occur atomically. Selection and scrolling are inhibited while this lock is busy.

## Functions

This section describes the functions that create and manipulate TableBrowser windows and their contents. Typical use for most of these functions is from the code invoked by commands from a menu attached to the window by the application.

### Creating a TableBrowser

(TB.MAKE.BROWSER *ITEMS* *WINDOWSPEC* *PROPS*)

[Function]

Creates a new browser, browsing *ITEMS*, a list of TABLEITEM records. *ITEMS* may be NIL, in which case an empty browser is created. It is permissible for the TISELECTED and/or TIDELETED fields to be true in any item; this is a way of setting the initial selection, and/or pre-deleting some items.

If *WINDOWSPEC* is supplied, it is a window or region; otherwise, the user is prompted for a window.

*PROPS* is a list in property list format specifying initial values for some features of the browser. The properties PRINTFN, COPYFN, CLOSEFN, AFTERCLOSEFN, LINESPERITEM, ITEMHEIGHT, BASELINE, HEADINGWINDOW, LINETHICKNESS, and USERDATA set the corresponding fields of the new TABLEBROWSER record (see above). The following additional properties are recognized:

**TITLE** The title to put on the window. If NIL, the window will not be given a title.

**FONT** The font in which to paint items (a font descriptor or any other argument acceptable to FONTCREATE). This font is made the window's current font. If the browser's print function displays items in more than one font, the application should choose the tallest font for the FONT property, and it is responsible for ensuring that the correct font is always in use. If this property is NIL, the default display font is used. The browser fields TBFONT and TBFONTHEIGHT are set from this font. In addition, if the caller did not specify the ITEMHEIGHT property, the fields TBITEMHEIGHT and TBBASELINE are calculated from the font. TableBrowser uses the sizes to know where each line starts.

If the caller specified the ITEMHEIGHT property but no BASELINE, the baseline is taken to be zero. If the caller did not specify ITEMHEIGHT, then the baseline is calculated to be the font's descent, or in the case of multiple lines per item, the descent plus the font height times

(#LINESPERITEM-1), so that the behavior described above for TB#LINESPERITEM holds.

TB.MAKE.BROWSER returns a new TABLEBROWSER object describing the browser. The application is free to attach further windows to this one. The TABLEBROWSER object is also stored on the window's TABLEBROWSER property.

(TB.REPLACE.ITEMS *BROWSER NEWITEMS*) [Function]

Completely replaces the items of *BROWSER* with *NEWITEMS*, a list of TABLEITEM records. This is a lot like creating a new browser, except that the window structure is already there.

(TB.SET.FONT *BROWSER FONT*) [Function]

Changes *BROWSER*'s display font to *FONT*, which is of the same form as the FONT property given to TB.MAKE.BROWSER. If FONT is NIL, TB.SET.FONT makes its computations based on the browser's current display-related fields (TBFONT, TB#LINESPERITEM, etc).

TB.SET.FONT clears the window; the application is responsible for ensuring that the window is redisplayed, e.g., by calling REDISPLAYW. TB.SET.FONT does not do the redisplay itself, so as to avoid double redisplay in the case where the application also wants to change the items at the same time (by calling TB.REPLACE.ITEMS).

(TB.FINISH.CLOSE *BROWSER WINDOW CLOSEFLG* —) [Function]

Takes care of closing or shrinking *WINDOW*, occupied by *BROWSER*, after the cleanup performed by the browser's TBCLOSEFN (see above).

*CLOSEFLG* is one of the symbols CLOSE or SHRINK.

(TB.BROWSER.BUSY *BROWSER*) [Function]

Briefly changes the cursor to a large "X", the conventional way TableBrowser indicates that an operation attempted with the mouse cannot be performed because the browser is busy.

## Simple Item Operations

(TB.SELECT.ITEM *BROWSER ITEM*) [Function]

Marks *ITEM* in *BROWSER* selected. Ordinarily, selection occurs by means of the mouse. However, applications may want to programmatically select items in response to a user command. Selected items are indicated by a small triangle in the left margin.

(TB.UNSELECT.ITEM *BROWSER ITEM*) [Function]

Marks *ITEM* in *BROWSER* not selected.

(TB.UNSELECT.ALL.ITEMS *BROWSER*) [Function]

Marks *all* items in *BROWSER* not selected. This is considerably faster than unselecting items one at a time. The typical use for this function is prior to calling TB.SELECT.ITEM, to ensure that the new selection is the browser's *only* selection.

(TB.DELETE.ITEM *BROWSER ITEM*) [Function]

Marks *ITEM* in *BROWSER* for deletion. The display shows a line drawn through the item. It is permissible to delete a deleted item (it is a no-op).

(TB.UNDELETE.ITEM *BROWSER ITEM*) [Function]

Removes the deletion mark from *ITEM* in *BROWSER*.

(TB.INSERT.ITEM *BROWSER NEWITEM BEFOREITEM*) [Function]

Adds *NEWITEM* to *BROWSER*'s set of items, inserting it immediately before the item *BEFOREITEM*, or at the end if *BEFOREITEM* is NIL.

(TB.REMOVE.ITEM *BROWSER ITEM*) [Function]

Removes *ITEM* from *BROWSER*'s set of items. This is the operation that an application's "Expunge" function would typically use, but it need not be in any way correlated with deleted items.

(TB.NORMALIZE.ITEM *BROWSER ITEM*) [Function]

Scrolls *BROWSER*'s window, if necessary, so that *ITEM* is visible.

(TB.CLEAR.LINE *BROWSER ITEM LEFT WIDTH*) [Function]

Clears the contents of *ITEM*'s line in *BROWSER*, starting at the x-position *LEFT* and clearing a region *WIDTH* pixels wide. *LEFT* defaults to zero, *WIDTH* to infinity, so omitting both clears the whole line.

This function is typically used by a browser's print function to clear the line before displaying fresh contents. An application wanting to print with minimal visual disruption may want to use TB.CLEAR.LINE only on those portions of the line not being printed to explicitly, so that repainting a line with only slight changes minimizes the apparent display activity.

(TB.REDISPLAY.ITEMS *BROWSER FIRSTITEM LASTITEM*) [Function]

Ordinarily, TableBrowser takes care of deciding when items need to be redisplayed (e.g., when scrolling, reshaping, inserting, etc.). However, there may be times when circumstances beyond the knowledge of TableBrowser require that an item be redisplayed, e.g., when the contents of the item are changed by the application. In such cases, the application is responsible for telling TableBrowser that repainting is needed.

TB.REDISPLAY.ITEMS explicitly invokes *BROWSER*'s repaint method to redisplay items *FIRSTITEM* through *LASTITEM*, which may be the same item in the case of redisplaying a single item. The item arguments may be given as TABLEITEM objects or as a number. *FIRSTITEM* defaults to the browser's first item and *LASTITEM* defaults to the last one; thus (TB.REDISPLAY.ITEMS *BROWSER*) forces redisplay of the entire browser, and is equivalent to calling REDISPLAYW on the window. Only those items currently visible in the window are actually repainted.

## Operations on Multiple Items

(TB.NUMBER.OF.ITEMS *BROWSER TYPE*) [Function]

Returns the number of items in *BROWSER* of the specified *TYPE*, one of the following symbols:

NIL Returns the total number of items.

SELECTED Returns the number of items currently selected.

DELETED Returns the number of items currently deleted.

(TB.NTH.ITEM *BROWSER* *N*) [Function]

Returns the *N*th item in *BROWSER*. *N* is an integer; the first item is numbered 1. Returns NIL if *N* is less than 1 or greater than the number of items in the browser.

Most of the following functions accept a predicate or mapping function. The results of the mapping are unpredictable if the mapping function adds or removes items, so an application wishing to do so should first collect the items of interest and map over that list itself.

(TB.COLLECT.ITEMS *BROWSER* *PREFN*) [Function]

Returns a list, in browser order, of all the items in *BROWSER* of the type specified by *PREFN*. *PREFN* can be one of the symbols NIL, SELECTED or DELETED, which are interpreted as for TB.NUMBER.OF.ITEMS. Otherwise, *PREFN* is a predicate function of two arguments, *BROWSER* and an item from the browser; *PREFN* should return T if the item is to be collected.

(TB.MAP.ITEMS *BROWSER* *MAPFN* *NULLFN*) [Function]

Applies the function *MAPFN* successively to each item in *BROWSER*. *MAPFN* should accept two arguments, *BROWSER* and the item.

If the browser is empty, TB.MAP.ITEMS instead calls *NULLFN*, if specified, with the single argument *BROWSER*.

(TB.MAP.SELECTED.ITEMS *BROWSER* *MAPFN* *NULLFN*) [Function]

Applies the function *MAPFN* successively to each selected item in *BROWSER*. *MAPFN* should accept two arguments, *BROWSER* and the item.

If no items are currently selected in the browser, TB.MAP.SELECTED.ITEMS instead calls *NULLFN*, if specified, with the single argument *BROWSER*.

A typical application calls TB.MAP.SELECTED.ITEMS in response to a menu selection to carry out the operation on the items you have selected.

(TB.MAP.DELETED.ITEMS *BROWSER* *MAPFN* *NULLFN*) [Function]

Applies the function *MAPFN* successively to each deleted item in *BROWSER*. *MAPFN* should accept two arguments, *BROWSER* and the item.

If no items are currently deleted in the browser, TB.MAP.DELETED.ITEMS instead calls *NULLFN*, if specified, with the single argument *BROWSER*.

(TB.FIND.ITEM *BROWSER* *PREFN* *FIRST#* *LAST#* *BACKWARDSFLG*) [Function]

Returns the first item in *BROWSER* in the range of items numbered *FIRST#* through *LAST#* that satisfies the predicate *PREFN* (a function of two arguments), *BROWSER* and the item.

*PREFN* can also be one of the symbols SELECTED or DELETED to search for selected or deleted items.

*FIRST#* defaults to 1, *LAST#* defaults to the number of items in the browser, so omitting both searches the whole browser.

If *BACKWARDSFLG* is true, the range is searched in reverse order.

## Miscellaneous Access Functions

These provide functional access to some of the fields defined in the Records section.

(TB.ITEM.SELECTED? *BROWSER ITEM*) [Function]

Returns T if *ITEM* in *BROWSER* is selected.

(TB.ITEM.DELETED? *BROWSER ITEM*) [Function]

Returns T if *ITEM* in *BROWSER* is deleted.

(TB.WINDOW *BROWSER*) [Function]

Returns a pointer to the window containing the browser.

(TB.USERDATA *BROWSER NEWVALUE*) [Function]

Returns the value of the TBUSERDATA field of the browser; if *NEWVALUE* is supplied, it is stored as the new value of this field.

TB.LEFT.MARGIN [Constant]

The left margin, in pixels, of the start of each item in the browser. Space to the left of this point is used by the selection marker. This constant is compiled into TableBrowser.

## Limitations

In the current implementation, the items in a browser are maintained as a simple list. This means that some operations that might be expected to take constant time (e.g., returning the *n*th item) instead take linear time (or worse). TableBrowser currently optimizes its operations for sequential access, so that the most typical operations are not adversely affected. However, note that performance may not be acceptable for very large browsers (on the order of 1000 items) when accessing items in random order, or in particular, searching a browser in reverse order.



[This page intentionally left blank]

# TEXEC

---

TExec is a version of the Interlisp-D executive which includes certain features of TEdit, so that commands can be edited, much like text. TExec preserves all of the functionality of the "old" executive (including history commands, ?=, DWIM, Programmer's Assistant, editing of the current input form, parenthesis matching/blinking, etc.) plus the ability to scroll anywhere in the output for viewing and/or copy-selecting old text.

TExec makes it easy to use Interlisp to get information, then use that information to build new commands to Interlisp.

TExec has two major advantages:

You can put into a window something longer than a windowful and still be able to scroll back and forth in it. In the regular exec window, all you see are the last few lines.

You can print something to the window, then use all or part of it at your next type-in.

## Requirements

---

TEdit

## Installation

---

Load TEDIT.LCOM and TEXEC.LCOM modules from the library.

## User Interface

---

The Executive is described in the *IRM* and in the *Lisp Release Notes*.

TEdit is described in the *Lisp Documentation Tools* manual.

### Starting TExec

TExec can be invoked interactively from the right-button (background) menu, or programmatically by calling

(TEXEC *REGION* *PROMPT* *MENUFN*) [Function]

If *REGION* is not specified, the system issues a prompt to create a window. If prompt is not supplied, a # is used as the prompt.

If *MENUFN* is not supplied, a command menu similar to TEdit is used. See the TEdit section in the *Lisp Documentation Tools* manual titled "Using the TEdit Window."

## Differences between TExec and TEdit

The following TEdit commands are not included in the TExec main menu: LOOKS, SUBSTITUTE, QUIT, and EXPANDED MENU.

TExec has two Find commands which are not in TEdit: FORWARD FIND and BACKWARD FIND.

FORWARD FIND searches forward from the beginning of the text stream if no previous text string has been found or if the caret is in the current/next type-in; otherwise the search continues forward from the last find.

BACKWARD FIND searches backwards from the type-in point if it is the first time, or from the last place it found the text. You can force BACKWARD FIND to start from the type-in point by placing the caret there with the mouse.

To allow the easy copy-selection of entire lines of input, use a carriage-return/line feed as the prompt, and the prompt will be printed on a different line from the type-in; e.g., (TEXEC REGION "<CR><LF>").

Pressing the escape key does not cause recognition of keywords in USERWORDS as it does under TTYIN. The ^R (retype input) and case-changing commands of TTYIN are not implemented. Display stream graphics are not saved in the output.

## Using TExec

TExec allows editing the current type-in using TEdit commands (see the TEdit section in the *Lisp Documentation Tools* manual titled "Editing Text"). Type-in is considered editable until a final matching right parenthesis, right bracket, or carriage return is typed, at which point it becomes immutable. Any output to a TExec window such as from CONTROL-T or ?= is placed in front of the current type-in so as not to interfere with your typing.

Unechoed input mode is implemented using a feature of TEdit known as invisible characters. Such characters, though invisible, are present in the buffer, and will be copied if they are within the bounds of a copy-selection. The primary terminal table, \PRINTERMTABLE (the value of (GETTERMTABLE)) is used (different from TEdit) to allow control characters to be echoed as CONTROL-X (where x is the control character), as they are in the Old Interlisp Executive.

The contents of a TExec window are saved in memory as a text stream. The maximum number of characters to be saved is specified by selecting the LIMIT command in the menu. When this limit is reached, characters are deleted from the beginning of the buffer as new ones are added to the end. The initial setting is 10,000 characters.

The escape key works the way it is described in the Programmer's Assistant section of the *IRM*. It is used as a character substitution mark by the Programmer's Assistant USE command.

---

## Limitations

TExec does not understand Common Lisp syntax, so it is best to call it from an Interlisp exec.

=? is not implemented.

[This page intentionally left blank]

# TEXTMODULES

---

TEXTMODULES converts source code files from File Manager format to Common Lisp style plain text and back again. When exporting to plain text, a small number of File Manager coms types are supported. When importing from a plain text file, several convenience features are available including comment upgrade and conversion of specific named defmacros into defdefiners.

Note: The Text File Translator changes source code format only; this is **not** an Interlisp to Common Lisp translator.

All symbols described in this section are in the TEXTMODULES package, nicknamed TM.

This section describes the load and make processes, and the static format of text files and their File Manager counterparts. The File Manager counterparts are discussed in increasing detail until their programmability is covered.

## Overview

---

The Text File Translator supports the development of portable Common Lisp source code in the Lisp Environment. It brings portable Common Lisp sources into the File Manager without losing any of their contents. It also makes new text files based on the File Manager's "filecoms."

The original file's function and ordering are retained, but exact formatting is not. The pretty printer causes all comments and expressions on the text file to be uniformly formatted.

Exporting a source file into text and back again will lose grouping of definitions under their coms.

## Installation

---

Load TEXTMODULES.DFASL from the library.

## Dependencies

---

Special support for editing and printing of comments is required. This are provided by the SEDIT-COMMONLISP file. Some caveats on the editing of presentations are mentioned below.

The support file is automatically loaded by the TEXTMODULES file. SEDIT-COMMONLISP cannot operate without TEXTMODULES and must be loaded by it or after it.

File Manager source files created with **load-textmodule** depend on having TEXTMODULES and SEDIT-COMMONLISP loaded.

## Programmer's Interface

---

`load-textmodule` *pathname* &*key* *module* *install* *package* *upgrade-comment-length*  
*join-comments* *convert-loaded-files* *defdefiner-macros*

[Function]

Like **lisp:load**; the file indicated by *pathname* is loaded, but in addition filecoms are created and other information is stored for the File Manager. Key arguments are described below.

(See below under **Text File Format** for a description of the format of text files which can be read by **load-textmodule**).

Local bindings of reader affecting variables are established and set to Common Lisp defaults, except for the readtable.

A special readtable is used which creates internal representations for objects normally lost during reading (see below under **Presentation types**).

If there are some simple forms to set up the read environment at the front of the file, they are recognized and moved into a newly created makefile environment (see below under **Makefile Environment** for a complete description of this).

Each form is read from the file (one at a time). If the form is recognized a description of it is given to the File Manager and its definition is installed. If the form is not recognized it is wrapped in a "top level form" filecom and then installed by stripping presentation objects and evaluating.

**defun** in a **let** at top-level is treated like any top-level form. Such forms should be edited directly in the filecoms. Not doing this can have curious consequences, since calling **ed** on the function name will not modify the definition in the **let** (which remains in the FILECOMS as a top level form).

No forms after the read environment forms should change the reader's environment.

When the file has been completely read its content description is given to the File Manager. Also added to the content description are properties declaring its `il:filetype` as `:compile-file` and `makefile-environment` as that of the text file (whether given by setting forms at the front of the file or by default).

Several key options are available:

*module* A string or symbol used to create the symbol used as the File Manager's name of this module. Strings have their case preserved. Symbols have their name strings taken. Defaults to the uppercased root name of the path.

*install* T or NIL. Indicates whether the definitions in the file should be installed in the running system. Any package setup makes it mandatory to install the definitions in a source file; e. g. since `:INSTALL NIL` means forms in the file are not evaluated, any `IN-PACKAGE` form would not be evaluated and the file would be read in the wrong package. This can sometimes be worked around using the `:PACKAGE` argument.

*package* A package name or package, defaults to USER. This is the package the file will be read into.

- upgrade-comment-length* A number or NIL. Defaults to the value of *\*upgrade-comment-length\** (which defaults to 40). The length, in characters, at which single semicolon comments are upgraded to double semicolon comments.
- join-comments* T or NIL. Defaults to the value of *\*join-comments\** (which defaults to T). Causes comments of the same level in the coms to be joined together. This makes for more efficient editor operation, but loses all formatting inside of comments; e.g. inter-comment line breaks are not preserved.
- convert-loaded-files* T, :QUERY or NIL. Defaults to the value of *\*convert-loaded-files\** (which defaults to :QUERY). If a REQUIRE or LOAD statement is noticed at top level a recursive call to LOAD-TEXTMODULE will be made. With :QUERY turned on the user is first prompted. If the pathname specified in the LOAD or REQUIRE is computed based on variables in the file being loaded :INSTALL must be true. Complex systems that contain special loading functions will not be handled by this mechanism.
- defdefiner-macros* A list of defmacro names. Defaults to the value of *\*defdefiner-macros\** (which defaults to NIL). If a top-level defmacro is found whose name is on this list, the defmacro will be translated into an IL:FUNCTIONS defdefiner form. The defdefiner form then creates a macro that builds definers. Definners are the basic definition units maintained by the File Manager. DEFUN is itself a defdefiner macro. A particular DEFUN form is a definer for the named function (see the *Lisp Release Notes*, 4. Changes to Interlisp-D in Lyric/Medley, Section 17.8.2 Defining New File Manager Types, for more information on the defdefiner form).

Warning: Names on this list must be in the correct package, i.e. the one the file will be read in. A typical way to use this feature is:

- Examine the text source file for DEFMACRO forms that are used to create defdefinners.
- Make the package which the text file's IN-PACKAGE expression will later find.
- Do LOAD-TEXTMODULES, giving the :DEFDEFINER-MACROS key argument a list of fully package qualified symbols naming the defdefinners contained in the file.

*make-textmodule* *module* &key *type* *pathname* *filecoms* *width* [Function]

The File Manager's description of the file *module* is used to create a text file. *module* may be provided as either a string or a symbol. A string's case will be preserved. A symbol's name string is used. Keyword arguments are described below.

(See below under **File manager description of contents** for a description of filecoms that can be written out by **make-textmodule**.)

Local bindings of printer affecting variables are established and set to Common Lisp defaults, except for the readtable.

The file's environment is written out, based on its makefile-environment property (see below under **File manager source file format** for ways of expressing the environment).

The specially made description of the file's contents (from the File Manager) is iterated over to write out each form in the file.

Several key options are available:

|                 |                                                                                                                                                           |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>type</i>     | A string, defaults to ".LISP". The file type extension to be used on the text file being written.                                                         |
| <i>pathname</i> | A pathname, defaults to the module name merged with the extension and <b>*default-pathname-defaults*</b> . The file which will contain the new text file. |
| <i>filecoms</i> | A list of file commands may be supplied here. Defaults to the commands for the File Manager file named by module.                                         |
| <i>width</i>    | A positive integer, defaults to 80. The width, in characters, of lines in the text file. Used by the prettyprinting routines for formatting.              |

## Variables that Control Loading

**\*join-comments\*** [Variable]

T or NIL. Defaults to T. Causes comments of the same level in the file coms to be joined together. This makes for more efficient editor operation, but loses any formatting inside of comments, e.g. inter-comment line breaks are not preserved.

**\*convert-loaded-files\*** [Variable]

NIL, :QUERY or T. Defaults to :QUERY. Controls whether a LOAD or REQUIRE statement at top level in a loaded text file will cause the referred to file to be recursively load-textmodule'd. If the pathname specified in the LOAD or REQUIRE is computed based on variables in the file being loaded the :INSTALL argument to **load-textmodule** must be true.

**\*upgrade-semicolon-comments\*** [Variable]

NIL or a positive integer. Defaults to 40. Controls whether and at what length (in characters) a single semicolon comment is upgraded to a double semicolon comment. NIL inhibits upgrading.

**\*defdefiner-macros\*** [Variable]

A list of defmacro names. Defaults to NIL. If a top-level defmacro is found by LOAD-TEXTMODULES whose name is on this list, the defmacro will be translated into an IL:FUNCTIONS defdefiner form. The defdefiner form creates a macro that builds definers. Definers are the basic definition units maintained by the File Manager. DEFUN is itself a defdefiner macro. A particular DEFUN form is a definer for the named defun (see the *Lisp Release Notes*, 4. Changes to Interlisp-D in Lyric/Medley, Section 17.8.2 Defining New File Manager Types, for more information on the defdefiner form).

Warning: Names on this list must be in the correct package, i.e. the one the file will be read in. A typical way to use this feature is:



- Examine the text source file for DEFMACRO forms that are used to create defdefiners.
- Make the package which the file's IN-PACKAGE expression will later find.
- Do LOAD-TEXTMODULES giving the :DEFDEFINER-MACROS keyword argument a list of fully package qualified symbols naming the defdefiners contained in the file.

## Text File Format

TextModules creates and understands the format of portable pure Common Lisp text files with very simple and constrained package setup information. The overall form of these files is described here as a guide to what sort of files may be imported.

An EMACS style mode line comment may optionally be present as the file's first item. It corresponds to the makefile-environment in the file manager.

```
; -*- Mode: LISP; Package: (FOO GLOBAL 1000); Base:10 -*-
```

mode For some versions of the EMACS editor this will declare the major mode, which arranges key to command bindings for LISP instead of documents.

package Name, used packages and initial space for symbols.

base Numeric "ibase"

The mode line is generated by TextModules and is provided purely as a convenience in transporting code to EMACS based environments. It has no effect on the File Manager. The makefile-environment is actually instated using expressions directly following the mode line.

The Common Lisp community has agreed that portable text files will use only one reader environment and hence not switch packages or alter the readable partway through. TextModules assumes that the reader environment is set up by the seven (plus two) standard environment modifying forms. These forms are recognized by the TextModules parser only if they appear at the front of the file and in order (comments being ignored):

```
Put provide
In in-package
Seven shadow
Extremely export
Random require (or il:filesload)
User use-package
Interface import
... shadowing-import
... setf *read-base*
Commands Contents of module
```

Portable files may optionally add **\*read-base\*** setting and **shadowing-import** expressions. Also, **il:filesload** may be used in place of **require**, when a Lisp file (not containing a **provide** form) must be loaded.

Any one of these forms is optional, but they must appear first in the file (and in order) to be parsed into a makefile-environment when **load-textmodule** is called.

*Warning:* this software applies heuristics to the package forms to make them independent of the package environment they are read in. These may not work and it is recommended that the makefile-environment be checked for correctness after load-textmodule brings in the file. Complex package setups will almost certainly not be handled correctly and should be created in a separate file which the main text file can REQUIRE.

The contents of a text file are a sequence of forms. Certain forms are understood by the File Manager and hence specially recognized. These recognized forms include:

*Comments* which are translated into Interlisp style comments

*Definers* such as defun or defvar

*eval-when* which is translated into a File Manager eval-when

*Read time conditionals* which may become "unread objects"

All other forms are considered *top level forms* and simply saved as is.

Definers hold onto presentations, e.g., read time conditionals, as well as comments. Comments and presentations are always available to be edited.

To see if something is a definer form, examine the property list of its name (like **defun**). Use the Exec's **pl** (print property list) command to look for the property **:definer-for**.

Note that only the above kinds of forms will be recognized by the TextModules parser on a portable Common Lisp file.

There are a few somewhat common problems that can arise when importing a text file. Chief among these are "bootstrapping definitions" and circularity.

## Known Problems

Occasionally, when starting up a complex software system, it is useful to install a temporary definition until the mechanism required by the actual definition is in place. This can cause a definition by the same name to appear in more than one place in a text file. The Textmodules system will simply use the latter definition in the file, causing the next loading of it to fail for lack of the lost bootstrap definition. It is recommended that bootstrap definitions be made into "top level forms", e.g. a DEFUN can become a (SETF (SYMBOL-FUNCTION <name>) ...) form, DEFPARAMETER can become (SETF <name> ...), etc.

Also, some styles of programming may encourage creation of circular structure. Textmodules must map over top level forms to install presentations that contain comments, etc. Circular structures can cause these routines to fill memory with list structure.

---

## File Manager Source Files

Unlike standard Common Lisp, the File Manager is designed to keep all of a file's contents resident in memory as structure, rather than text. This scheme allows very fast update and editing of definitions. To maintain its own source files the File Manager keeps descriptions of the format (makefile-environment) and contents (filecoms) of a file.

The makefile-environment of the file is used to note the readtable and package the rest of the file to be read and printed in, and any file dependencies.

The filecoms maintain a description of the top-level defining forms in a file, like function and variable definitions. They also store plain top-level forms. Within any form there can appear lisp data, like vectors or "number in a radix." How these are read and printed are controlled by presentation types (described separately; see below).

It is important to separate the environment of the file from its contents because the File Manager (not TextModules) first reads all the forms in the file, and then evaluates them. Text based source files sometimes change the package as needed. This cannot work for the File Manager since the file's forms are all read and then executed, i.e. the package changes would not occur until after the entire file had been read, and forms after any `IN-PACKAGE` form would have been read incorrectly.

The File Manager first reads the makefile-environment forms in a well known environment (INTERLISP package, INTERLISP readtable) evaluates them to find the environment of the rest of the source file, then reads the rest of the source file in that environment. This is why the package environment setup forms are so carefully parsed out of text files being imported into the environment.

File Manager source files created by **load-textmodule** depend on both the TEXTMODULES & SEDIT-COMMONLISP modules. These must be loaded before source files created with them can be reloaded.

## File Manager Source File Format

The makefile-environment of a "managed" source file is used to control both how the exported text file and managed source files are printed. It is kept in a property named **il:makefile-environment** on the symbol with the root name of the file (in the INTERLISP package). This property is automatically generated when a portable text file is imported. The property is itself a plist containing :readtable, :package and :base values. The readtable used is called "LISP-FILE", a readtable defined by the TextModules program (hence File Manager source files created by **load-textmodules** depend on TextModules). The part of the makefile-environment of main interest is the :package. It sets the package in which the exported text file is printed.

Three forms of the makefile-environment's :package property are recognized.

- a string or symbol naming a package
- a **defpackage** statement
- a **let** statement

A string or symbol is simply taken to name a package.

A **defpackage** statement will have its portable components translated into a **let** statement as described below.

A **let** used for the makefile-environment should bind **\*package\*** and contain some form of the standard seven package and module setup forms (See above under "Text file format"). It should finally return the altered value of **\*package\***. For example:

```
(let ((*package* *package*))
 ...environment setup forms.
 package
)
```

The forms in this expression must be written in a standard, pre-existing package, such as USER or XCL-USER. This is to break the circularity of writing a package defining form in the package it defines.

The package defining expressions in the let should follow all of the rules for portable text files (See above under "Text file format"), e.g., they should appear in order.

## File Manager Description of Contents

When a portable text file is imported its contents are parsed to produce the File Manager's filecoms (File Commands). File commands are a high-level way of viewing and controlling the ordering of definitions in a file. The following describes the filecoms produced when a text file is imported.

Forms on the file are either recognized as top level defining forms or wrapped in a "top level form" filecom. Several forms are recognized by TextModules itself and others can be added (see below under "Making new specifiers"). The constructs that recognize filecoms, both to export and import plain text, are called *specifiers*.

The following are placed in the filecoms based on the parsed contents of the text file:

|                                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>(il:* type string)</code>           | Contains a comment string. <i>type</i> is a symbol of one, two, three or four semicolons, or a vertical bar. This handles top level single, double or triple semicolon comments, as well as balanced comments. When viewed in SEdit these display in real comment format, instead of the internal list representation.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>(eval-when when . filecoms)</code>  | Wrapper with an evaluation time and containing more filecoms.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <code>definers</code>                     | All definers are recognized, e.g. DEFUN, DEFMACRO, DEFVAR, DEFPARAMETER, DEFSTRUCT, etc. The definer specifier also converts DEFMACRO forms on the <b>*defdefiner-macros*</b> list to defdefiners during loading, and vice versa on printed to a text file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <code>(il:p (top-level-form form))</code> | <p>Top level form wrapper with a macro that calls the presentation translator. This filecom contains expressions which were not recognized and must be evaluated at load time. This kind of filecom also handles top level occurrences of conditional read and read time evaluations (hash comma and hash dot). The top-level-form specifier also looks for LOAded or REQUIRed files and, depending on the variable <b>*convert-loaded-files*</b>, attempts to convert the loaded files as well.</p> <p>When the file is loaded and before evaluating these forms any presentation objects in them are stripped out (as for comments) or installed (as for read time evaluations). This is done by the TOP-LEVEL-FORM macro, which dispatches to the translation functions for the particular presentation objects. i.e., this allows comments to appear anywhere in the forms and not affect evaluation.</p> <p>The above coms are created when a text file is imported. There are also a few specifiers provided to export filecoms, but not create them on import. These are convenient for exporting typical File Manager files. They are:</p> |
| <code>(il:coms . filecoms)</code>         | Used to group together definitions in a File Manager source file. The <i>filecoms</i> are dumped onto the text file in order. No information is placed on the resulting text file to preserve the grouping of the filecoms; if the exported text file is later imported the coms grouping will not reappear.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

|                                               |                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>(il:vars . descriptors)</code>          | As described in the <i>IRM</i> . The <i>descriptors</i> are exported as <b>defparameter</b> forms. If the exported text file is later imported these <b>vars</b> coms will reappear under <b>variables</b> coms.                                                                                                                                                                                             |
| <code>(il:initvars . descriptors)</code>      | As described in the <i>IRM</i> . The <i>descriptors</i> are exported as <b>defvar</b> forms. If the exported text file is later imported these <b>initvars</b> coms will reappear under <b>variables</b> coms.                                                                                                                                                                                               |
| <code>(il:constants . descriptors)</code>     | As described in the <i>IRM</i> . The <i>descriptors</i> are exported as <b>defconstant</b> forms. If the exported text file is later imported these <b>constants</b> coms will reappear under <b>variables</b> coms.                                                                                                                                                                                         |
| <code>(il:props . descriptors)</code>         | As described in the <i>IRM</i> . The <i>descriptors</i> are exported as <b>(setf (getf ...) ...)</b> forms. If the exported text file is later imported these <b>props</b> coms will reappear under <b>p</b> coms (top-level forms).                                                                                                                                                                         |
| <code>(il:prop <i>props</i> . symbols)</code> | As described in the <i>IRM</i> . The <i>props</i> and <i>symbols</i> descriptors are used to generate forms for export, e.g. <b>(setf (getf 'foo 'bar) 21)</b> . If the exported text file is later imported these <b>prop</b> coms will reappear under <b>p</b> coms (top-level forms).                                                                                                                     |
| <code>(il:files . items)</code>               | As described in the <i>IRM</i> . The <i>items</i> are used to generate forms for export, e.g. <b>(load "Foo.lisp")</b> . All options except <b>noerror</b> are ignored, the latter will cause the <b>:if-does-not-exist nil</b> key argument to be included in the load expression. If the exported text file is later imported these <b>files</b> coms will reappear under <b>p</b> coms (top-level forms). |

## Making New Specifiers

Specifiers are the glue that relate forms on a plain text file to filecoms in a File Manager source file. They can be considered addenda to the filepkgtype mechanism of the File Manager.

**\*specifiers\*** [Variable]

A list of specifiers (its default contents are described below). New specifiers should be added to this list. This list is searched linearly; its order is significant mostly in that the default top-level form recognizer must always be last.

**make-specifier** &key *name filecom-p form-p add-form install-form print-filecom* [Function]

This function creates new specifiers for inclusion on the **\*specifiers\*** list. A specifier maps between the forms in a text file's contents and filecoms. It is the basis for importing and exporting top-level forms. Specifiers can be nested, as for EVAL-WHEN.

To do all of this a specifier contains functions that recognize forms of its kind on the text file and coms in filecoms, as well as functions that add the definition to the filecoms and install the definition as the one to be used at runtime. Finally, there is a function which prints a form onto a text file based on a com on the filecoms.

|                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>name</i>          | A string naming the specifier.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <i>filecom-p</i>     | Predicate on FILECOM which returns true if it is the one used to represent this specifier in the filecoms of a managed file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <i>form-p</i>        | Predicate on FORM, a form read from a text file being imported, which returns true if this is the specifier for the definition in FORM.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <i>add-form</i>      | <p>Function of FORM and FILECOMS. FORM is a form read from a text file being imported, and which has already been confirmed with the <i>form-p</i> method. Should make the definition in FORM available (editable) in the programming environment (File Manager). It should make the definition editable and add a filecom for FORM to the FILECOMS description. It should return the new FILECOMS description. To <i>add-form</i> runs of subforms use <b>add-form</b> and <b>form-specifier</b> (see below).</p> <p>Care should be used when making a definition editable. The simplest instance of this occurs when the FORM's definition is a definer. In this case its evaluation may be wrapped in a binding of <b>il:dfnflg</b> to <b>il:prop</b> to ensure that the definition form goes into the table of current definitions <i>without being evaluated</i>.</p> <p>Adding a filecom to the FILECOMS should be done in a way that preserves ordering. The simplest way to do this is to append the new filecom to the end of the current FILECOMS.</p> |
| <i>install-form</i>  | <p>Function of a FORM which makes the definition in FORM the current one to be used in execution. If the defining mode flag indicates that the file is being loaded for editing only this function <i>will not be called</i> during loading of the form (il:dfnflg is set by the :install option to load-textmodule). To <i>install</i> runs of subforms use <b>install-form</b> and <b>form-specifier</b> (see below).</p> <p>Care should be used when making a definition executable. The simplest instance of this occurs when the FORM's definition is a definer. In this case its evaluation may be wrapped in a binding of <b>il:dfnflg</b> to <b>t</b> to ensure that the definition form <i>is actually evaluated</i>.</p>                                                                                                                                                                                                                                                                                                                               |
| <i>print-filecom</i> | <p>Function of FILECOM and STREAM which should generate a new line and pretty print a form, representing the FILECOM, onto the stream. To print runs of subforms use <b>print-filecom</b> and <b>filecom-specifier</b> (see below).</p> <p>The semantics of the <i>add-form</i> and <i>install-form</i> methods remove some confusion between loading a definition into memory for editing (loading PROP) and installing that definition as the currently executable one (loading T). The <i>add-form</i> method makes the definition editable and the <i>install-form</i> makes it executable.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                              |

The following functions are used to handle subforms EG. In the eval-when specifier there are subforms that need to be parsed.

`form-specifier form` [Function]

Searches the current list of specifiers in an attempt to recognize *form* (as read from a text file being imported). Returns a specifier for *form*. If none is found a warning is signalled and a "do nothing" specifier is returned.

`filecom-specifier filecom` [Function]

Searches the current list of specifiers in an attempt to recognize *filecom* (as found in the filecoms of the managed file). Returns a specifier for *filecom*. If none is found a warning is signalled and a "do nothing" specifier is returned.

`add-form form filecoms &optional specifier` [Function]

Adds the *form* to the *filecoms* description based on the add method in the *specifier*. If *specifier* is not provided `form-specifier` will be used to get it from *form*. Returns the new filecoms. `nil` is used as an empty contents description.

`install-form form &optional specifier` [Function]

If the current definition mode (`il:dfnflg`) allows it, installs the *form* as current and executable based on the *specifier*. If *specifier* is not provided `form-specifier` will be used to get it from *form*.

`print-filecom filecom stream &optional specifier` [Function]

Prints a new line on *stream* and then pretty prints a form representing the filecom onto the *stream*. If *specifier* is not provided `filecom-specifier` will be used to get it from *filecom*.

## Presentation Objects

Presentation objects represent things that normally disappear during reading, like comments or numbers written in a particular base. Each presentation object must be capable of being read from a text file, edited with SEdit, installed as it would be when normally read, and printed to a text file in its original form.

### Presentations Supported by Lisp

Many presentations are already supported by SEdit :

|                           |                                                                                                                                                                                                                                                                                                                |
|---------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>#\character</code>  | Character object                                                                                                                                                                                                                                                                                               |
| <code>#:symbol</code>     | Uninterned symbol                                                                                                                                                                                                                                                                                              |
| <code>#*function</code>   | Hash quote function abbreviation                                                                                                                                                                                                                                                                               |
| <code>; comment</code>    |                                                                                                                                                                                                                                                                                                                |
| <code>;; comment</code>   |                                                                                                                                                                                                                                                                                                                |
| <code>;;; comment</code>  |                                                                                                                                                                                                                                                                                                                |
| <code>;;;; comment</code> | Semicolon comments. Internal formatting is preserved when these are imported, including CRs and tabs, etc. Adjacent comments are <i>not</i> smashed together so that line breaks are preserved. A single leading space in a comment is ignored, since comments are always printed with a single leading space. |
|                           | These comment types are represented internally in the same way as in Lisp, i.e., a list beginning with the symbol                                                                                                                                                                                              |

IL:\*, following by a symbol (interned in the INTERLISP package) containing one through four semicolons.

`#|comment|`

A balanced comment. Imported and exported by TextModules. Directly supported by SEdit as though it were a "level 5" semicolon comment.

This comment type is represented internally in a manner similar to semicolon comments, but where the symbol containing semicolons is replaced by a symbol whose name is the vertical bar character.

## Presentations Supported by SEDIT-COMMONLISP

Several presentations are specially supported by SEDIT-COMMONLISP. Any of these can be created using the following commands in SEdit:

|                        |           |            |
|------------------------|-----------|------------|
| Read time conditionals | Control-N | Hash minus |
|                        | Control-P | Hash plus  |
| Read time evaluation   | Control-Q | Hash dot   |
| Load time evaluation   | Control-F | Hash comma |
| Octal notation         | Control-I | Hash "O"   |
| Hexidecimal notation   | Control-J | Hash "X"   |
| Binary notation        | Control-K | Hash "B"   |

`#+feature form`

`#-feature form`

Read time conditionals.

A conditional expression can be either *unread* or *read* depending on whether the truth of its features expression and sign parse true (see *Common Lisp, the Language*). *Read* conditional expressions are stored as structure. *Unread* conditional expressions are stored as strings due to the potential inclusion of, e.g., numbers of higher precision, symbols in unknown packages, or the inclusion of unknown reader macros.

*Unread* read time conditionals are read by remembering file position, doing a read suppress read, backing up to the original position and saving all the characters between in a string. This means that streams from which conditional read presentations are read must be capable of random access (the TTY is not).

These are represented by hash-plus and hash-minus structures.

Editing of read time conditional presentations is not quite WYSIWYG. Feature symbols should always be given as keywords (this is done implicitly by the lisp reader, but not in SEdit) and unread forms appear in strings and must be edited as such.

`#.form`

Read time evaluation. Hash dot is represented by the hash-dot presentation.

`#.form`

Load time evaluation. Hash comma is represented by the hash-comma structure.

`#Orational`



*#Xrational*  
*#Brational* Rational representations (**O**ctal, **h**ex**I**decimal, and **B**inary). These are represented by the hash-o, hash-x and hash-b structures.

## Presentations Not Directly Supported

It is not possible to directly edit the following presentations in SEdit:

*#(contents)* Vectors  
*#rankA(contents)* Arrays  
*#S(name field1 value1 . . .)* Structures  
*#\*1010101* Bit vectors

Any of these may be edited by opening an inspector from SEdit: selecting the object and using the Meta-E command on it. Any of these may be created in SEdit by inserting the appropriate **make-** expression, selecting it, and using the Meta-Z command with **cl:eval** as the mutating function.

## Presentations Not Supported

The following standard Common Lisp presentations are not supported by either SEdit or TextModules:

*#n=object* and *#n#* Object tag and reference notation  
*#baseRnumber* Radix notation

[This page intentionally left blank]

---

---

## UNICODE

---

---

By Ron Kaplan

This document was last edited in July 2023.

The `UNICODE` library package defines external file formats that enable Medley to read and write files where 16 bit character codes are represented as UTF-8 byte sequences or big-endian UTF16 byte-pairs. It also provides for character codes to be converted (on reading) from Unicode codes to equivalent codes in the Medley-internal Xerox Character Code Standard (`XCCS`) and (on writing) from `XCCS` codes to equivalent Unicode codes.

### UTF-8 External formats

Four external formats are defined when the package is loaded:

- `:UTF-8` codes are represented as UTF-8 byte sequences and `XCCS`/Unicode character conversion takes place.
- `:UTF-16BE` codes are represented as 2-byte pairs, with the high order byte appearing first in the file, and characters are converted.

The two other external formats translate byte sequences into codes, but do not translate the codes. These allow Medley to see and process characters in their native encoding.

- `:UTF-8-RAW` codes are represented as UTF-8 byte sequences, but character conversion does not take place.
- `:UTF-16BE-RAW` codes are represented as big-ending 2-byte pairs but there is no conversion.

These formats all define the end-of-line convention (mostly for writing) for the external files according to the variable `EXTERNALEOL` (`LF`, `CR`, `CRLF`), with `LF` the default.

The external format can be specified as a parameter when a stream is opened:

```
(OPENSTREAM 'foo.txt 'INPUT 'OLD ' ((EXTERNALFORMAT :UTF-8)))
(CL:OPEN 'foo.txt :DIRECTION :INPUT :EXTERNAL-FORMAT :UTF-8)
```

The function `STREAMPROP` obtains or changes the external format of an open stream:

```
(STREAMPROP stream 'EXTERNALFORMAT) -> :XCCS
(STREAMPROP stream 'EXTERNALFORMAT :UTF-8) -> :XCCS
```

In the latter case, the stream's format is changed to `:UTF-8` and the previous value is returned, in this example it is Medley's historical default format `:XCCS`.

Entries can be placed on the variable `*DEFAULT-EXTERNALFORMATS*` to change the external format that is set by default when a file is opened on a particular device. Loading `UNICODE` executes

```
(PUSH *DEFAULT-EXTERNALFORMATS* '(UNIX :UTF-8))
```

so that all files opened (by `OPENSTREAM`, `CL:OPEN`, etc.) on the UNIX file device will be initialized with `:UTF-8`. Note that the UNIX and DSK file devices reference the same files (although some caution is needed because `{UNIX}` does not simulate Medley versioning), but the device name in a file name (`{UNIX}/Users/...` vs. `{DSK}/Users/...`) selects the particular device. The default setting above applies only to files specified with `{UNIX}`; a separate default entry for DSK must be established to change its default from `:XCCS`.

The user can also specify the external format on a per-stream basis by putting a function on the list `STREAM-AFTER-OPEN-FNS`. After `OPENSTREAM` opens a stream and just before it is returned to the calling function, the functions on that list are applied in order to arguments `STREAM`, `ACCESS`, `PARAMETERS`. They can examine and/or change the properties of the stream, in particular, by calling `STREAMPROP` to change the external format from its device default.

## Mapping between Unicode and XCCS character codes

The XCCS/Unicode mapping tables are defined by the code-mapping files for particular XCCS character sets. These are typically located in the Library> sister directory

```
..>Unicode>Xerox>
```

and the variable `UNICODEDIRECTORIES` is initialized with a globally valid reference to that path. The global reference is constructed by prepending the value of the Unix environment-variable "MEDLEYDIR" to the suffix `>Unicode>Xerox>`.

The mapping files have conventional names of the form `XCCS-[charsetnum]=[charsetname].TXT`, for example, `XCCS-0=LATIN.TXT`, `XCCS-357=RSYMBOLS4.TXT`. The translations used by the external formats are read from these files by the function

```
(READ-UNICODE-MAPPING FILESPEC NOPRINT NOERROR) [Function]
```

where `FILESPEC` can be a list of files, charset octal strings ("0" "357"), or XCCS charset names (LATIN EXTENDED-LATIN). Reading will be silent if `NOPRINT`, and the process will not abort if an error occurs and `NOERROR`. The value is a flat list of the mappings for all the character sets, with elements of the form `(XCCC-code Unicode-code)`.

When `UNICODE` is loaded the mappings for the character sets specified in the variable `DEFAULT-XCCS-CHARSETS` are installed. This is initialized to

```
(LATIN SYMBOLS1 SYMBOLS2 EXTENDED-LATIN FORMS SYMBOLS3 SYMBOLS4 ACCENTED-
LATIN GREEK)
```

but `DEFAULT-XCCS-CHARSETS` can be set to a different collection before `UNICODE` is loaded.

The internal translation tables used by the external formats are constructed from a list of correspondence pairs by the function

```
(MAKE-UNICODE-TRANSLATION-TABLES MAPPING [FROM-XCCS-VAR]
[TO-XCCS-VAR]) [Function]
```

This returns a list of two arrays (XCCS-to-Unicode Unicode-to-XCCS) containing the relevant translation information organized for rapid access. If the optional from/to-variables arguments are provide, they are the names of variables whose top-level values will be set to these arrays, for convenience. For the external formats defined above, these variables are `*XCCSTOUNICODE*` and `*UNICODETOXCCS*`.

The macro

`(UNICODE.TRANSLATE CODE TRANSLATION-TABLE)` [Macro]

is used by the external formats to perform the mappings described by the translation-tables.

The following utilities are provided for lower-level manipulation of codes and strings.

`(XTOUCODE XCCSCODE)` -> corresponding Unicode

`(UTOXCODE UNICODE)` -> corresponding XCCS code

`(NUTF8CODEBYTES N)` -> number of bytes in the UTF-8 representation of N.

`(NUTF8STRINGBYTES STRING RAWFLG)` -> number of bytes in the UTF-8 representation of STRING, translating XCCS to Unicode unless RAWFLG.

`(XTOUSTRING XCCSSTRING RAWFLG)` -> The string of bytes in the UTF-8 representation of the characters in XCCSSTRING (= the bytes in its UTF-8 file encoding).

`(HEXSTRING N WIDTH)` -> the hex string for N, padded to WIDTH

The UNICODE file also contains a function for writing a mapping file given a list of mapping pairs. The function

`(WRITE-TRANSLATION-TABLE MAPPING [INCLUDEDCHARSETS] [FILE])`

produces one or more mapping files for the mapping-pairs in mapping. If the optional FILE argument is provided, then a single file with that name will be produced and contain all the mappings for all the character sets in MAPPING. If FILE and INCLUDEDCHARSETS are not provided, then all of the mappings will again go to a single file with a composite name XCCS-csn1,csn2,csn3.TXT. Each cs may be a single charset number, or a range of adjacent charset numbers. For example, if the mappings contain entries for characters in charset LATIN, SYMBOLS1, SYMBOLS2, and SYMBOLS3, the file name will be XCCS-0,41-43.TXT.

If INCLUDEDCHARSETS is provided, it specifies possibly a subset of the mappings in MAPPING for which files should be produced. This provides an implicit subsetting capability.

Finally, if FILE is not provided and INCLUDEDCHARSETS is T, then a separate file will be produced for each of the character sets, essentially a way of splitting a collection of character-set mappings into separate canonically named files (e.g. XCCS-357=SYMBOLS4.TXT).

# UNIXCHAT

---

The UNIXChat library module is similar to Chat, but communicates with a C-Shell on your own host rather than with another machine.

## Requirements

---

UNIXChat depends on Chat and UNIXComm.

## Installation

---

Load UNIXCHAT.LCOM from the library. We recommend that you also load VTCHAT.LCOM from the library to ensure support.

## User Interface

---

Use the following procedure to open a Chat window and activate the terminal emulator.

1. Invoke Chat in one of the following ways:
  - a. Choose CHAT from the background menu and type **SHELL** at the Host: prompt in the Prompt Window.
  - b. Call the Chat function:

**(CHAT 'SHELL LOGOPTION INITSTREAM WINDOW)**

See the CHAT module in this manual for the definition of the Chat function.  
Use the variable CHAT.DISPLAYTYPES, if necessary.

You should now be talking to a UNIX C-Shell in the Chat window.

2. Set your terminal type:

prompt% **setenv TERM vt100**

Note: Alternately, shells started from UNIXChat have the shell variable LDESHELL set. Thus, in your .cshrc file you could have

```
if ($?LDESHELL == 1) then
 setenv TERM vt100
 stty erase ↑H
endif
```

This makes your Chat C-Shells easier to use.

3. When you've completed a Chat session, close the connection by using CLOSE in the right button window menu or by typing exit to the C-Shell.

Note: You can have several Chat windows open to SHELL at the same time.

[This page intentionally left blank]

# UNIXCOMM

---

UNIXComm starts up a UNIX process on a Sun Workstation. UNIXComm, with two functions, provides the user with a stream interface to the SunOS Bourne shell.

## Installation

---

Load UNIXCOMM.LCOM from the library.

## User Interface

---

Two functions allow you to open and close SunOS subprocess streams.

(IL:CREATE-PROCESS-STREAM *STRING*) [Function]

Interfaces to the SunOS system function (see Chapter 3 of the *SunOS Reference Manual*). This causes a subprocess running the Bourne shell to be spawned; *STRING* is passed as a command to it. A bidirectional stream is returned with input reading data which the process writes to its standard output (`stdout`) and standard error (`stderr`). Output writes data which the process can read from its standard input (`stdin`). EOFP does not return a meaningful value for those streams; instead, you must change the default EOF mechanism for the end of stream.

(IL:UNIX-STREAM-CLOSE *STREAM*) [Function]

Returns the numeric status value of the process created by IL:CREATE-PROCESS-STREAM if it has exited; otherwise, kills the process. IL:CLOSE can be used instead of IL:UNIX-STREAM-CLOSE if the process status value is not of interest.



[This page intentionally left blank]

UnixPrint lets you arrange to have hardcopy sent directly to a PostScript printer via a UNIX print command. You can set your default printing host so that it happens automatically.

---

## Installation

---

Load UNIXPRINT.DFASL. Customize UNIXPRINTCOMMAND.

Then set the two control variables appropriately, as described below:

DEFAULTPRINTINGHOST [Variable]

This is a list of printer names, described in the *Interlisp Reference Manual* (refer to the IRM for a general description). To add a PostScript printer to the list, add an entry in the form (POSTSCRIPT printername). To continue the example above, DEFAULTPRINTINGHOST should have a value like the following:

```
(SETQ DEFAULTPRINTINGHOST
 '(-- (POSTSCRIPT daisy) --))
```

UnixPrinterName [Variable]

A string or symbol, the name of the UNIX printer to which you want output sent. This should be the name that you would give in the lpr command. For example, if you normally print files by entering:

```
lpr -Pdaisy...
```

then you should (SETQ UnixPrinterName "daisy"). If you do not normally specify a printer name, set UnixPrinterName to NIL.

---

## Customization

---

You can get UNIXPRINT to use lp or lpr by modifying the function UnixPrintCommand. Your site may have a printing program other than lpr. For further information about printing on your system, please refer to your system manual.

(UnixPrintCommand PRINTER COPIES NAME TMPNAME) [function]

Returns a string that is used by /bin/sh in the printing of the postscript code. The arguments are PRINTER, COPIES, NAME and TMPNAME. PRINTER is the name of the printer. COPIES is NIL or a fixp specifying how many copies to print. NAME is the string printed on the banner page of your hardcopy. TMPNAME is the name of the temporary file used to store the postscript code for your job.

A call to `(UnixPrintCommand "daisy" 1 "Erik" "/tmp/foot")` should return something like the string `"/usr/ucb/lpr -Pdaisy -#1 -JErik -r -s /tmp/foot"`.

The source code of the function `UnixPrintCommand` is supplied with Medley. You are encouraged to write your own versions of this function depending on the site he or she uses. The function `UnixPrintCommand` is included in `UNIXPRINT.DFASL`. Sources and examples for different versions of `UnixPrintCommand` are included in the file `UNIXPRINTCOMMAND`.

[This page intentionally left blank]

# KEYBOARDEDITOR

---

KeyboardEditor is intended for use with the VirtualKeyboards module. You should read that module's documentation before reading this. The KeyboardEditor module lets you create new virtual keyboards and change existing ones to suit your needs.

## Requirements

---

VIRTUALKEYBOARDS

## Installation

---

Load KEYBOARDEDITOR.LCOM and VIRTUALKEYBOARDS.LCOM from the library.

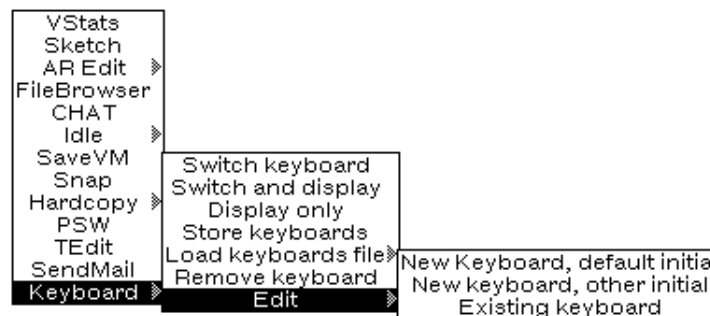
## User Interface

---

Loading KeyboardEditor adds EDIT to the Virtual Keyboard submenu on the background menu.

### Background Menu

The keyboard editor is used to modify and create virtual keyboards. You can call it by selecting EDIT from the main KeyboardEditor/VirtualKeyboards menu and sliding the cursor to the right to bring up the editor menu. You can also simply select EDIT, which gives you the same options as NEW KEYBOARD, DEFAULT INITIAL.



### Creating a New Keyboard From a Copy of the Default Keyboard

Choose NEW KEYBOARD, DEFAULT INITIAL to create a keyboard from a copy of the default keyboard (which initially has the same key assignments as the 1108 keyboard). The system prompts you for a name for the new keyboard, then call the editor with a copy of the default keyboard as the initial keyboard. The key assignments that are not changed during the editing session remain as they are in the default keyboard.

## Creating a New Keyboard From a Copy of Any Known Keyboard

To create a new keyboard from a copy of a known keyboard other than the default keyboard, select `NEW KEYBOARD, OTHER INITIAL` from the Edit submenu. You are prompted for a name for the new keyboard. The system then displays a menu of the known keyboards from which to choose the initial keyboard.

```
Quit
DEFAULT
EUROPEAN
logic
MATH
OFFICE
DVORAK
GREEK
ITALIAN
SPANISH
FRENCH
GERMAN
STANDARD-RUSSIAN
```

## Changing an Existing Keyboard

You can change an existing keyboard by selecting `EXISTING KEYBOARD` from the Edit submenu. Like the `NEW KEYBOARD, OTHER INITIAL` command, this brings up a menu of known keyboards from which you can choose a keyboard for editing. However, you are not prompted for a keyboard name first, because you are editing the actual keyboard rather than using it as a base for a new keyboard.

## Calling the Keyboard Editor From Lisp

The editor can also be called using the function

`(EDITKEYBOARD KEYBOARD INITIALKEYBOARD)` [Function]

where *KEYBOARD* is either a virtual keyboard (i.e., a list) or the name of a virtual keyboard. If *KEYBOARD* is a virtual keyboard or the name of a known keyboard (a keyboard that was defined before), the editing is done on that keyboard and the second argument is ignored.

If *KEYBOARD* is a new name, the editing is done on a copy of *INITIALKEYBOARD*, with *KEYBOARD* as its new name. If *INITIALKEYBOARD* is `NIL`, the default keyboard is used as a base keyboard.

Examples:

To create a totally new virtual keyboard, call `(EDITKEYBOARD NEWNAME)`.

To create a new keyboard that is similar to a keyboard with the name `K1`, call `(EDITKEYBOARD NEWNAME 'K1)`

To modify a keyboard with the name `GREEK`, call `(EDITKEYBOARD 'GREEK)`.

---

## Using the Keyboard Editor

There are four different keyboard editor menus, three of them displayed at any given time. After you call the editor, the command menu is at the top, the character menu in the middle, and the keys menu at the bottom.

| CharSet           | Stop           |    |    |     | Quit          |     |     |     | Define |     |     |     |
|-------------------|----------------|----|----|-----|---------------|-----|-----|-----|--------|-----|-----|-----|
| SHIFT<br>LOCKDOWN | CTRL<br>LOCKUP |    |    |     | META<br>EVENT |     |     |     | LOCK   |     |     |     |
| Character set 0   |                |    |    |     |               |     |     |     |        |     |     |     |
| 0                 | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 0                 | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 1                 | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 2                 | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 3                 | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 4                 | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 5                 | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 6                 | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 7                 | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 10                | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 11                | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 12                | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 13                | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 14                | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 15                | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 16                | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |
| 17                | 20             | 40 | 60 | 100 | 120           | 140 | 160 | 200 | 220    | 240 | 260 | 300 |

Virtual Keyboard : MATH

|       |   |   |   |   |   |   |   |   |   |   |   |   |    |       |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|----|-------|
| ESC   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | - | = | BS |       |
| TAB   | C | + | × | R | ⌞ | ↔ | ↶ | ↷ | ↵ | ↶ | ↷ | [ | ]  | CR    |
| LOCK  | A | f | D | F | ≈ | H |   |   |   |   | ⌞ | ⌞ | ⌞  |       |
| SHIFT | ± | ÷ | L | J | ⌞ | ⌞ | ⌞ | ⌞ | ⌞ | ⌞ | ⌞ | ⌞ | ⌞  | SHIFT |
| SPACE |   |   |   |   |   |   |   |   |   |   |   |   |    |       |

**Figure 6. Character Display**

The character menu is a 16-by-16-character display of the 256 characters available in the current character set. The set that is displayed when you enter the editor is character set 0, which includes all of the ASCII characters plus many other symbols. See Figure 6. If you need characters from other character sets, you have to select Char Set from the command menu. A new menu pops up that contains numbers from 0 to 377 octal. This is the character set menu, and it lets you switch the character menu to display characters from other sets. Most of the character set numbers are not currently implemented. The most useful ones are shown in Figure 7.

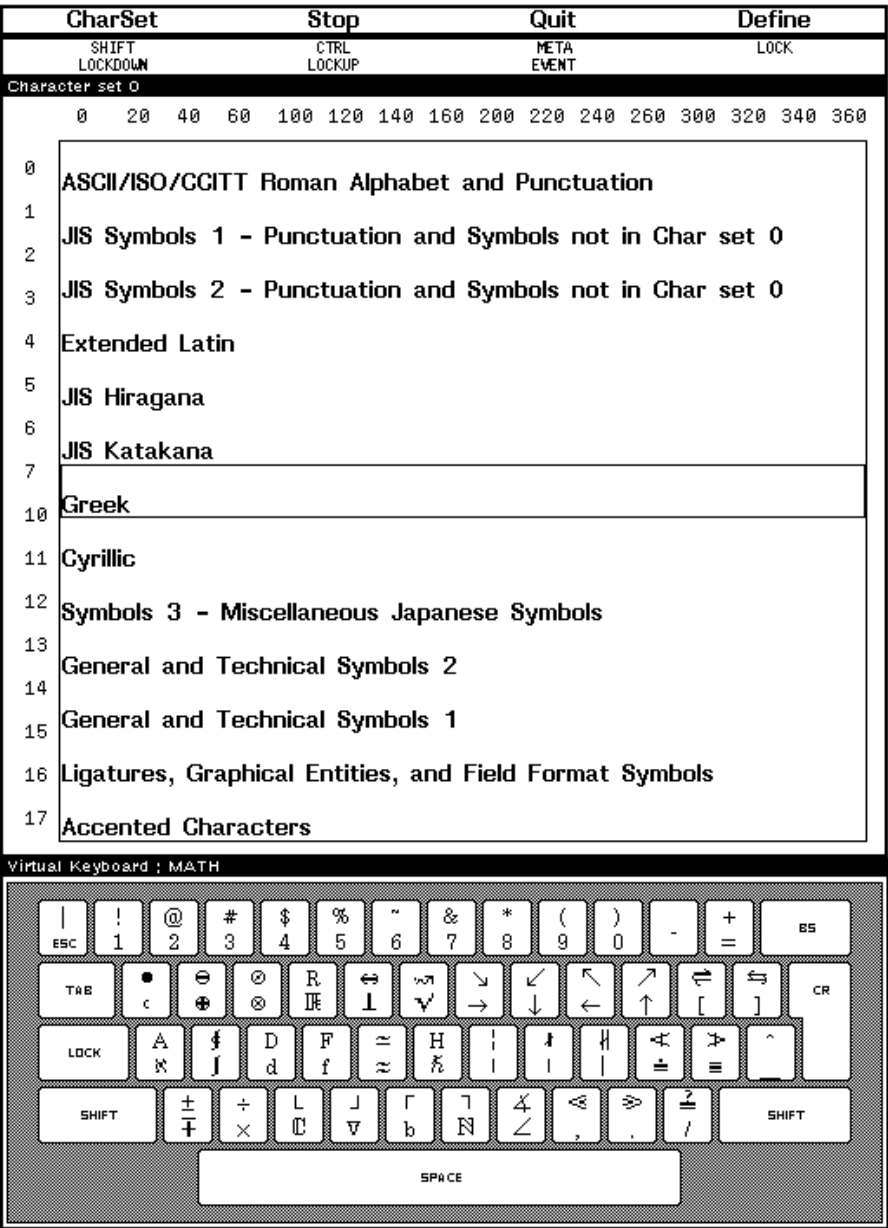


Figure 7. Character Sets

The keys menu lets you make a key the current key by selecting it. A selected key is marked by a black frame. To make a shifted key the current key, Shift-select the key (hold the Shift key down and click on the icon with the left button); it is marked by inverted Shift keys in addition to the black frame.

The basic operation of editing is assigning a character to a key. You can only assign character keys; keys other than character keys retain their current definitions. You assign a character to a key by selecting the key from the keys menu, then selecting the character from the character menu. If the character is to be assigned to the shifted key, select the shifted key as the current key.

A second type of editing operation is to change the LOCKSHIFT state of a key. Each key either has or does not have a LOCKSHIFT property. If a key has a LOCKSHIFT property and the shift lock key of the keyboard is down, typing the key on your



workstation keyboard sends the shifted character of the key, regardless of the state of the shift keys. The same rule applies to a virtual displayed keyboard; if the LOCK item is inverted and the key has a LOCKSHIFT property, selecting a key sends the shifted character to the current input stream.

If a key has the LOCKSHIFT property, the lock key is inverted in the keys menu. To change the LOCKSHIFT property of a key, first make the shifted key the current key. Then set or unset the LOCKSHIFT property by selecting the lock key from the keys menu.

If you are creating a new keyboard and you are satisfied with the key assignments, select Define from the command menu. This adds the newly created keyboard to the list of known keyboards (it will thus appear on future menus). Quit exits after modifying the virtual keyboard; Stop exits without modifying the keyboard. In both cases the new keyboard is returned to the caller of EDITKEYBOARD function (above).

## Creating New Keyboard Configurations

KEYBOARDCONFIGURATION

[Record]

Describes a physical keyboard: its layout, the key numbers that are used with KEYACTION. It also describes each key: its default meaning, its default label, whether you can change the key's meaning with the keyboard editor.

A configuration consists of a number of parts:

CONFIGURATIONNAME

[Record field]

The name of this configuration.

For example, KeyboardEditor comes with configurations named DANDELION (1108), DORADO (1132), DOVE (1186), and FULL-IBMPC.

KEYSIDLIST

[Record field]

An Alist of the IDs you use for the keys in the rest of the configuration; i.e., your names for the keys. For simplicity, these are usually numbers starting beyond 100 (to avoid overlapping the true range of key numbers).

KEYREGIONS

[Record field]

An Alist of key IDs and the regions they occupy in the keyboard's image when it is displayed. For example, the alphabetic keys in the DANDELION keyboard are 29 screen points wide and 33 high.

DEFAULTASSIGNMENT

[Record field]

An Alist of key IDs and their default KEYACTIONS (see *IRM*).

KEYNAMESMAPPING

[Record field]

An Alist of key names to key IDs. The key names should be mnemonic, and should distinguish relevant differences; e.g., the 7 on the 1186's numeric keypad is named NUMERIC7, while the 7 key in the main keyboard cluster is named 7.

MACHINETYPE [Record field]

The kind of machine for which this configuration is intended.

For example, the FULL-IBMPC configuration is meant to be used with a DAYBREAK keyboard, so its MACHINETYPE is DAYBREAK.

KEYLABELS [Record field]

An Alist of key numbers to special labels. This is used to label keys such as the “Next” key, where the key assignment may not be a printable character.

KEYLABELSFONT [Record field]

The font you want to use for the key labels. The default value is Helvetica 5.

BACKGROUNDSHADE [Record field]

The shading for the non-key parts of the virtual keyboard’s image. This defaults to a reasonable gray value.

KEYBOARDDISPLAYFONT [Record field]

The font used to display actual character assignments. This should probably be Classic 12, since it is the most complete font.

CHARLABELS [Record field]

An Alist from character codes to names. Used to give symbolic names to characters such as ESCAPE, which do not otherwise print.

ACTUALKEYSMAPPING [Record field]

A function that takes one of your key IDs and returns a true key number, for use by KEYACTION.

Note: To create a new configuration, create an instance of the KEYBOARDCONFIGURATION record, using the field names shown above. Then add it to the list VKBD.CONFIGURATIONS. You may then edit it using the configuration editor described below.

Note: You must save your own configurations. There is no user interface for saving them, nor any automatic scheme.

---

## Editing a Keyboard Configuration

Once you have created a KEYBOARDCONFIGURATION, you can make modest changes to it using the function:

(EDITCONFIGURATION *CONFIGNAME*) [Function]

where *CONFIGNAME* is the CONFIGURATIONNAME you have assigned to your new configuration. This creates a virtual keyboard editing window with a menu on top of it as shown in Figure 8.

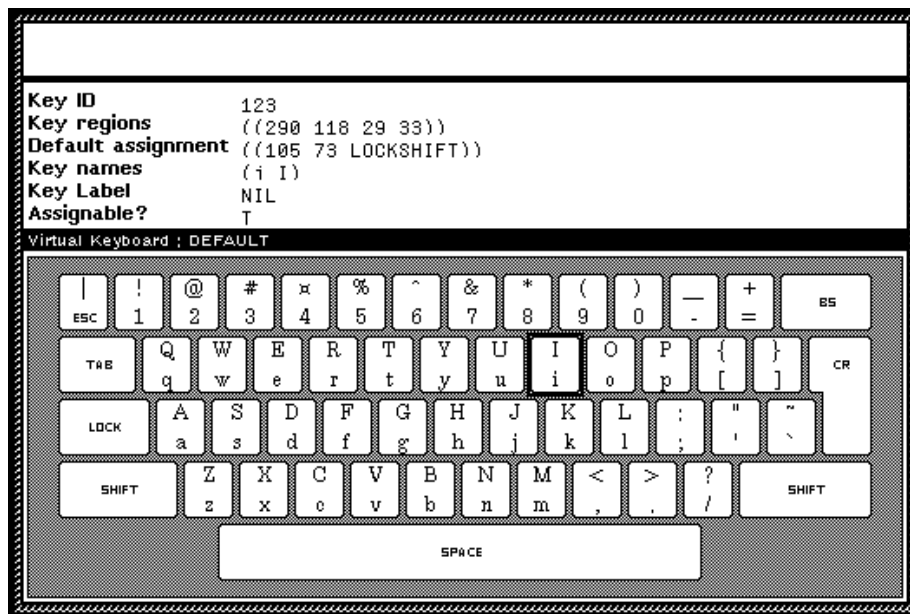


**Figure 8. Virtual Keyboard Editing Window**

Selecting a key with the mouse fills in the fields in the menu. The figure shows the 1108's configuration being edited, with the I key selected. To change one of the values, select the label at the left edge of the menu (e.g., ASSIGNABLE?). You are prompted to edit the existing value using TTYIN.

The keyboard image is not automatically updated. To refresh it, select REDISPLAY in the right-button window menu.

When you have finished editing, simply close the keyboard window.

[This page intentionally left blank]

# VIRTUAL KEYBOARDS

---

VirtualKeyboards lets you change the behavior of your Lisp workstation keyboard to mimic another keyboard, hence making yours a “virtual” version of that other keyboard. It also lets you display pictures of keyboards on your screen and use them as menus for typing occasional special characters. Several keyboards may be displayed on the screen at once, letting you switch easily among keyboards for several languages and making hundreds of characters available for typing.

The virtual keyboards supplied with the module are Dvorak, German, Greek, Italian, logic, math, Spanish, European accents, and standard Russian. You can also define new keyboards with the associated Keyboard Editor module, which lets you edit a keyboard while seeing the actual look of the characters.

The virtual keyboards can be used with TEdit, DEdit, in the Lisp Executive window, for any application with which you use your keyboard.

## Requirements

---

You need one of the following files, as appropriate for the machine you are using:

DANDELIONKEYBOARDS

DORADOKEYBOARDS

DOVEKEYBOARDS

## Installation

---

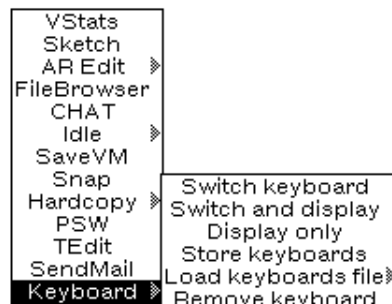
Load VIRTUALKEYBOARDS.LCOM from the library. VirtualKeyboards loads the xxxKEYBOARDS files.

## User Interface

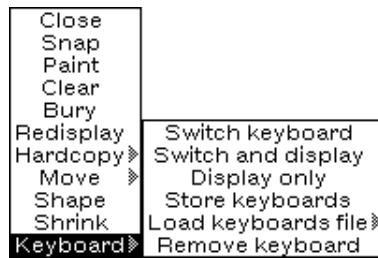
---

Loading VirtualKeyboards adds the item KEYBOARD to the background menu and the default window menu. Using the mouse in this module is the same as in the KeyboardEditor.

Selecting this item from the background menu changes your keyboard for all windows.



Selecting this item from the default window menu allows you to specify a keyboard for an individual window.



The main keyboard module commands are described in detail below.

## Switch Keyboards

Use the `SWITCH KEYBOARD` command to change the behavior of your keyboard to that of a selected virtual keyboard. This brings up a menu of the keyboards currently known to the program.



Select the keyboard you want to substitute for your workstation keyboard. Once you have changed your keyboard's behavior, pressing a key will send the character newly assigned to that key to the current input stream.

## Switch & Display a Keyboard

Use the `SWITCH AND DISPLAY` command to change the behavior of your keyboard to that of a selected keyboard and, in addition, display that keyboard's layout on the screen. You will be offered a menu of the keyboards known to the program; select the one you want to substitute for your workstation's keyboard. Displaying the keyboard layout helps if you're typing on an unfamiliar keyboard. `SWITCH AND DISPLAY` lets you type characters by using the keyboard displayed on the screen as a menu.

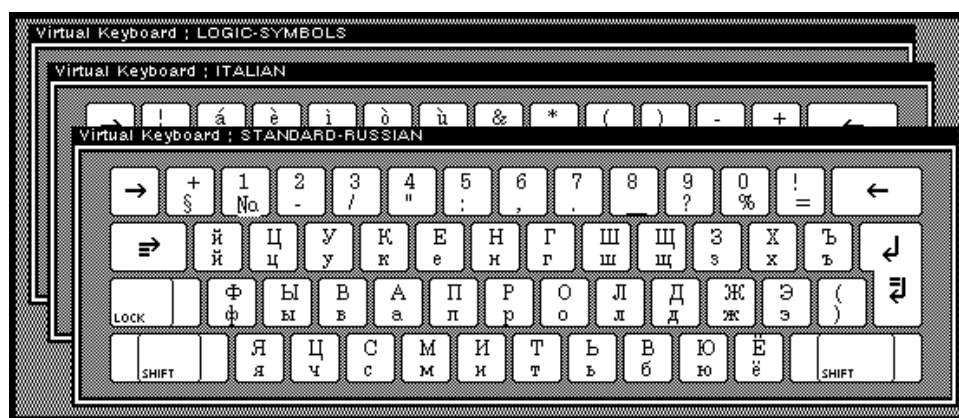


Figure 9. Keyboard layout display

## Display-Only a Keyboard

You can display the layout for any given virtual keyboard using the `DISPLAY ONLY` command. You will be offered a menu of the keyboards known to the program (such as above); select the one you want to display. This is useful if you are primarily using the standard English keyboard but need to type some characters in other languages, or some special characters such as mathematical symbols.

You can use the displayed image as a menu: Selecting a key from the image with the left mouse button will send the character assigned to that key, and pressing the shift key while you click on a key will send the shifted character. Middle-clicking also sends the shifted character.

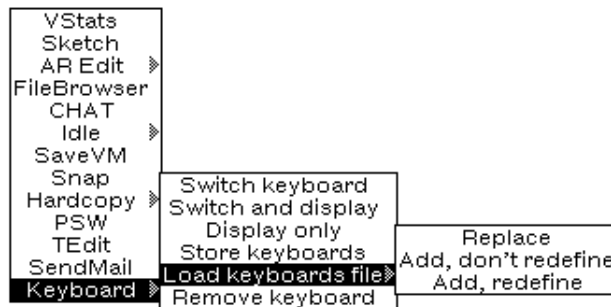
The effect is exactly as if you had pressed a key with that character assigned to it (except that interrupt characters are treated as ordinary characters; i.e., they do not cause an interrupt). The character is sent to the process that has the TTY (usually where the caret is flashing).

## Store Keyboards

After you edit a keyboard (using the `KeyboardEditor` module), you can store it using the `STORE KEYBOARDS` command in the top-level menu. When you select `STORE KEYBOARDS`, the system will prompt you for a file name. After you type the file name into the prompt window, the system will store all the keyboards known to it (both new and old) in that file in a form that will enable it to load them.

## Loading Keyboards File

To load a keyboards file, choose the `LOAD KEYBOARDS FILE` command, then slide the mouse cursor to the right and choose one of the three items in its submenu.



## Replace All Known Keyboards

Choosing REPLACE will load a set of keyboards that you stored using the STORE KEYBOARDS command, replacing all the keyboards currently known to the system.

The currently known keyboards will be lost.

## Add New Keyboards to the List of Known Keyboards

To add new keyboards without replacing any of the currently known keyboards, select ADD--DON'T REDEFINE. This will load a set of keyboard definitions.

If a keyboard in the file has the same name as one that is already known to the system, that keyboard will not be loaded and the current definition will stay in effect.

## Load New Keyboards and Redefine Existing Keyboards

The ADD--REDEFINE command is similar to ADD--DON'T REDEFINE, except that it redefines existing keyboards that have the same name as keyboards on the file.

Currently known keyboards that do not have the same name as newly loaded keyboards will remain in the list of known keyboards.

## Removing Keyboards From the Menu

To remove a keyboard from the set of currently known keyboards, select the REMOVE KEYBOARD command. This will pop up a menu of the known keyboards (such as above), from which you can select a keyboard to be deleted.

---

## Defining a Virtual Keyboard

A virtual keyboard is a list whose CAR is the name of the keyboard and whose CDR is a list of key actions. Creating a new virtual keyboard can be done directly in Lisp or interactively, using the KeyboardEditor module.

## Using the Functional Interface

The list of keyboards that are known to the program appears in the menu of keyboard names that pops up when you select SWITCH KEYBOARD from the background menu. This list is stored in the global variable VKBD.KNOWN-KEYBOARDS (see below). To add a keyboard to the list, you have to define that keyboard. To define a keyboard you can



either call the function `DEFINEKEYBOARD` or manipulate the variable `VKBD.KNOWN-KEYBOARDS` directly as explained herein.

You may also use the `KeyboardEditor` module, which provides a menu-based user interface for creating and changing keyboard layouts.

A virtual keyboard is a list of the form

```
(KEYBOARD-NAME KEY-ASSIGNMENT1 KEY-ASSIGNMENT2 . . .)
```

A *KEY-ASSIGNMENT* is a list of the form

```
(KEY (UNSHIFTED-CHAR SHIFTED-CHAR LOCK/UNLOCK))
```

*KEY* is a key name (the character that appears on the actual keyboard).

*UNSHIFTED-CHAR* and *SHIFTED-CHAR* are character codes. Each can be either an integer representing the actual code or a list of two elements: the number of the character set and the number of the character in the set.

*LOCK/UNLOCK* is either the atom `LOCKSHIFT`, in which case *SHIFTED-CHAR* will be transmitted when the shift-lock key is down, or `NOLOCKSHIFT`, in which case the shift-lock key has no effect on that key. *LOCK/UNLOCK* is `LOCKSHIFT` by default.

```
(DEFINEKEYBOARD KEYBOARD-NAME LIST-OF-KEY-ASSIGNMENTS KEYS-ARE-
NUMBERS?) [Function]
```

Creates a new virtual keyboard after parsing the list of key assignments and adds the keyboard to the list of known keyboards.

If *KEYS-ARE-NUMBERS?* is `T`, the function expects to find key numbers instead of key names.

```
(SWITCHKEYBOARDS NEW-KEYBOARD SWITCH-FLG DISPLAY-FLG MENU-
POSITION) [Function]
```

Switches the current keyboard to *NEW-KEYBOARD*, where *NEW-KEYBOARD* is either a virtual keyboard or the name of a known keyboard.

If *SWITCH-FLG* is non-`NIL`, the actual key actions of the keyboard will be modified.

If *DISPLAY-FLG* is non-`NIL`, a window with a menu will be displayed. This displayed keyboard will act as a menu and will send characters to the current input stream when a character is selected.

```
VKBD.KNOWN-KEYBOARDS [Variable]
```

Contains the list of all currently known virtual keyboards.

## Limitations

After loading the Dvorak keyboard, and then restoring defaults, you lose the shift-lock key.

[This page intentionally left blank]

# WHERE-IS

---

This is a new implementation of a facility similar to but not compatible with the Lyric library module WhereIs. Where-Is indexes all definers, but WhereIs only indexed Interlisp FNS definitions.

## Requirements

---

Hash-File and Cash-File.

## Installation

---

Load WHERE-IS.DFASL and the required .DFASL modules from the library.

## Changed File Manager Functions

---

Where-Is allows the file manager to know of many more definitions than are actually in the files which have been noticed. To achieve this behavior, the following file manager functions are changed when Where-Is is loaded.

Both of these functions are called by the edit interface (the function cl:ed). Thus when Where-Is is loaded the contents of its databases are known to the editor.

(il:whereis *name type files filter*) [Function]

Performs as described in the *Interlisp Reference Manual*. Returns the subset of *files* that contain a *type* definition for *name*. *Files* defaults to `il:filelst` (all noticed files). When Where-Is is loaded and `il:whereis` is passed `t` as its *files* argument, `il:whereis` looks in the Where-Is databases.

(il:typesof *name possible-types impossible-types source filter*) [Function]

Performs as described in the *Interlisp Reference Manual*. Returns the subset of *possible-types* that *name* is defined as. *Possible-types* defaults to `il:filepkgtypes` (all define types). When Where-Is is loaded, `il:typesof` also includes the types for *name* in its databases.

## Databases

---

Where-Is provides functions to use and build databases.

### Using a Database

(xcl::add-where-is-database *pathname*) [Function]

Adds the database in the file named by *pathname* to the databases known to Where-Is. If a database on an older version of this file is already known, then Where-Is will start using the new version.

(xcl::del-where-is-database *pathname*) [Function]

Deletes the database named by *pathname* from the databases known to Where-Is.

xcl::\*where-is-cash-files\* [Variable]

Contains the list of databases known to Where-Is.

There is a proceed case for errors while accessing a database which will delete the offending database. This can be useful when a file server goes down.

## Building a Database

(xcl::where-is-notice *database-file* &key *files new hash-file-size temp-file-name*  
*define-types quiet*) [Function]

Records the definers on files in the file named by *database-file*.

Files can be a *pathname* or a list of *pathnames*. The default for *files* is "*\* . ;*". Note that it is important to include the trailing semicolon so that only definers on the most recent version are recorded.

If *new* is true, a new database file is created, otherwise *database-file* is presumed to name an existing Where-Is database to be augmented. The default for *new* is nil.

*Hash-file-size* is only used when *new* is false and is passed as the *size* argument to *make-hash-file*. The default for *hash-file-size* is *xcl::\*where-is-hash-file-size\**, which has a default top-level value of 10,000.

If *temp-file-name* is provided, all changes happen in the temporary file named, which will afterwards be renamed to *database-file*. This can both make things faster (if the temporary file is on a faster device) and doesn't generate a new version of a database until the new version is ready to be used. The use of a temporary file may slow things down when a large existing database is just being updated to reflect a small number of changes.

*Define-types* is the list of define types (file package types) which should be recorded. The default define types are all those on *IL:FILEPKGTYPES* which are not aliases for others and which are not in the list *xcl::\*where-is-ignore-define-types\**.

Unless *quiet* is true, *xcl::where-is-notice* prints the name of each file as it is processed.

*xcl::where-is-notice* returns the *pathname* of the hash file written.

[This page intentionally left blank]

Before you can print the Medley LispUsers' tedit files you must do the following:

1. If you aren't running in Lispcore you need to load  
{eris}<tedit>teditcopy.lcom
2. In IL:\ASCIITONS change entry 183 so that it reads 183 instead of what it normally reads.

Doing these two things causes the Envos logo to print correctly. To print some of the LispUsers' modules you must have some functions loaded. The following table shows what you must have loaded to print certain LispUsers' modules:

| Module name             | File needed to print or display correctly                             |
|-------------------------|-----------------------------------------------------------------------|
| equation examples ..... | Mathtons.lcom<br>equations.lcom<br>and set il:   EQ.UseNSChars   to T |
| Doc-objects .....       | docobjects.lcom (loads dateformat.lcom)                               |
| hpgl.....               | sketch.lcom                                                           |

# LISPUSERS' MODULES MANUAL



Address comments to:  
Envos Corporation  
1157 San Antonio Road  
Mountain View, CA 94043

## LISPUSERS' MODULES MANUAL

309000

Medley Release

September, 1988

Copyright © 1988 by Envos Corporation.

All rights reserved.

Envos is a trademark of Envos Corporation

Xerox® is a registered trademark of Xerox Corporation.

UNIX® is a registered trademark of AT&T Bell Laboratories.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

---

The information in this document is subject to change without notice and should not be construed as a commitment by Envos Corporation. While every effort has been made to ensure the accuracy of this document, Envos Corporation assumes no responsibility for any errors that may appear.



TABLE OF CONTENTS

|                                        |     |
|----------------------------------------|-----|
| ACE .....                              | 1   |
| ADDRESSBOOK .....                      | 8   |
| AIREGIONS .....                        | 11  |
| ANALYZER .....                         | 19  |
| BACKGROUNDIMAGES.....                  | 20  |
| BACKGROUNDMENU .....                   | 23  |
| BITMAPFNS .....                        | 26  |
| CALENDAR .....                         | 27  |
| CANVASCONVERTER.....                   | 34  |
| CD .....                               | 36  |
| CHATEMACS .....                        | 38  |
| CHATSERVER.....                        | 40  |
| CHECKPOINT.....                        | 43  |
| CL-TTYEDIT .....                       | 44  |
| COMPARESOURCES .....                   | 45  |
| COMPARETEXT .....                      | 48  |
| COMPILEBANG .....                      | 50  |
| COURIERDEFS .....                      | 51  |
| COURIEREVALSERVE .....                 | 52  |
| COURIERIMAGESTREAM.....                | 53  |
| COURIERSERVE.....                      | 55  |
| CROCK .....                            | 58  |
| DATEFORMAT-EDITOR .....                | 60  |
| DEFAULTSUBITEMFN.....                  | 62  |
| DIGI-CLOCK.....                        | 63  |
| DOC-OBJECTS .....                      | 65  |
| DONZ .....                             | 68  |
| DSPSCALE.....                          | 70  |
| EDITBG.....                            | 72  |
| EDITKEYS .....                         | 73  |
| EQUATIONS .....                        | 74  |
| EQUATION EDITOR PROGRAMMERSGUIDE ..... | 77  |
| EQUATIONEXAMPLES.....                  | 85  |
| ETHERBOOT.....                         | 87  |
| FILEWATCH .....                        | 89  |
| FILLREGION.....                        | 92  |
| FTPSERVER-MULTI-CONNECTIONS.....       | 94  |
| GRAPHCALLS .....                       | 95  |
| GREP .....                             | 100 |
| GRID-ICONS.....                        | 101 |

|                         |     |
|-------------------------|-----|
| HANOI.....              | 104 |
| HASHBUFFER.....         | 105 |
| HASHDATUM .....         | 107 |
| HEADLINE .....          | 108 |
| HPGL .....              | 109 |
| IDLEHAX.....            | 111 |
| INSPECTCODE-TEDIT ..... | 113 |
| KEYOBJ.....             | 115 |
| KINETIC .....           | 116 |
| KOTOLOGO .....          | 117 |
| LIFE.....               | 118 |
| LOADMENUITEMS.....      | 119 |
| LOGIC .....             | 122 |
| LOOKUPINFILES .....     | 130 |
| MAGNIFIER .....         | 132 |
| MAKEGRAPH.....          | 133 |
| MANAGER.....            | 137 |
| MATHTONS .....          | 143 |
| MICROTEK .....          | 144 |
| MONITOR .....           | 152 |
| NEATICONS .....         | 154 |
| NOTEPAD.....            | 158 |
| NSCOPYFILE .....        | 161 |
| NSDISPLAYSIZES .....    | 162 |
| NSPROTECTION.....       | 164 |
| PAGEHOLD .....          | 169 |
| PIECE-MENUS .....       | 172 |
| PLOT .....              | 174 |
| PLOTEXAMPLES .....      | 188 |
| PLOTOBJECTS .....       | 189 |
| PLOTOBJECTS1 .....      | 194 |
| POSTSCRIPT .....        | 196 |
| PS-SEND .....           | 200 |
| PS-TTY.....             | 201 |
| PREEMPTIVE .....        | 202 |
| PRESSFROMNS.....        | 203 |
| PRETTYFILEINDEX .....   | 206 |
| PRINTERMENU .....       | 215 |
| PROGRAMCHAT.....        | 216 |
| PROMPTREMINDERS .....   | 217 |
| PROOFREADER .....       | 220 |
| QEDIT .....             | 223 |

|                           |     |
|---------------------------|-----|
| READAIS .....             | 226 |
| READAPPLEFONT .....       | 228 |
| READBRUSH .....           | 229 |
| READDATATYPE.....         | 230 |
| READDISPLAYFONT .....     | 231 |
| REGION .....              | 232 |
| REMOTEPSW .....           | 234 |
| RPC.....                  | 235 |
| SCREENPAPER .....         | 244 |
| SEDI-MENU-ALWAYS .....    | 245 |
| SETDEFAULTPRINTER.....    | 246 |
| SHOWTIME.....             | 247 |
| SIMPLECHAT .....          | 249 |
| SOLID-MOVEW .....         | 250 |
| SOLITAIRE .....           | 252 |
| STARBG .....              | 253 |
| STEP-COMMAND-MENU .....   | 255 |
| STORAGE.....              | 256 |
| SYSTATS.....              | 258 |
| TALK.....                 | 259 |
| TCPTIME.....              | 265 |
| TEDITKEY.....             | 267 |
| TILED-SEDI .....          | 273 |
| TRAJECTORY-FOLLOWER ..... | 275 |
| TRICKLE.....              | 276 |
| TURBO-WINDOWS .....       | 277 |
| TWODGRAPHICS .....        | 281 |
| UNBOXEDOPS.....           | 285 |
| UUENCODE.....             | 288 |
| VSTATS .....              | 289 |
| WDWHACKS .....            | 293 |
| WHO-LINE .....            | 294 |
| WHOCALLS.....             | 299 |
| XCL-BRIDGE .....          | 300 |

[This page intentionally left blank]

{ERIS}<LISPUSERS>LYRIC>LISPUSERS-DEPENDENCIES.TEDIT  
Edited 12/16/86 Susana Wessling

This is a list of the LispUsers files that are to be released and their dependencies. Please add new LispUsers files to this list when you write them onto <lispusers>lyric>. If any of the packages on this list have changed or are not to be released, please remove them.

Key:

\* internal release only

+ release source AND dcom files

indented: dependent on the above package

access.;1  
ACE.;1  
    ACE-APPLEDEMO.;1  
    ACE-BOUNCINGBALL.;1  
    ACE-FOUETTE.;1  
ACTIVEREGIONS.;1  
ACTIVEREGIONS2.;1  
AIREGIONS.;1  
    AIREGIONS-DEMO.;1  
animate.;1  
\*ARCHIVETOOL.;1  
ARITHDECLS.;1  
ARRAYSORTER.;1  
AUTOSAMEDIR.;1  
AUXMENU.;2  
BACKGROUND.;1  
    +BACKGROUNDDEMO.;1  
BACKGROUNDIMAGES.;3  
    +BACKGROUND-\*.BITMAP  
    +BACKGROUND-\*.PRESS  
BackgroundMenu.;1  
BICLOCK.;1  
BIZGRAFIX.;1  
bltdemo.;1  
BMFROMW.;1  
BOUNCE.;1  
BOUNDARY.;1  
BOYERMOORE.;1  
    BOYERMOOREDATA.;1  
BQUOTE.;1  
    BQUOTE GACHA10-C0.DISPLAYFONT  
    BQUOTE GACHA12-C0.DISPLAYFONT  
    BQUOTE GACHA8-C0.DISPLAYFONT  
BTMP.;1  
    BTMP-DEBUG.;1  
BUGREPORT.;1  
CALENDAR.;2  
    PROMPTREMINDERS.;1  
CCACHE.;1  
CD.;1  
CD-COMMAND.;1  
CHANGEPRINTER.;1  
+CIAPROPOS.;1  
COLORNNGS.;1  
COMHACK.;1  
COMPAREDIRECTORIES.;1  
COMPARESOURCES.;1  
COMPARETEXT.;1  
COMPILEFORMSLIST.;1  
CONNTITLE.;1  
COURIERDEFS.;1  
COURIEREVALSERVE.;1  
COURIERIMAGESTREAM.;1  
COURIERSERVE.;1  
    REMOTEGRAPHER.;1  
    REMOTEPSW.;1  
CROCK.;1  
\*CRYPT.;1  
DATEFNS.;1  
DEDITAU.;1  
DEDITICON.;1 (no .dcom)  
DEDITK.;1  
DEFAULTICON.;1  
DEFAULTSUBITEMFN.;1  
DINFOEDIT.;1  
DIRECTORYTOOLS.;1  
DIRGRAPHER.;1  
DIRMENU.;1

```

DLIONFNKEYS.;1
DOCTOR.;1
DONZ.;1
*DRAFFILE.;1
DSL.;1
dumper.;1
+DUMpload.;1
EDITBG.;1
EDITFONT.;1
+EDITKEYS.;1
EDITRECALL.;1
EMACS.;1
EMACUSER.;1
EQUATIONFORMS.;1
EQUATIONS.;1
EXEC.;1
+EXECFNS.;1
FACEINVADER.;1
FASTBITMAPBIT.;1
*FILECACHMSGWINDOW.;1
+FILEOBJ.;1
FILEPERCENT.;1
FILLPRINT.;1
FILLREGION.;1
FINGER.;1
FLAGBROWSER.;1
FLOPPY4.;3
FONTMENU.;1
FULLSCREEN.;1
GKS.;1
GKSEXTERN.;1
GKSINTERN.;1
GKSMATRIX.;1
GLISPA.;1
 GLISPB.;1
 GLISPDWINDOW.;1
 GLISPGEV.;1
 GLISPGEVAUX.;1
 GLISPGEVTYPE.;1
 GLISPR.;1
 GLISPTTEST.;1
 GLISPVECTOR.;1
graphcalls.;1
*GREP.;1
HANOI.;1
HASHBUFFER.;1
HASHDATUM.;1
HEADLINE.;1
HISTMENU.;1
IDEASKETCH.;1
IDLEHAX.;1
IMAGEWRITER.;1
*IMTEDIT.;1
 IMNAME.;1
 IMTOOLS.;1
 IMTRAN.;1
INSPECTCODE-TEDIT.;1
IRISCONSTANTS
 IRISDEMOFNS
 IRISIO
 IRISLIB
 IRISNET
 IRISSTREAM
 IRISVIEW
 LOADIRIS
 SFFONT
 GACHAE.LC1-SF;2
 GACHAE.LC2-SF;2
 GACHAE.NUM-SF;2
 GACHAE.S1-SF;2
 GACHAE.S2-SF;2
 GACHAE.UC1-SF;2
 GACHAE.UC2-SF;2
JARGON.;1
 JARGON.DB
KAL.;1
KEYOBJ.;1
KINETIC.;1
*LAFITEHIGHLIGHT.;1
LCROCK.;1

```

```

LIFE.;1
LISTEN.;1
LoadPatches.;1
LOGOCLOCK.;1
LSET.;1
MACWINDOW.;1
MAGNIFIER.;1
magnifyw.;1
*MAILMAT.;1
MAKEGRAPH.;1
MANAGER.;1
MATHFNS.;1
*MESATOLISP.;1
MOVE-WINDOWS.;1
multimenu.;1
MULTIW.;1
MUSICKEYBOARD.;2
NOTEPAD.;1
 NOTEPAD-CORESTYLES.;1
NQUEENS.;1
PACMAN.;1
PAGEHOLD.;1
PARSER.;1
PARSERG.;1
PATCHUP.;1
PCDAC.;1
PEANO.;1
PERFORMTRAN.;1
PIECE-MENUS.;1
PLAY.;1
PLOT.;2
 PLOTEXAMPLES.;1
 PLOTOBJECTS.;1
PLURAL.;1
+PQUOTE.;1
PREEMPTIVE.;1
PRESSFROMNS.;1
PRESSTOIP.;1
 DPRESS.;1
PRINTERMENU.;1
Proofreader.;1
 ANALYZER.;1
 SpellingArray.;1
PULLDOWNMENUS.;1
+QIX.;1
*READBRUSH.;1
RECORDPRINT.;1
REGION.;1
RESETMACROS.;1
ROTATEBM.;1
*RPC.;1
 RPC-EXAMPLE.;1
 RPC-EXAMPLECLIENT.;1
 RPC-EXAMPLESERVER.;1
 RPC-EXAMPLEUSER.;1
 RPC-LUPINE.;1
 RPCEVAL.;1
 RPCEVALCLIENT.;1
 RPCEVALSERVER.;1
*SAMPLER.;1
SERVERSTATUS.;1
SETDEFAULTPRINTER.;1
SETF.;1
SHOW.;1
SIGNAL.;1
SINGLEFILEINDEX.;1
SLIDEPROJECTOR.;1
SNAPSCROLL.;1
SOLITAIRE.;1
SPACEWINDOW.;1
sprint.;1
STARBG.;1
STOCKICONS.;1
STYLESHEET.;1
+SUPERMENUEDIT.;1
+SUPERMENUS.;1
SYSTAT.;1
+TEDITKEY.;1
THERMOMETER.;1
 THERMOMETERDEMO.;1

```

```

TILEDEDIT.;1
TIMEPANEL.;1
+TINYTIDY.;1
TMENU.;1
TOGMENU.;1
TRACEIN.;1
TRAJECTORY-FOLLOWER.;1
 TRANSOR.;1
TRUEHAX.;1
TSET.;1
TTY.;1
ttyio.;1
TURING.;1
TWOD.;1
TWODGRAPHICS.;1
UNBOXEDOPS.;1
utilisoprs.;1
VMEMSTATE.;1
VSTATS.;1
+WAM.;1
WDWHACKS.;1
WINK.;1
WINNER.;1
WORM.;1
+YAPFF.;1

```

Other TEDIT files: [on {eris}<lisp>koto>lispusers> or {eris}<lispusers>koto>]

```

Release-intro.tedit intro for released doc's
Release-rules.tedit rules for released doc's
Internal-intro.tedit intro for internal doc's
Internal-rules.tedit rules for internal doc's
blankpage.tedit blank page for printing screwups...
documentationtemplate.tedit long template for doc's
easytemplate.tedit short template for doc's
lispusers-rules.tedit old LU rules
RELEASE-INFO.tedit This document
LISPUSERS.TEDIT Short package summaries

```



Subject: Re: Lispusers packages (updates)  
In-reply-to: bloomberg.pa's message of 16 Dec 86 12:38 PST  
To: bloomberg.pa  
cc: LispUsers↑.x  
Reply-to: Wessling.pa  
Format: TEdit

I received this inquiry today:

-----  
Date: 16 Dec 86 12:38 PST  
From: bloomberg.pa  
Subject: Lispusers packages (updates)  
To: wessling.pa  
cc: bloomberg.pa

When I replace text or documentation of a Lispusers package with new versions, should I message you so that you can do the appropriate thing(s) vis-a-vis phylum?

--Dan  
-----

...and I thought this would be a good time to just give an update on how to handle the lispusers packages.

**--- Directories:**

{eris}<lisp>koto>lispusers> is where the frozen, released packages for Koto were put.  
{eris}<lispusers>koto> is where the Koto packages that have been changed since release should be put.  
{eris}<lispusers>lispcore> is where new packages that run in LispCore and updates of old packages that are updated to Lispcore should be kept.

**--- Creating new packages:**

if you create a new package that runs in LispCore, i.e. to be release with Lyric, write it onto {eris}<lispusers>lispcore>.

**--- Editing old packages:**

if you want to change a package that was released with Koto, make changes to the latest version (if it hasn't changed since release this is on {eris}<lisp>koto>lispusers> or if it has been updated, it should have been put on {eris}<lispusers>koto>) and copy it to:  
{eris}<lispusers>koto> if it **only** still runs in Koto  
{eris}<lispusers>Lispcore> if you **know** it runs in Lyric.

**--- What should be included:**

No package will be released from LispUsers without:  
- Source file  
- Compiled file

- Documentation

Documentation can be made through the template (`{eris}<lispusers>LispCore>easytemplate.tedit`) or by using the new `lispusers` package, `DOCUMENT` ( on `{eris}<lispusers>koto>`).

**--- for more information:**

`{eris}<lispusers>lispcore>LispUsers-rules.tedit`, `LispUsers-Summary.tedit`, `LispUsers-dependencies.tedit`. I will store this on `{eris}<lispusers>lispcore>lispusers-info-msg.tedit`.

**--- Nice things to do:**

When you create a new package, please:

- send a note to `LispUsers`↑ announcing it.
- add it to `LispUsers-summary.tedit` (a list of all the `lispusers` packages)
- add it to `LispUsers-dependencies.tedit`, which is a list of all the `lispusers` packages and which files they need to run.
- Edit `LispUsers-dependencies.tedit` to reflect the current state of `Lispcore` `lispusers` files (right now it is just the info from `Koto`).

If you created a package for `Koto`, please try to update it to `Lispcore` and store it onto `<lispusers>lispcore>`. If the package is un-updatable or useless in `LispCore`, please take it off of the aforementioned lists and/or message me about it.

**--- and to answer the question that started all this:**

Just put your packages on the directories mentioned. Any copying will be done.

Thank you very much for your cooperation....

Susana...

Date: 23 Jan 87 17:21 PST  
From: Wessling.pa  
Subject: {eris}<lispusers>lyric> exists!  
To: LispUsers↑.x  
Really-from: The Tired Floppy Lady (let's all go home... it's Friday...)  
Reply-to: Wessling.pa

NOTICE: If you are either a user or a creator/maintainer or ANY LispUsers module, please please please do not delete this message. I am sick of resending information...

All files that existed on {eris}<lispusers>lispcore> as of NOW have been moved to {eris}<LispUsers>Lyric>.

From now on, follow these rules for storing new or changed LispUsers modules:

- If the module was created or changed in Koto, please store it under {eris}<lispusers>koto>.
- If the module was created or changed in the {eris}<lyric>basics> sysout, please store it on {eris}<lispusers>lyric>.
- If the module was created or changed in the {eris}<lispcore>next> sysout, please store it on {eris}<lispusers>lispcore>.

Though this may seem a bit complicated, it will help us to keep track of which modules are assured of working where.

\*\*\*\*\*

#### RULES:

AND NOW... I am once again going to remind you of how you can make our lives a bit easier when interacting with LispUsers:

All modules that are stored on ANY of the aforementioned directories MUST have:

- source file
- compiled file (.LCOM or .DFASL for Lyric, .DCOM for Koto)
- documentation

If you do not store any of the above with your module, it WILL NOT be released.

In writing documentation, you can use EasyTemplate.tedit (stored on any/all LU directories) to make life easier. For a long version, see DocumentationTemplate.tedit. Please please please note in your documentation:

- any dependencies between modules
- if the module is for internal use only

For further and more detailed rules, see Lispusers-rules.tedit.

If you are nice, I would appreciate your updating all of the <lispusers>lispusers\* files. These files contain information about releasable modules that is very important. You should look into these files if

- your module requires other modules to be loaded with it.
- your module is for internal release only
- your module is new

\*\*\*\*\*  
 QUESTIONS/REMINDERS:

It has been brought to my attention that there are some modules that are stored on {eris}<lispusers>lyric> that are definitely NOT lyric compatible. Please do not do this. It will not make anyone's job easier. Such modules should go on <lispusers>koto>. Please move them if you are responsible for such horrific actions!

If you are responsible for or just have a vested interest in some module that is in <lispusers>koto> but has not been updated, please see to getting it updated and put into <lyric> or <lispcore>. If it is not done, it will not go out with Lyric.

For those who are anxiously waiting for an NS LispUsers directory... I'm working on it. I will send out further information if it happens.

\*\*\*\*\*  
 THAT'S ALL FOLKS...

Sorry for the long message, but these things just had to be said....

Susana

—End of message—

---

### LISP USERS' RULES

---

This document describes the rules and procedures for Xerox Lisp "Lisp Users'" modules. This document is mainly for Lisp Users' module writers, but users should also understand the rules. The deadline for Medley Lisp Users' module submissions is August 5, 1988.

#### **DEVELOPING A LISP USERS' MODULE**

A Lisp Users' module is a useful program made available to the general Xerox Lisp programming community. Neither the author nor the custodian of the Lisp Users' directory imputes any warranty of suitability or responsibility for errors.

Lisp Users' modules should be easily distinguishable from released library modules. In particular, this means that a Lisp Users' module may not have the same name as a Library module and should be visibly different. Lisp Users' modules derived from released software should be announced to the public only after communicating with the organization responsible for that released software.

Testing is important. If you make significant changes to a Lisp Users' module, enlist developers at your site as alpha testers. Avoid having to rerelease a package within hours of its announcement because of fatal bugs. A Lisp Users' module is not shoddy software; it is software made available outside the regular release channels.

Try not to flood your user community with a constant flow of new versions (or messages) so your messages can be appreciated rather than discarded without reading. If your module is undergoing continual changes, adopt a release strategy of regularly spaced, well tested releases. Your users will thank you.

#### **LISP USERS' MODULE OWNERSHIP**

A module submitted for Lisp Users' remains the "property" of the submitter. Others may not make changes, except for their own private use, without negotiating with the owner (who may already be making similar or incompatible changes).

As the owner of a module, you are not required to fix bugs, but if not, you must be willing to transfer ownership (permanently) to someone who volunteers to fix them. Ownership may pass back and forth among several people as long as they agree.

Like all software developed at Xerox, Lisp Users' modules are officially the property of Xerox. You should run with the COPYRIGHT option set to produce an appropriate Xerox copyright in the source.

A Lisp Users' module may become so useful that it becomes part of a standard Xerox Lisp release and is thereafter supported. Ownership then passes to the Xerox product organization.

#### **SUBMITTING LISP USERS' MODULES**

If you are not an internal user, you should submit your new module to Xerox either through e-mail or on a floppy. External users should make sure that they include all relevant information, such as documentation containing an e-mail or US mail address where he/she can be reached. External users are also held responsible for the support of their modules.

#### **SUBMITTING FILES TO LISP USERS'**

As with released software, it is important to submit not just the resulting product, but all the files needed to build and maintain a Lisp Users' module:

1. the runnable compiled file ( .LCOM or .dfasl)
2. documentation describing it, following the set formatting rules (see below)
3. a source file that can be released (optional)
4. data files, if needed

Packages submitted once are released once. Do not assume that a package submitted in one release will be automatically released in subsequent releases.

### **DOCUMENTATION**

Documentation, essential to any module being used effectively, should be put on the submitted floppy. No modules will be released without documentation. Documentation can be as simple as a paragraph describing what the module does and how to use it, or it can be as extensive as a dozen-page user manual. All modules should have a file with a .TEDIT extension. Formatting should be done according to the rules outlined in the appropriate (standalone or networked user) Lisp Users' Template. All users, external users included, should follow the Lisp Users' Template rules. If the documentation is large and formatting time consuming, you can also produce an interpress file (with the .ip extension), as well as submitting a .Tedit file. (Be sure to update the interpress file if you update the documentation!) Documentation should include the full electronic mail address of the submitter.

### **COMPATIBILITY WITH LISP USERS'**

Any submitted Lisp Users' files should be compilable in a "vanilla" environment. The file itself should load in any auxiliary modules under a suitable (DECLARE: EVAL@COMPILE -- ) when necessary.

Thanks for your cooperation.

## Lispusers Summary

A one-line description of (most) LispUsers packages.

All files listed here should be in {eris}<lispusers>lispcore>. If you want to look at the list for Koto files, see <lispusers>koto>, the same file. Please add any new files you create to this list.

---NO TEDIT FILE means that no documentation is needed.  
---UNSUPPORTED means that there is no author or the author is unreachable.  
---an asterisk before a file means its part of the previous package.  
---INTERNAL means that it is not for release outside of Xerox  
---SOURCE means that you load the source & not the DCOM  
---Two dashes (--) before it means that it exists in Koto but hasn't been updated for lyric yet.

| Module                                                                                                    | Comments                                                     |
|-----------------------------------------------------------------------------------------------------------|--------------------------------------------------------------|
| --ACCESS                                                                                                  |                                                              |
| --Ace                                                                                                     | Animation Compiler; includes graphics editor.                |
| --* ACE-EDIT, ACE-Main, ACE-Prim, ACE-AppleDemo (Source), Ace-BouncingBall (Source), ACE-Fouette (Source) |                                                              |
| --ActiveRegions                                                                                           | Mouse sensitive regions in a window.                         |
| --ActiveRegions2                                                                                          | Rewrite of ActiveRegions                                     |
| --AIRegions                                                                                               |                                                              |
| --* AIREGIONS-DEMO (Source)                                                                               |                                                              |
| --Analyzer                                                                                                | Part of proofreader                                          |
| --Animate                                                                                                 | Smoothly move arrow, finger around on screen.                |
| --ArithDecls                                                                                              | For use with decl                                            |
| --ArchiveTool                                                                                             | (INTERNAL) Cedar archive system interface                    |
| --ArraySorter                                                                                             |                                                              |
| --AutoSameDir                                                                                             | Put sources back where you got them at MAKEFILE              |
| --Auxmenu                                                                                                 | Useful middle-button menu in background                      |
| --Background                                                                                              | BITBLT to screen background or obscured windows.             |
| --* BACKGROUNDDEMO                                                                                        |                                                              |
| --BackgroundImages                                                                                        | screen backgrounds, including rhine, two dollar bill, castle |
| BackgroundMenu                                                                                            | Useful menu in the background                                |
| BiClock                                                                                                   | Swedish clock                                                |
| BITMAPFNS                                                                                                 |                                                              |
| --BizGrafix                                                                                               | Pie & bar chart, line graph creation                         |
| --BLTDemo                                                                                                 | graphics demo/idle hack                                      |
| --BMFromW                                                                                                 | bitmap from window                                           |
| --Bounce                                                                                                  | Idle hack with lines                                         |
| --Boundary                                                                                                |                                                              |
| BoyerMoore                                                                                                | The Boyer-Moore Theorem Prover                               |
| --* BoyerMooreData (Source)                                                                               |                                                              |
| --BQuote                                                                                                  | Yet another backquote macro (like Common Lisp's)             |
| BSEARCH                                                                                                   |                                                              |
| --BTMP                                                                                                    | Basic Text Macro Processor                                   |
| --* BTMP-Debug                                                                                            |                                                              |
| --BugReport                                                                                               |                                                              |
| --Calendar                                                                                                | calendar/appointment-reminder program                        |
| --CCache                                                                                                  | keep files on local disk, like FILECACHE but less automatic  |
| CD                                                                                                        | Connect-to-directory command                                 |
| --CD-Command                                                                                              | Connect-to-directory command                                 |
| --Changeprinter                                                                                           | Interface for dealing with printers                          |
| CHATSERVER                                                                                                |                                                              |
| CHECKSET                                                                                                  |                                                              |
| --CIApropos                                                                                               | Case Independent apropos                                     |
| CIRCLPRINT                                                                                                |                                                              |
| --ColorNNGS                                                                                               | For use with busmaster and Number 9 Graphics Card on 1108    |
| --ComHack                                                                                                 | Comments in if and fors                                      |
| COMMENTSTRINGS                                                                                            |                                                              |
| COMMON-MAKEFILE                                                                                           | Creates plain-text exportable Common Lisp source code files  |
| COMMWINDOW                                                                                                |                                                              |
| --CompareDirectories                                                                                      | What files are different on two directories                  |
| CompareSources                                                                                            | Compare two lisp files                                       |
| --CompareText                                                                                             | Compare two text files                                       |
| COMPILEBANG                                                                                               |                                                              |
| --CompileFormsList                                                                                        |                                                              |
| --ConnTitle                                                                                               | Show where I'm connected                                     |
| --CourierDefs                                                                                             | Implementation of XNS protocol for remote procedure call     |
| --CourierEvalServe                                                                                        | EVAL server using courier                                    |
| --CourierImageStream                                                                                      | Image stream server using courier                            |
| CourierServe                                                                                              | courier server                                               |
| --Crock                                                                                                   | Analog clock                                                 |
| --Crypt                                                                                                   | (INTERNAL)                                                   |
| --Datefns                                                                                                 |                                                              |
| --DEditAug                                                                                                |                                                              |
| --DEditHardcopy                                                                                           | (NEW) hardcopy of Dedit window prettyprints whole function   |

|                     |                                                        |
|---------------------|--------------------------------------------------------|
| --DEditIcon         | An icon for Dedit                                      |
| DEditK              | New DEdit menu with common combo's                     |
| --DefaultIcon       | new default icon for shrunk windows                    |
| --DefaultSubitemFN  |                                                        |
| DES                 |                                                        |
| DIGEST              |                                                        |
| DInfoEdit           |                                                        |
| --DirectoryTools    |                                                        |
| --Dirgrapher        | Shows a graph of directory structure                   |
| DLionFNKeys         | Dandlion keys (center/bold/etc) for 1132 users         |
| DMSG                |                                                        |
| --Doctor            | Infamous Eliza program (computer shrink)               |
| --Donz              | talking windows                                        |
| --DPress            | (INTERNAL) take apart press files                      |
| DSKTEST             |                                                        |
| --DSL               | Digital Speech Lab, uses Busmaster speech analysis     |
| --Dumper            | (INTERNAL) Alto Exec DUMP files                        |
| --DumpLoad          | Alto Exec DUMP files                                   |
| --EditBG            | Edit background & background border shade              |
| --EDITFONT          | Create & edit display fonts                            |
| --EditKeys          | Dandlion keys (center/bold/etc) for 1132 users         |
| --EditRecall        |                                                        |
| EMACS               | EMACS commands on top of tedit                         |
| --EMACSUser         | Use EMACS as your programming environment              |
| --EquationForms     | TEdit file=EquationEditor.tedit; for editing equations |
| --Equations         | TEdit file=EquationProgram; programmatic interface     |
| --Exec              | Extra exec windows (like Listen)                       |
| --ExecFNS           | INTERNAL                                               |
| --FaceInvader       | A game                                                 |
| --FastBitmapBit     |                                                        |
| FILECACHE           |                                                        |
| FILECACHE-BROWSER   |                                                        |
| FILECACHE-DECLS     |                                                        |
| FILECACHE-HOSTUP    |                                                        |
| FILECACHE-SCAVENGE  |                                                        |
| FILECACHEMSGWINDOW  |                                                        |
| --FileObj           | Files as image objects                                 |
| --FilePercent       | Lanning                                                |
| FILEWATCH           |                                                        |
| --FillPrint         |                                                        |
| --FillRegion        |                                                        |
| --Finger            | Who else is running Finger on the net?                 |
| --FlagBrowser       |                                                        |
| --FontMenu          |                                                        |
| --FullScreen        |                                                        |
| --GKS               | Graphics Kernel Standard implementation                |
| --* GKSEextern      |                                                        |
| --* GKSIntern       |                                                        |
| --* GKSMatrix       |                                                        |
| --GLISP             | Compiler for GLisp, an object-oriented Lisp language   |
| --* GLISPA          |                                                        |
| --* GLISPB          |                                                        |
| --* GLISPDWINDOW    |                                                        |
| --* GLISPGEV        |                                                        |
| --* GLISPGEVAUX     |                                                        |
| --* GLISPGEVTYPE    |                                                        |
| --* GLISPR          |                                                        |
| --* GLISPTTEST.LSP  |                                                        |
| --* GLISPVECTOR.LSP |                                                        |
| --* GLISP.tty       |                                                        |
| --GraphCalls        | Graph calls from interpreted, compiled code            |
| --GREP              | NEW. Search file(s) for strings, e.g. phone book.      |
| --Hanoi             | Graphics demo/idle hack                                |
| --HashBuffer        |                                                        |
| --HashDatum         |                                                        |
| --Headline          | Big titles on the screen                               |
| --HistMenu          | History list as a menu                                 |
| --IdeaSketch        |                                                        |
| IdleHax             | collection of idle hacks                               |
| --IdleSwap          | (NEW) idle hack                                        |
| --ImageWriter       | output to Apple image writer                           |
| --IMName            | (INTERNAL) edit Interlisp Manual                       |
| --IMTedit           | (INTERNAL) convert Interlisp Manual to TEdit           |
| --IMTools           | (INTERNAL) Tools for dealing with Interlisp Manual     |
| --IMTran            | (INTERNAL) Translate Interlisp Manual                  |
| --InspectCode-TEdit |                                                        |
| INVISIBLEWINDOW     |                                                        |
| --Jargon            | definitions from the hacker's dictionary               |
| --Kal               | b/w or color kaleidoscope (also in Idlehax)            |
| --? KeyboardTool    |                                                        |



|                                 |                                                        |
|---------------------------------|--------------------------------------------------------|
| Keyobj                          | key image-objects                                      |
| --Kinetic                       | Graphics demo/idle hack                                |
| --LCrock                        | Clock in the logo window                               |
| Life                            | Conway's game of life, as an Idle hack                 |
| LISPIXCONVERT                   |                                                        |
| --Listen                        | Lisp Executives from the background menu               |
| --LoadPatches                   |                                                        |
| --LogoClock                     | Another clock in the logo window                       |
| --LSet                          | Lists as sets                                          |
| MacWindow                       | Shrink & expand windows like a MAC                     |
| --Magnifier                     | Magnify areas of the screen                            |
| --MagnifyW                      |                                                        |
| --MailoMat                      | INTERNAL                                               |
| --MakeGraph                     | help for Grapher users making graphis                  |
| Manager                         | Window/menu file package interface                     |
| MANDELBROT                      |                                                        |
| --MathFNs                       | trig, complex functions, constants                     |
| MERGE-FILEGEN                   |                                                        |
| --Move-Windows                  |                                                        |
| --MTP                           | (INTERNAL) Mail Transfer Protocol for Lafite<->Tops-20 |
| --MultiMenu                     | Attached menus in groups                               |
| --MultiW                        | Heirarchical window environment                        |
| --MusicKeyboard                 |                                                        |
| --Notepad                       | Graphics paint program                                 |
| --* NOTEPAD-CORESTYLES (SOURCE) |                                                        |
| NOVAFONT                        |                                                        |
| NSCHATSERVER                    |                                                        |
| NSDISPLAYSIZES                  |                                                        |
| NSMAINTAIN                      |                                                        |
| NSREADERPATCH                   |                                                        |
| --NSThasize                     | (INTERNAL NEW) Convert GV Distribution list to NS      |
| --NQueens                       | Graphics demo                                          |
| --Pacman                        | Game                                                   |
| --Pac-Man-Idle                  | (NEW) Idle hack                                        |
| PageHold                        | Changes "window full" behavior on scrollers            |
| --Parser                        | Parser generator for making new parsers                |
| --Patchup                       |                                                        |
| --PCDAC                         | A-to-D and D-to-A using BusMaster and PC boards        |
| --Peano                         | Graphics demo                                          |
| --Performtran                   | add clisp word to record package                       |
| --Piece-Menus                   |                                                        |
| PLANETS                         |                                                        |
| --Play                          | Tunes on 1108/1186 beeper                              |
| --Plot                          | Making plots                                           |
| --* PlotExamples                | Some samples                                           |
| --* PlotObjects                 | Plots as image objects in documents                    |
| --Plural                        | Plural of words                                        |
| --PQuote                        | Prettyprint (QUOTE x) as 'x.                           |
| --Preemptive                    | make scheduler preemptive (caveats)                    |
| --PressFromNS                   |                                                        |
| --PressToIP                     |                                                        |
| --PrinterMenu                   |                                                        |
| --PromptReminders               | reminders at a given time, used by Calendar            |
| ProofReader                     | Spelling checker in Tedit                              |
| PUPCHATSERVER                   |                                                        |
| --* SpellingArray               |                                                        |
| --PullDownMenus                 |                                                        |
| --QIX                           | Shrager                                                |
| --RecordPrint                   |                                                        |
| --Region                        |                                                        |
| --RemoteGrapher                 | Grapher over XNS connection on another machine         |
| --RemotePSW                     | Someone elses Process Status WIndow on your screen     |
| --ResetMacros                   |                                                        |
| --RotateBM                      | Rotate bitmaps                                         |
| --RPC                           | (Internal) Cedar-style PUP based Remote Procedure Call |
| --* RPC-Example                 |                                                        |
| --* RPC-ExampleClient           |                                                        |
| --* RPC-ExampleUser             |                                                        |
| --* RPC-Lupine                  |                                                        |
| --* RPCEVAL                     |                                                        |
| --* RPCEVALCLIENT               |                                                        |
| --* RPCEVALSERVER               |                                                        |
| RS232CHATSERVER                 |                                                        |
| --Sampler                       | Graphics demo                                          |
| SCREENPAPER                     |                                                        |
| --ServerStatus                  |                                                        |
| --SetDefaultPrinter             |                                                        |
| --SetF                          | Common Lisp style SETF                                 |
| --Show                          |                                                        |

|                       |                                                      |
|-----------------------|------------------------------------------------------|
| --Signal              | Mesa-style signals                                   |
| SingleFileIndex       | Add index to ListFiles output                        |
| --SlideProjector      | Cycle thru tedit file of "slides"                    |
| --SnapScroll          | Scrollable "snap" windows                            |
| --Solitaire           | Graphics demo/Idle hack                              |
| --SpaceWindow         | space allocation use in a window                     |
| --SPrint              |                                                      |
| --StarBG              | Stars in the background/Idle hack                    |
| --? StockIcons        |                                                      |
| --StyleSheet          | create block of menus                                |
| --SuperMenus          |                                                      |
| --* SuperMenuEdit     |                                                      |
| --Systat              | control-T puts up window w/graphic display           |
| TCPCHATSERVER         |                                                      |
| TEditKey              | New TEdit commands as various meta-characters        |
| --Thermometer         |                                                      |
| --* THERMOMETERDEMO   |                                                      |
| --TileDEdit           | Dedit windows place themselves so they don't overlap |
| --TimePanel           | 1108 Maintenance panel => clock                      |
| TinyTidy              | Move icons over to edge of screen                    |
| --TMenu               | menus that stuff input buffer, pull down menus       |
| --TogMenu             |                                                      |
| --TraceIn             | stepper/tracer for debugging                         |
| --Trajectory-Follower | animation of following a trajectory                  |
| --Transor             |                                                      |
| --* TSet              |                                                      |
| --TrueHax             |                                                      |
| --TTY                 |                                                      |
| --TTYIO               |                                                      |
| --Turing              | Turing machine simulator                             |
| --TwoD                | Two dimensional graphics package                     |
| --TwoDGraphics        | Two dimensional graphics package                     |
| UnboxedOps            | Unboxed arithmetic for dandetiger's                  |
| UPCSTATS              |                                                      |
| --UtilISopr           | Additional iterative operators                       |
| --VMemState           | Turn on/off VMEM.PURE.STATE                          |
| --VStats              | Show time, space used                                |
| --WAM                 | Window Action Menu                                   |
| --WDWHacks            |                                                      |
| WHO-LINE              |                                                      |
| WHOCALLS              |                                                      |
| --Wink                | Graphics demo                                        |
| --Winner              |                                                      |
| --Worm                | Idle hack                                            |
| --Yapff               | Yet Another Page Full Function                       |

---

---

>>MODULE NAME<<

---

---

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

>>Type INTERNAL here if the file is for Internal Use Only<<

This document last edited on >>DATE<<.

## INTRODUCTION

>>This paragraph should be replaced by an overview of your module. The attached Lisp Users' Template explains the documentation conventions to be used for each Lisp Users' modules.<<

## MODULE EXPLANATIONS

### >>Functions, Variables, and Lisp Code Examples<<

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE *DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN*) [Function]

The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system. Variables look like functions, except that the word "Variable," enclosed in **square brackets**, follows the variable name. Please note that these are the characters [], not the parentheses

This is an example of code. It is in 10-point Terminal font.

Function names, commands, file names, and the like are in 10-point modern.

Be sure to include the following information in any module explanations:

- any file dependencies
- definitions of all arguments
- module, function variable, etc. limitations
- a liberal number of examples for all functions, variables, etc.

---

---

## LISP USERS' TEMPLATE

---

---

Updated by: Melissa Biggs (Biggs.PA @ Xerox.COM)

This document provides a template and instructions for formatting the Lisp Users' module documentation. This template applies primarily to standalone workstation users. Using the Lisp Library module TEdit, and this document, you should be able to create a standard Lisp Users' module for the Lisp Users' manual. This document gives you the written specifications for formatting your document. The specifications are given in the order in which you would most likely use them to format a document, with the basic text and margins described first, then the various levels of headings, then special elements such as page numbers.

### RULES FOR CONTENT

Documentation should always include the name of the module, the name of the author (and Xerox, Arpanet, CSNET or other electronic mailing address, when available—otherwise US mail address), the names of all other Lisp Users' modules required, the names of all files which are part of the module (data files, other Lisp files, etc.), and enough detail to allow someone to effectively use it. A sample Lisp Users' template appears at the end of this document.

### BASIC SPECIFICATIONS

It is wise to apply the specifications for the body text, headings, functions, variables, and page numbers as you write the document.

#### Font, Type Size, Leading, and Margins

For the text, choose a 10-point Modern font and Apply it to the appropriate text using the Character Looks menu.

#### Character Looks Menu

##### APPLY SHOW NEUTRAL

Props: **Bold** *Italic* Underline ~~StrikeThru~~ <sup>Oxerbar</sup>

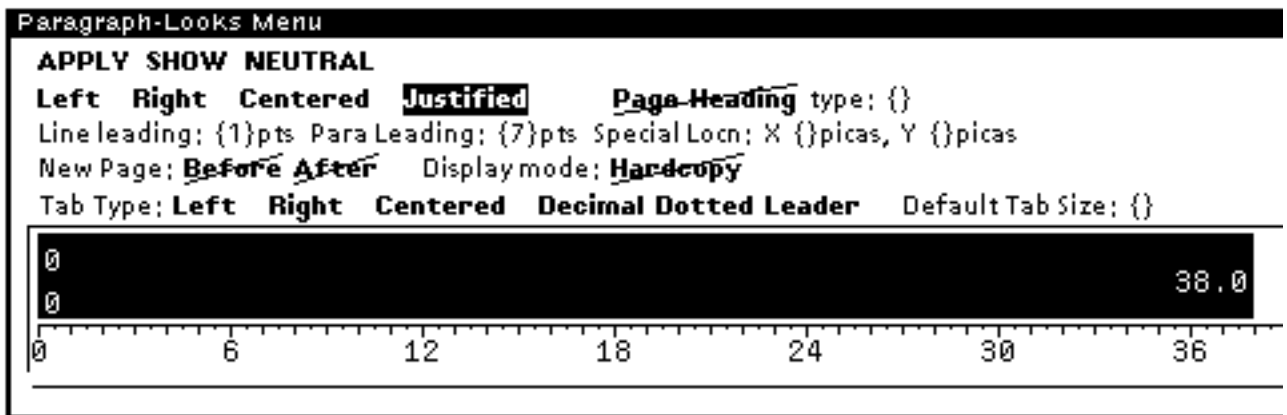
**TimesRoman** **Helvetica** **Gacha** **Modern** **Classic**

**Terminal** **Other**

otherfont: {}

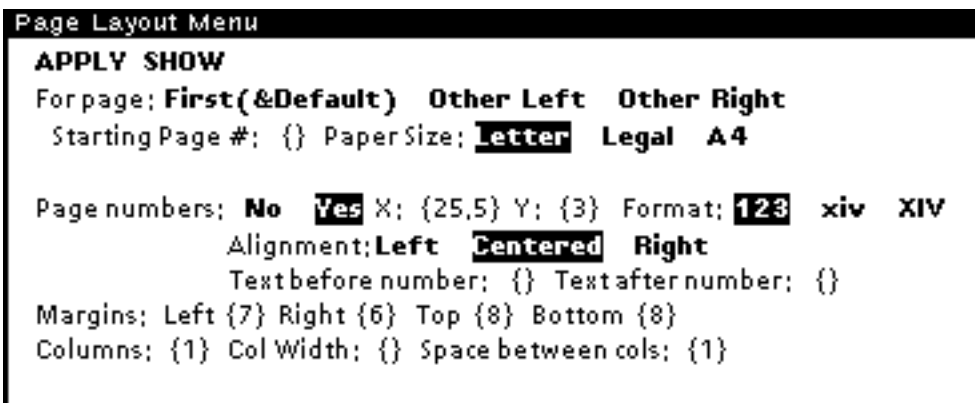
Size: {10} **Normal** **Superscript** **Subscript** distance: {}

Then, in the Paragraph Looks menu, set the leading, the spacing between lines of type; the justification; and the left and right margin settings. Set line leading to 1 point and paragraph leading to 7 points. Apply all paragraph looks to the appropriate text.



(Because 7 points paragraph leading is all you need, you should use only one carriage return between paragraphs when typing in text.)

Finally, in the Page Layout menu, set the left margin to 7 picas, the right margin to 6, and the top and bottom margins to 8. Apply these to all types of pages (first, other left, and other right).

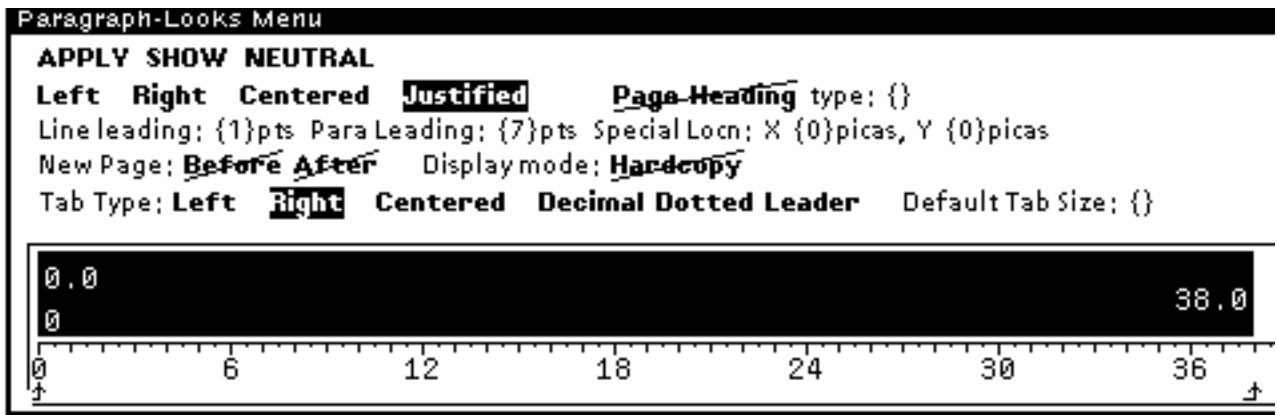


### Functions, Variables, and Lisp Code Examples

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN) [Function]

The Paragraph Looks menu for a function is set up like this:

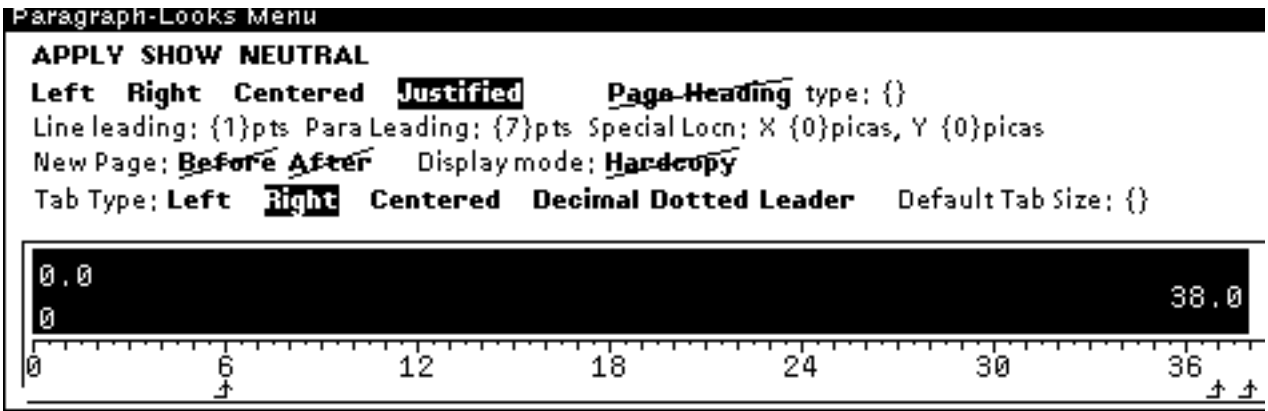


The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system.

If the function description is more than one line long, the runover arguments should be indented under the function name and the word [Function] placed on the last line of the argument list, like this:

```
(MAKE-ARRAY INDICESLIST &KEY :ELEMENT-TYPE: ^
 INITIAL-ELEMENT :INITIAL-CONTENTS :ADJUSTABLE: ^
 FILL-POINTER :DISPLACED-TO :DISPLACED-INDEX-OFFSET) [Function]
```

The Paragraph Looks menu should be set as follows to produce this type of argument format:



Long function descriptions should have two points leading between lines. To space them correctly, hold down the meta key when typing the carriage returns so that TEdit breaks the lines without creating separate paragraphs.

Parentheses are in 10-point regular Modern type. The type of definition is in 10-point Modern, caps and lowercase, enclosed in square brackets, and flushed right with a right tab set to 38.0. Note that tabs are used to indent the second and third lines of arguments.

Variables look like functions, except that the word "Variable," enclosed in square brackets, follows the variable name.

Examples of code should be in 10-point Terminal font (but not function names, commands, file names, and the like). If 10-point Terminal is not available on your printer, use 8-point Terminal. Code examples should have two points line leading and no paragraph leading.

**Quotation Marks, Bullets, and Dashes**

We recommend using TEdit's "expanded abbreviations" to produce professional-looking quotation marks, bullets, em-dashes—used to separate text phrases—and en-dashes (used to indicate inclusive numbers, as in "pages 3–6").

- To produce a bullet, type a lowercase b, select it, and type Control-X.
- To produce an em-dash, type a lowercase m, select it, and type Control-X.
- To produce an en-dash, type a lowercase n, select it, and type Control-X.

**HEADS**

There are four levels of heads in the Lisp Users' documentation: chapter (level 1) heads, level 2, level 3, and level 4 heads.

Note: A head that falls at the bottom of a page (a "widow") is undesirable. You eliminate a widow by selecting it, then applying the Before option of the New Page command in the Paragraph Looks menu.

**The Chapter Head**

The chapter head appears at the beginning of the document and identifies it. The heading "Lisp Users' Template," above, is a correctly formatted Lisp Users' chapter head.

When submitting a Lisp Users' module on floppy disk format the chapter head as follows:

Module Name: your Lisp Users' Module (all caps, 12-point bold Modern)

Your name and ARPANET address (if you have one) in 10-point Modern

**The Level 2 Head**

Level 2 heads identify major sections of a document. The level 2 heads for the Lisp Users' documentation are in 10-point bold Modern, all caps.

**The Level 3 Head**

Level 3 heads identify subsections of a document. For the Lisp Users' manual, they are in 10-point bold Modern, caps and lowercase.

**The Level 4 Head**

Level 4 heads identify the lowest level of subsection in the Lisp Users' documentation. They are in 10-point regular Modern, caps and lowercase, underlined.

**PAGE NUMBERS**

Page numbers are specified and applied in the Page Layout menu. First, specify the alignment of the page numbers to be centered, with the X position being 26.5 picas and the Y position 3.5. Then specify the character looks to be 10-point regular Modern. Finally, apply the page numbers to the First(&Default) pages.

| Page Layout Menu                                                                                                     |                                                  |
|----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|
| <b>APPLY SHOW</b>                                                                                                    |                                                  |
| For page: <b>First(&amp;Default)</b> <b>Other Left</b> <b>Other Right</b>                                            |                                                  |
| Starting Page #: {1}                                                                                                 | Paper Size: <b>Letter</b> <b>Legal</b> <b>A4</b> |
| Page numbers: <b>No</b> <b>Yes</b> X: {26,5}   Y: {3,5}   Format: <b>123</b> <b>xiv</b> <b>XIV</b>                   |                                                  |
| Alignment: <b>Left</b> <b>Centered</b> <b>Right</b>                                                                  |                                                  |
| Text before number: {}   Text after number: {}                                                                       |                                                  |
| Margins: Left {7}   Right {6}   Top {8}   Bottom {8}                                                                 |                                                  |
| Columns: {1}   Col Width: {}   Space between cols: {1}                                                               |                                                  |
| Page Headings:                                                                                                       |                                                  |
| Heading Type: {}   X: {}   Y: {}   Heading Type: {}   X: {}   Y: {}                                                  |                                                  |
| Heading Type: {}   X: {}   Y: {}   Heading Type: {}   X: {}   Y: {}                                                  |                                                  |
| Heading Type: {}   X: {}   Y: {}   Heading Type: {}   X: {}   Y: {}                                                  |                                                  |
| Heading Type: {}   X: {}   Y: {}   Heading Type: {}   X: {}   Y: {}                                                  |                                                  |
| Character Looks for Page Numbers: Props: <b>Bold</b> <b>Italic</b> <b>Underline</b> <b>StrikeThru</b> <b>Oxerbar</b> |                                                  |
| <b>TimesRoman</b> <b>Helvetica</b> <b>Gacha</b> <b>Modern</b> <b>Classic</b>                                         |                                                  |
| <b>Terminal</b> <b>Other</b>                                                                                         | otherfont: {}                                    |
| Size: {10} <b>Normal</b> <b>Superscript</b> <b>Subscript</b>                                                         | distance: {}                                     |

After you submit your document, XEROX AIS will add running heads, put in additional formatting, and provide final page numbering to assemble it into the Lisp Users' manual.

### MANUAL TEMPLATE

The following page has a sample template for the information required in a Lisp Users' module. Use this template to produce your module documentation.



Module Name:

**>>MODULE NAME<<**

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

>>Type INTERNAL here if the file is for Internal Use Only<<

This document last edited on >>DATE<<.

## **INTRODUCTION**

>>This paragraph should be replaced by an overview of your module. The information on the previous pages explains the documentation conventions to be used for each Lisp Users' module.<<

## **MODULE EXPLANATIONS**

### **>>Functions, Variables, and Lisp Code Examples<<**

It is usual to first give the name of a function, then describe its purpose and each of its arguments.

Module explanations may have several level headings.

Be sure to include the following information in any module explanations:

- any file dependencies
- definitions of all arguments
- module, function variable, etc. limitations
- a liberal number of examples for all functions, variables, etc.

---



---

## LISP USERS' TEMPLATE

---



---

Updated by: Melissa Biggs (Biggs.PA @ Xerox.COM)

This document provides a template and instructions for formatting the Lisp Users' module documentation. This template applies primarily to standalone workstation users. Using the Lisp Library module TEdit, and this document, you should be able to create a standard Lisp Users' module for the Lisp Users' manual. This document gives you the written specifications for formatting your document. The specifications are given in the order in which you would most likely use them to format a document, with the basic text and margins described first, then the various levels of headings, then special elements such as page numbers.

### RULES FOR CONTENT

Documentation should always include the name of the module, the name of the author (and Xerox, Arpanet, CSNET or other electronic mailing address, when available—otherwise US mail address), the names of all other Lisp Users' modules required, the names of all files which are part of the module (data files, other Lisp files, etc.), and enough detail to allow someone to effectively use it. A sample Lisp Users' template appears at the end of this document.

### BASIC SPECIFICATIONS

It is wise to apply the specifications for the body text, headings, functions, variables, and page numbers as you write the document.

#### Font, Type Size, Leading, and Margins

For the text, choose a 10-point Modern font and Apply it to the appropriate text using the Character Looks menu.

#### Character Looks Menu

##### APPLY SHOW NEUTRAL

Props: **Bold** *Italic* Underline ~~StrikeThru~~ <sup>Oxerbar</sup>

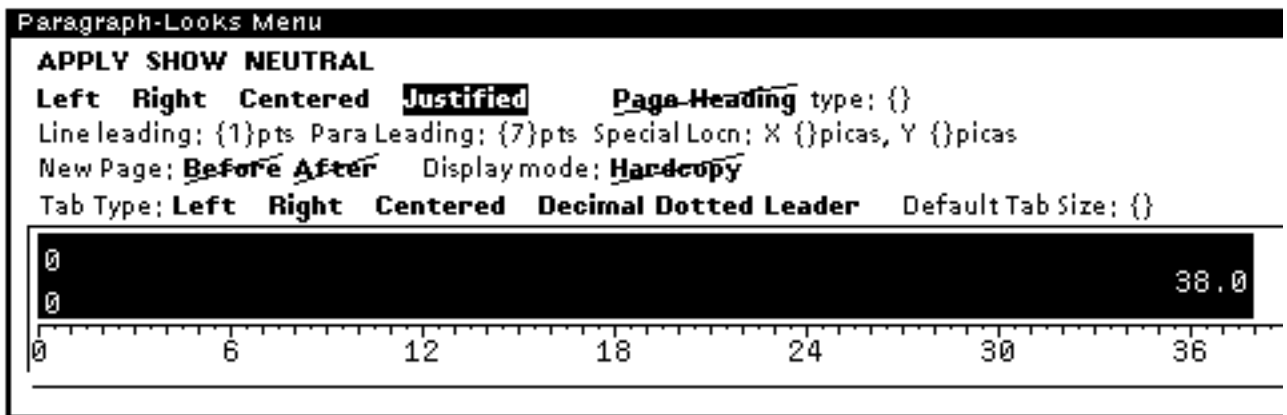
**TimesRoman** **Helvetica** **Gacha** **Modern** **Classic**

**Terminal** **Other**

otherfont: {}

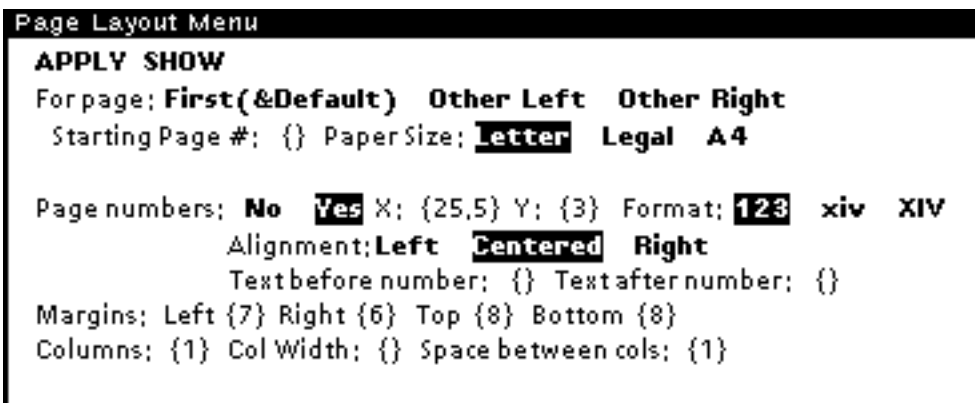
Size: {10} **Normal** **Superscript** **Subscript** distance: {}

Then, in the Paragraph Looks menu, set the leading, the spacing between lines of type; the justification; and the left and right margin settings. Set line leading to 1 point and paragraph leading to 7 points. Apply all paragraph looks to the appropriate text.



(Because 7 points paragraph leading is all you need, you should use only one carriage return between paragraphs when typing in text.)

Finally, in the Page Layout menu, set the left margin to 7 picas, the right margin to 6, and the top and bottom margins to 8. Apply these to all types of pages (first, other left, and other right).

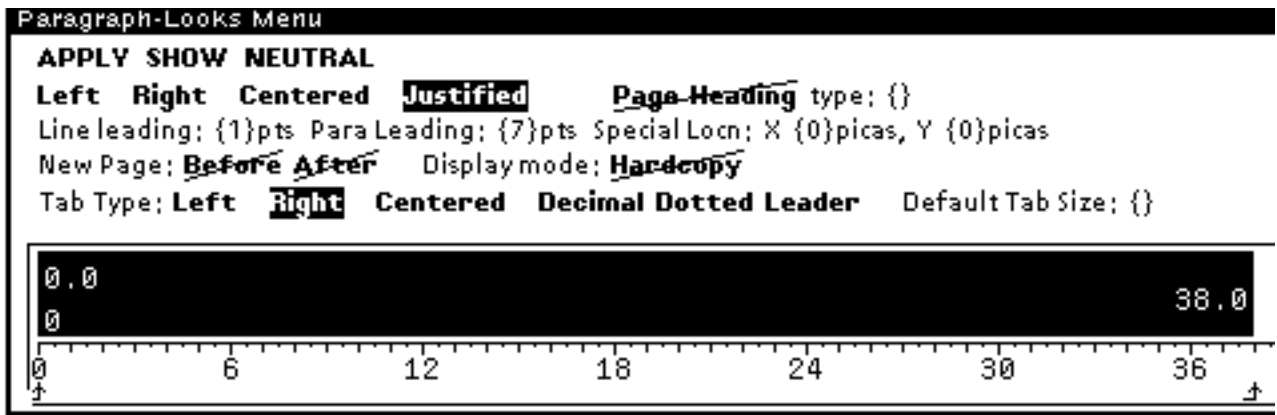


### Functions, Variables, and Lisp Code Examples

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN) [Function]

The Paragraph Looks menu for a function is set up like this:

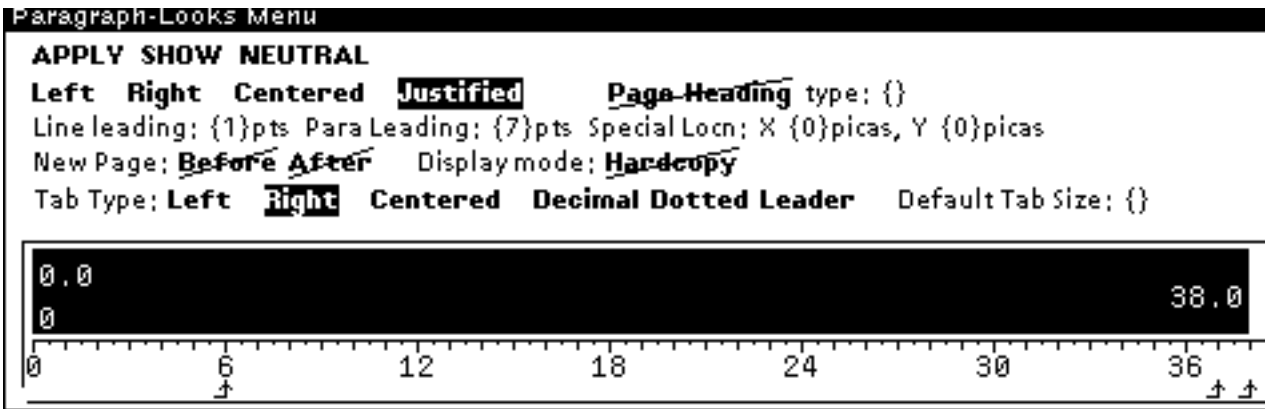


The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system.

If the function description is more than one line long, the runover arguments should be indented under the function name and the word [Function] placed on the last line of the argument list, like this:

```
(MAKE-ARRAY INDICESLIST &KEY :ELEMENT-TYPE: ^
 INITIAL-ELEMENT :INITIAL-CONTENTS :ADJUSTABLE: ^
 FILL-POINTER :DISPLACED-TO :DISPLACED-INDEX-OFFSET) [Function]
```

The Paragraph Looks menu should be set as follows to produce this type of argument format:



Long function descriptions should have two points leading between lines. To space them correctly, hold down the meta key when typing the carriage returns so that TEdit breaks the lines without creating separate paragraphs.

Parentheses are in 10-point regular Modern type. The type of definition is in 10-point Modern, caps and lowercase, enclosed in square brackets, and flushed right with a right tab set to 38.0. Note that tabs are used to indent the second and third lines of arguments.

Variables look like functions, except that the word "Variable," enclosed in square brackets, follows the variable name.

Examples of code should be in 10-point Terminal font (but not function names, commands, file names, and the like). If 10-point Terminal is not available on your printer, use 8-point Terminal. Code examples should have two points line leading and no paragraph leading.

## Quotation Marks, Bullets, and Dashes

We recommend using TEdit's "expanded abbreviations" to produce professional-looking quotation marks, bullets, em-dashes—used to separate text phrases—and en-dashes (used to indicate inclusive numbers, as in "pages 3–6").

- To produce a bullet, type a lowercase b, select it, and type Control-X.
- To produce an em-dash, type a lowercase m, select it, and type Control-X.
- To produce an en-dash, type a lowercase n, select it, and type Control-X.

## HEADS

There are four levels of heads in the Lisp Users' documentation: chapter (level 1) heads, level 2, level 3, and level 4 heads.

Note: A head that falls at the bottom of a page (a "widow") is undesirable. You eliminate a widow by selecting it, then applying the Before option of the New Page command in the Paragraph Looks menu.

### The Chapter Head

The chapter head appears at the beginning of the document and identifies it. The heading "Lisp Users' Template," above, is a correctly formatted Lisp Users' chapter head.

When submitting a Lisp Users' module on floppy disk format the chapter head as follows:

Module Name: your Lisp Users' Module (all caps, 12-point bold Modern)

Your name and ARPANET address (if you have one) in 10-point Modern

### The Level 2 Head

Level 2 heads identify major sections of a document. The level 2 heads for the Lisp Users' documentation are in 10-point bold Modern, all caps.

### The Level 3 Head

Level 3 heads identify subsections of a document. For the Lisp Users' manual, they are in 10-point bold Modern, caps and lowercase.

### The Level 4 Head

Level 4 heads identify the lowest level of subsection in the Lisp Users' documentation. They are in 10-point regular Modern, caps and lowercase, underlined.

## PAGE NUMBERS

Page numbers are specified and applied in the Page Layout menu. First, specify the alignment of the page numbers to be centered, with the X position being 26.5 picas and the Y position 3.5. Then specify the character looks to be 10-point regular Modern. Finally, apply the page numbers to the First(&Default) pages.

| Page Layout Menu                                                                         |                                    |
|------------------------------------------------------------------------------------------|------------------------------------|
| <b>APPLY SHOW</b>                                                                        |                                    |
| For page: <b>First(&amp;Default)</b>                                                     | <b>Other Left Other Right</b>      |
| Starting Page #: {1}                                                                     | Paper Size: <b>Letter</b> Legal A4 |
| Page numbers: <b>No Yes</b> X: {26,5} Y: {3,5} Format: <b>123</b> xiv XIV                |                                    |
| Alignment: <b>Left Centered Right</b>                                                    |                                    |
| Text before number: {} Text after number: {}                                             |                                    |
| Margins: Left {7} Right {6} Top {8} Bottom {8}                                           |                                    |
| Columns: {1} Col Width: {} Space between cols: {1}                                       |                                    |
| Page Headings:                                                                           |                                    |
| Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}                                |                                    |
| Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}                                |                                    |
| Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}                                |                                    |
| Heading Type: {} X: {} Y: {} Heading Type: {} X: {} Y: {}                                |                                    |
| Character Looks for Page Numbers: Props: <b>Bold Italic Underline StrikeThru Overbar</b> |                                    |
| <b>TimesRoman Helvetica Gacha Modern Classic</b>                                         |                                    |
| <b>Terminal Other</b>                                                                    | otherfont: {}                      |
| Size: {10} <b>Normal</b> Superscript Subscript distance: {}                              |                                    |

After you submit your document, XEROX AIS will add running heads, put in additional formatting, and provide final page numbering to assemble it into the Lisp Users' manual.

### MANUAL TEMPLATE

The following page has a sample template for the information required in a Lisp Users' module. Use this template to produce your module documentation.

Module Name:

**>>Module Name<<**

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

>>Type INTERNAL here if the file is for Internal Use Only<<

This document last edited on >>DATE<<.

## **INTRODUCTION**

>>This paragraph should be replaced by an overview of your module. The information on the previous pages explains the documentation conventions to be used for each Lisp Users' module.<<

## **MODULE EXPLANATIONS**

### **>>Functions, Variables, and Lisp Code Examples<<**

It is usual to first give the name of a function, then describe its purpose and each of its arguments.

Module explanations may have several level headings.

Be sure to include the following information in any module explanations:

- any file dependencies
- definitions of all arguments
- module, function variable, etc. limitations
- a liberal number of examples for all functions, variables, etc.

---

---

## ACE

---

---

By: Michel Denber (Denber.wbst@Xerox.com) Compiled for Medley by Larry Masinter  
(Masinter.PA@Xerox.COM)

**Files: ACE.LCOM**

**Data files: ACE-APPLEDEMO.ACE, ACE-BOUNCINGBALL.ACE, ACE-FOUETTE.ACE**

## Animation Compiler and Environment

### Introduction

ACE is a system for computer-assisted animation. It is based on the traditional cel-oriented animation process with the computer taking over many of the tedious jobs. You enter a succession of frames which represent a *sequence*. The system then plays back your frames to create the animated effect. It lets you draw pictures, enter text, and edit your work. The animated images you make are displayed on the screen in real-time. The two main parts of ACE system are a frame compiler and an environment. The environment provides the editing tools, frame manipulation, and display capabilities. The compiler operates automatically to produce a compressed-storage representation for frames.

You can also use the graphic editing features in ACE to make individual pictures, whether or not they're intended to be used for animation. Finally, you can use the compiler directly to compress any bitmap image so that it take up less space on your disk.

The majority of the code for ACE was originally written by Paul Turner, a student at the University of Rochester. I am currently maintaining the system. Please send all bug reports, comments, and suggestions directly to me, Denber.WBST, or Denber.WBST@Xerox.COM (Arpanet). This document describes the features available in ACE version 2.1.

### Background

In this document: *holding* the mouse on a menu selection means to press down a mouse button on a menu item (inverting the item) and keeping it down for about 1.5 seconds (at this point, you can release the button or move to another selection). *Clicking* the mouse means pressing a mouse button down and releasing it. Unless otherwise stated, the left mouse button is used for selecting items from menus and to click at objects.

In addition to the mouse, ACE supports a graphics tablet (*Summagraphics MM1201*) . [LMM: The graphics tablet hasn't been tested in Medley.] The tablet is more convenient for doing free-hand drawing; in fact, most commercial animation systems include a graphics tablet. The pen has two buttons: the stylus *tip* (which is activated by pressing down on it) and a blue button on the *barrel* of the pen (activated by pressing with the forefinger). We have adopted a convention with regard to the tablet: the stylus button acts like the left button on the mouse and the barrel button acts like the middle button on the mouse.



## Terms and System Organization

A *region* is simply a rectangular area.

A *frame* is a region that contains one complete "picture" in an animation sequence; it is a rectangular bitmap with a fixed width and height.

A *sequence* is a collection of frames defining one complete animated segment.

The *current frame* is the frame in a sequence to which operations will be applied. As all the frames in any given sequence are the same size, you may say that one characteristic of a sequence is a region of a particular size. Frames are referred to by number for convenience; the numbering is from 1 to n.

There are two principal windows used in ACE. The *sequence window* is the window on the screen where a particular sequence will be created, edited and displayed. Typically, you define the shape of the sequence window to give you just the area you want to work in, although a sequence can also be edited and displayed in any existing window.

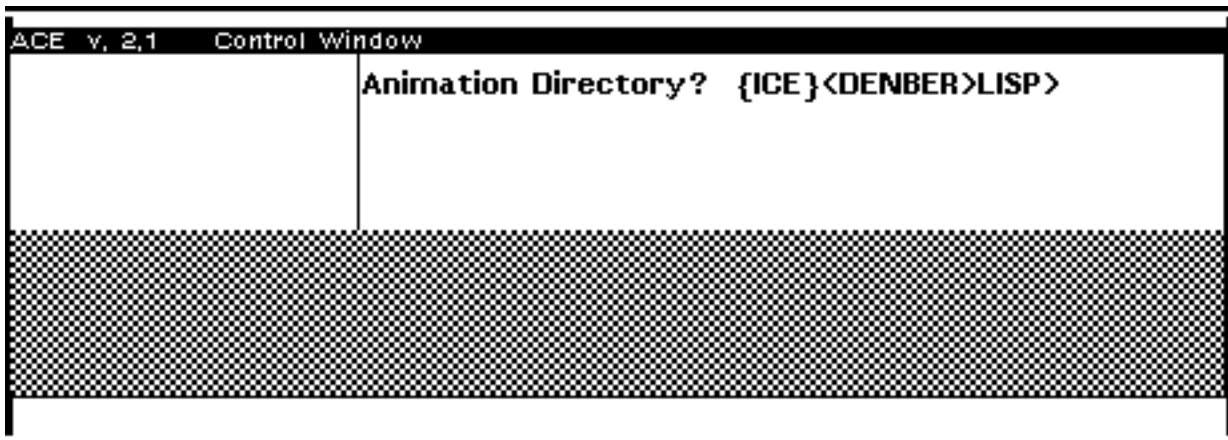
The *ACE Control Window* holds a menu of commands and displays animation state, prompt, and help information. The upper left region in the control window, referred to as the *status region*, tells which frame is currently being displayed (which frame is the current frame); which device (mouse or tablet) is being used in line art and painting operations; what operation is currently being performed; and, the size of a region (width, height) or the location (x, y) of the cursor within the sequence. The upper right portion of the control window is the *prompt region*; it is used to get user input and display helpful information. The bottom part of the control window is a menu of animation functions.

## GETTING STARTED

Load ACE.LCOM from your lispusers directory (e.g., (FILESLOAD (SYSLOAD) ACE). When this is complete, type:

(ACE)

At this point an Ace control window will appear by the cursor. You can place it wherever you wish on the screen. The window initially contains a prompt "Animation Directory?" asking for a default directory to use for storing and retrieving animation files (The default selection is your login directory. Just press the return key to accept the default.) The control window can be moved around just like any other window. While you never need to "quit" from ACE, if you close the control window with the right mouse button menu it permanently aborts ACE; if you then re-type (ACE), the animation system will be restarted from scratch.



*ACE Control Window at start-up*

## ACE Commands

### Main Menu

The menu selections from the control window will now be described; an example of using ACE is given in the next section. The main menu is divided into three columns. The left-most one contains commands that affect the entire sequence, the middle column operates on frames, and the right-most column contains utility commands.

|                      |                      |                          |
|----------------------|----------------------|--------------------------|
| Get Sequence         | Edit Frame           | Run Sequence             |
| Put Sequence         | New Frame            | Increment Frame          |
| New Sequence         | Delete Frame         | Decrement Frame          |
| Reset Sequence       | Adjust Timing Delays | Initialize MM1201 Tablet |
| Change compression % | Change Input Device  | Quit                     |

*Sequence commands (left column)*

**Get Sequence** Loads a sequence-file from a file server or local disk. You are prompted for the name of the file; incomplete file names will be completed from the directory given as the default ACE directory. You will then be prompted for a window specification. Unless you want to display the sequence in a particular window, select 'create window automatically'.

**Put Sequence** Saves the current sequence to a file. You will be asked for a file name and given the option to overwrite the existing version of the file (if any) or create a new version.

**New Sequence** Discards the current sequence (if any), and prompts you for a new sequence by requesting a size for the new sequence. Dragging out a rectangular region with the mouse; the exact size of the region is displayed in the status region of the control window. A blank first frame is created, ready for editing.

**Reset Sequence** "Rewinds" the current sequence to the beginning (i.e. there is no current frame and the next frame to be displayed will be the first frame).

**Change compression %** This can be used to change the amount of space compression performed by the animation compiler. For general use, there is no need to ever call this command.

### *Frame commands (middle column)*

**Edit Frame** Allows editing on the current frame. Brings up a menu of editing options, described in the next section.

**New Frame** Inserts a new frame after the current frame (ie. before the next frame). The frame editor is then automatically invoked.

**Delete Frame** Deletes the current frame. The current frame is removed and the previous frame become the current frame. The first frame can not be deleted.

**Adjust Timing Delays** Lets you set the amount of time (in milliseconds) that any particular frame is displayed; for example, a delay of 50 on the 5th frame would mean that the 5th frame will be visible for 50 milliseconds before the 6th frame is put up. You can change the entire sequence or a single frame at a time. For individual frame setting, you get a menu of frames and their current delays. Holding down a selection will display that particular frame in the sequence window (this is also a convenient way to rapidly move to an arbitrary frame); if you select a frame you will be prompted for a new delay value. When new frames are created, they always get a default delay time of 0.

**Change Input Device** Lets you select either 'mouse' or 'tablet' from a menu. This sets the device to be used for line art and painting operations. The status (upper left) region of the control window always shows which device is active. All menu selections have to be done with the mouse, even when the tablet is being used for drawing

*Utility commands (right column)*

**Run Sequence** Runs the *remainder* of the sequence. To run the entire sequence, select Reset Sequence before Run. This command has a submenu with two additional commands:

**Loop** Runs the entire sequence in a continuous loop. To stop the loop, hold down the space bar. This is checked only at the end of the sequence, so just tapping the space bar may not stop the loop.

**Loop part** Runs a portion of the sequence in a continuous loop. You can specify the starting and ending frame numbers. To stop the loop, hold down the space bar, as in Loop above.

**Increment Frame** Displays the next frame and makes it the current frame.

**Decrement Frame** Goes back to the preceeding frame and makes it the current frame.

**Initialize MM1201 Tablet** This performs the necessary RS232 port initializing, sets the baud rate, activates the tablet, etc. This must be called after the tablet is plugged in and before it is used. If your tablet doesn't seem to be responding, it may need to be reinitialized.

### *Edit Frame Submenu*

For the commands described below, it is sometimes useful to know the exact coordinates at which a drawing operation will take place. If you hold the T key down, ACE will put the current coordinates in the status window. Release the key to stop this function.

When you select **Edit Frame**, a sub-menu of editing options appears. Their functions are as follows:

**Paint** This lets you to paint on and erase bits from the current frame. The painting operation is the standard Interlisp-D window paint command. Either the mouse or tablet can be used (which ever device is currently selected). Pressing the left mouse button or pen stylus draws; the middle mouse or pen barrel button erases. Pressing the left shift key brings up a menu. From this menu, you can quit painting, change the brush size or shape, and the color or texture of the "paint brush". Note: selecting items from this menu requires using the mouse (unfortunately, the tablet cannot be used for menu selecting). For more information on the paint command, please see page 19.20 in the Interlisp manual.

**Line Art** This lets you add straight lines to a frame by selecting one vertex, dragging out a line, and then selecting another vertex. In this way, an arbitrary string of connected line segments can be created. The left mouse button or pen stylus will "put down" vertices, the middle mouse button or pen barrel stops the line dragging. The right mouse button brings up a menu of line art options (paint or invert drawing; several line width choices); as with all menus, the menu selection must be made with the mouse.

**Edit Bits** This lets you use the Interlisp-D bitmap editor on the selected frame. The mouse is used to turn specific bits on or off (the tablet is not used as it isn't helpful for this application). A complete description of the the bitmap editor is given in the Interlisp Reference Manual. Always exit the editor by selecting **OK**.

**Text** Lets you put text into a frame. After selecting the 'Text' option, you will be prompted for font characteristics. Then you point (with the mouse) to where the text should begin and click the left mouse button. You may now type in text from the keyboard and it will show up in the frame. A press of the return key ends text entering (the return is NOT included in the text).

**Move Region** Lets you move an arbitrary rectangular region in the current frame. You first drag out the region to be moved, then you will be asked what to do with the old image (i.e. leave it alone, erase it). At this point, a ghost image will attached to the cursor and you can position the image in its new location. Clicking the mouse will set the position for the image and then you will be asked how to combine the image (i.e. paint it in, exclusive-or it in).

**Combine Region** This is similar to move region, except that you can select any region on the screen (not just inside the current frame). After selecting the region, bringing the mouse within the sequence window will show a ghost image; the rest of the procedure is the same as for **Move Region**. This command is extremely useful for bringing images created elsewhere into your frame. For example, you might have a drawing made using Sketch or an AIS file.

**Texture Area Fill** This lets you fill in an arbitrary closed curve with a texture pattern. You are first prompted to select a bounding region. This reepresents a maximum area beyond which texturing will not occur, in the event that the texture "spills" outside the region being shaded. Next, select a starting point anywhere inside the desired region. You will then be offered a menu of predefined textures. You can choose one of these, or create your own by selecting **\* Other \***. The area is then filled with that texture. You can then confirm that the right thing happened by clicking left. Click any other button to undo the operation.

**Texture Region Fill** This is like Texture Area Fill, except that it is used to create filled-in rectangular boxes, rather than arbitrary areas.

**Scale Region** Lets you change the size of any rectangular area. You first select a region, and then indicate the shape of the scaled area by sweeping it out on the frame. Useful if you know how big you want the result to look but not what percent of the original it is. This command has a submenu:

**To a new region** Same as the top level Scale Region.

**In x and y** You first select a region as in **To a new region**, and then indicate the percentage of the original size to scale by, much like selecting reduction or enlargement on a copier. You can set the x and y scale factors independently.

**In x only** Use this if you only want to scale in the x direction, leaving y unchanged.

**In y only** Use this if you only want to scale in the y direction, leaving x unchanged.

**Clear Region** For clearing regions to white quickly. If you select a region within the sequence, it is immediately erased. There is no UNDO for this operation.

**Quit - Compile** This is the usual way of exiting the editor. This keeps the changes made to the frame and calls the compiler, resulting in adding the frame to the current sequence.

**Quit - ABORT** Exits the editor but does not update the frame. The sequence will be as it was before you selected **Edit**. Any changes made to this frame are lost.

## CONCLUSION

### Examples

You might want to see an existing animation before creating one of your own. There are several animation demos included in the Lispusers distribution of ACE: ACE-APPLEDEMO, ACE-BOUNCINGBALL, AND ACE-FOUETTE. The simplest one, BOUNCINGBALL, contains five frames showing a bouncing ball. To see this, start ACE running and select Get Sequence from the main menu. Type the name of the file, including the file server and directory if they are different from your current animation directory. Once the file is loaded ACE will let you position the sequence window. Then select Run Sequence. The system will display the five frames and then stop. To see the ball bounce continuously, select LOOP from the submenu on the Run Sequence command. The ball will now bounce until you hold the space bar down.

The file ACE-APPLEDEMO is a 125 frame sequence which shows an apple getting shot. ACE-FOUETTE is a six frame cycle of a ballet dancer performing a fouette turn.

### Animation Hints

Remember that ACE is just a tool - it will not do any animating for you. Our goals were to provide a system that simplified the frame creation process and let you create animation on the computer without having to learn a special animation programming language. However, animation is an art form in itself. Experience is gained only through practice and experimentation.

Avoid moving objects too far between frames or the motion will appear jerky. The D machines screens are designed to minimize flicker through the use of a long persistence phosphor. Unfortunately, this results in trailing streaks of light behind rapidly moving objects. Sometimes you can use this to artistic effect. It can be reduced by moving dark objects over a light background, rather than the reverse.

Sometimes you can simplify the animation process by creating frames out of order, especially for cyclic animation. For example, the bouncing ball was created by drawing frame 1 and then making a new frame (which is by default a copy of frame 1). Then we backed up to frame 1 and added a new frame between 1 and 2, showing the ball half-way down. Then this frame was copied, yielding four frames. Backing up again and adding the middle frame gave a symmetrical bounce sequence with frames 1, 5 and 2, 4 being identical.

It's often convenient to keep a snapshot of the object you're animating handy in a window next to the sequence window. It can then be brought into the frame whenever needed, for example in the case where it is being modified in some way. You are not limited to editing images with the ACE editor. In particular, you may want to use the *Sketch* editor (an Interlisp-D library package) to modify images, and then bring them into the sequence window for compilation.

### References

Denber, Michel, Paul Turner, "A differential compiler for computer animation", to appear in *Computer Graphics*, 20:3, 1986 (Proc. SIGGRAPH '86)

Fox, David, Mitchell Waite, *Computer Animation Primer*, McGraw-Hill, N.Y. 1984

Magnenat-Thalmann, Nadia, Daniel Thalmann, *Computer Animation: Theory and Practice*, Springer-Verlag, Tokyo, 1985

**ACE Maintainer's Notes**  
**Created: 4/29/85**  
**Revisions:**

## **INTRODUCTION**

This document is intended for programmers who intend to fix, update or otherwise significantly modify the ACE code. A general description and several specific points about the Animation Compiler and Environment will be brought out. This is neither a user's guide nor a guide for animation. A user's guide may be found filed under {ICE}<TURNER>ACE>ACE-USERS-GUIDE; a technical document is filed under {ICE}<WHOKNOWS>ACE>WHOKNOWS.

## **OVERVIEW**

The Animation Compiler and Environment is a program for creating, editing, displaying and storing/retrieving frame-oriented animations. It is based on traditional animation principals; successive frames (each frame usually slightly different from the previous frame) are displayed at a rate fast enough to create the illusion of motion (atleast 10 frames/second; usually more). To the user, the ACE system presents a sequence of whole frames (represented as bitmaps with the same width and height); the analogy being a stack of paper. In reality, however, the program maintains virtual frames. This decreases the storage requirements and allows the very rapid display of frames.

## **VIRTUAL REPRESENTATION**

The virtual frames maintained by ACE are differential frames; that is, a frame only contains the information that is different from the previous frame. By this mechanism, the first frame is a complete frame (complete bitmap) providing the initial information. The second frame, then, is just the differences that exist between the first frame and a hypothetical complete second frame (i.e. just the differences between two bitmaps). And so on with all successive frames. The details of data structures and the compilation process are given below.

## **TRILLIUM vs STAND-ALONE**

Currently, ACE is designed to operate as either a stand-alone program or in the Trillium environment. Conceptually, as a stand-alone, ACE should be considered more procedure oriented. When ACE is called (top level fn: (ACE)), a control window (ACE.CONTROL.WINDOW) is brought up which contains a menu. This window and menu remain up and stay active; once ACE is activated, it becomes a part of the user's current environment. In Trillium, ACE operates more functionally; that is, ACE is called as a one-time function which returns a value (although, ofcourse, it produces side-effects). The control window and menu are only active during the function call and are taken away when the ACE session is concluded. Also, a special animation-run-time function is provided for Trillium to use outside of ACE to run animation item types. At present, Trillium expects all editing (and most loading and storing) of animation sequences to be performed inside ACE.

## **ORGANIZATION**

ACE is organized in a top-down fashion which allows a general to specific description of it. What the components do and how (where appropriate) will now be outlined.

**FILES:** There are four files to the ACE system (and one utility: RS232): ACE, ACE-MAIN, ACE-EDIT and ACE-PRIM. The divisions exist partly for organizational purposes and partly for ease in locating and working with segments. The ACE file contains some macros and variables, but exists primarily to load in all the necessary files. ACE-MAIN contains all the functions necessary to define, manipulate, run and load/store animations. ACE-EDIT contains all the editing functions (including interfacing to the MM1201 graphics tablet). ACE-PRIM contains the compiler functions; it is only concerned with compiling two frames to create a virtual frame.

## **ACE-MAIN**

This is the guts of the animation environment. There are four main divisions in MAIN (you can see them by looking at the file coms): Top level fns, Trillium-gearred fns, I/O fns, and "helper" fns. In addition, there are several macros and a GLOBALVARS declaration. All the main functions that work with animation sequences are in the first division (including the startup fn: ACE). The I/O section is just for reading and writing files; user input and output fns are in the helper fns section. Control window operation and clipping region fns are also located in the helper fns section.

## **ACE-EDIT**

This segment also consists of four parts (again, note the file coms). The first section provides

entry/interfacing to MAIN and calls to the actual editing routines. LINEART contains all the major fns for line art drawing. The third part contains all the other editing fns (e.g. painting, text, moving, etc.). The last part consists of tablet access fns and helper fns (all MM1201 code is located in this section and the code to read the current input device). The RS232 package is used by ACE-EDIT to read the MM1201 Summagraphics tablet. At present, the RS232 package (RS232.DCOM) is loaded by the ACE file. However, the maintainer should keep aware of changes in this package and its whereabouts.

#### ACE-PRIM; How the Compiler Operates:

The compilation process is concerned only with compiling two frames at one time (MAIN and EDIT take care of administration). The entry function is ACE.COMPILE.FRAME with arguments: BM.ORIG (the first or original bitmap), BM.CHANGED (the successor bitmap), VERTICAL.BLOCK (defines the height in scanlines for primitive regions of change; the horizontal component is 16 so as to take advantage of the 11xx series word size), and THRESHOLD (the percent as an integer representing the minimum amount of "changed area" allowable in a combination; more on this later). Take note that there are two variables defined in PRIM: ACE.PIXPERWORD (16, just the 11xx word size) and ACE.BITMAP.MASK (a mask used for ignoring extra bits in the last raster word of a bitmap).

The compiler works by comparing each word in BM.ORIG to BM.CHANGED (NOTE: these bitmaps must have the same dimensions) VERTICAL.BLOCK words at a time. If a changed word is found, a region specification is entered on list denoting this "region of change". After the whole area of the bitmaps is checked, the compiler attempts to merge these smallish changed-regions (see ACE.MAX.REGIONS). The algorithm first attempts to combine regions which result in no space wastage (this is always desirable). When no more 100% regions can be formed, the algorithm tests all pairings of primitive regions to find the highest efficiency (i.e. least space waste). If this efficiency is greater than or equal to THRESHOLD, the two regions are combined, otherwise, the algorithm terminates. The region merging process is the slowest aspect of ACE; the problem of merging regions is thought to be NP-complete.

#### DATA STRUCTURES

- A sequence is of the form: (FRAME FRAME FRAME ...).
- A FRAME is of the form: (DELAY BLITS).
- A DELAY is an integer representing a time delay in milliseconds.
- BLITS is a list: (BLIT BLIT BLIT ...).
- A BLIT is of the form: (BITMAP XCOORD . YCOORD).

A BLIT is essentially a small changed area with information on where it should be placed (relative to the animation). The record definitions for the above structures are in the file ACE. The compiler uses the structures: REGION : (LEFT BOTTOM WIDTH HEIGHT); a modified REGION : (REGION . AREA) for merging; and lists or both of these types.

#### GLOBALVARS

The following are important global variables with explanations where needed.

- ACE.CONTROL.WINDOW contains the top-level menu and status information.
- ACE.DIRECTORY is a default directory used to store/retrieve files.
- ACE.SEQ.WINDOW the current window where animations are displayed.
- ACE.SEQ.WIDTH and ACE.SEQ.HEIGHT refer to the current sequence.
- ACE.SEQ.WINDOW.XOFF and ". ".YOFF the offset in the ACE.SEQ.WINDOW.
- ACE.CURRENT.SEQUENCE points to data which is the current animation sequence.
- ACE.CURRENT.SEQUENCE.NAME for retaining file name information.
- ACE.FRAME.TAIL tail of frames starting one after the current frame.
- ACE.CURRENT.FRAME a tail of frames starting with the current frame.
- ACE.VERTICAL.BLOCK value to use when compiling (see above on compiler).
- ACE.AREA.THRESHOLD for compiling.
- ACE.RUNNING.UNDER.TRILLIUM T if ACE was called by Trillium.
- Various .CURSORS are just cursors.

There are also a GLOBALVARS list of menus in MAIN and EDIT. The approach on menus was that they should be created only once to save both time and space.



---

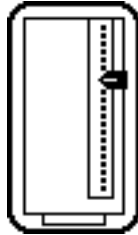
---

**ADDRESSBOOK**

---

---

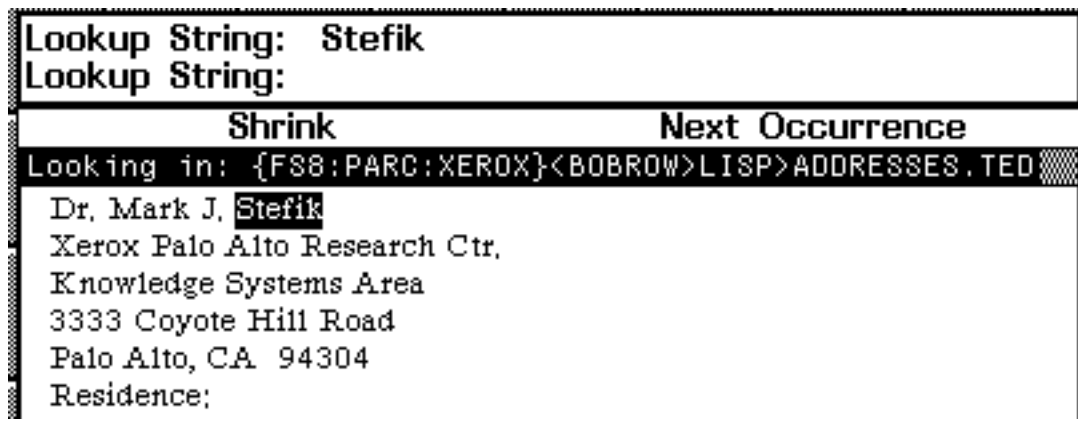
By: dgb (Bobrow.pa@Xerox.com)

**INTRODUCTION**

The ADDRESSBOOK package provides quick and easy access to on-line address books or phone directories. It allows you to copy (shift select) from entries found in the book, for example, for use as a letter or electronic mail address. When you load the ADDRESSBOOK package, the icon shown above will appear on your screen. Opening this icon will provide a window interface to a simple search process. To find an entry containing any string in one of your \*AddressBookFiles\*, type the string followed by a return. The ADDRESSBOOK program will quickly search through the files and show you an occurrence of the string typed. The located string is shown in inverse video. The title of the window will contain the name of the file in which the entry was found. You can use a name, part of the address or any keywords to locate the appropriate part of the text. The search ignores case; e.g. "bobrow" matches "Bobrow". The text of the document is scrollable, and any portion can be shift selected into another document.

Type carriage return, ^X, or click on **Next Occurrence** to search further in the files for the same string. If no (further) occurrences are found, the text window will display a message indicating the failure. Searching again after failure will start the search from the beginning of all the files, using the same lookup string. Typing a new string can be repeated as many times as you like. When you are done, just SHRINK the window back to its icon by using the Shrink selection in the title bar .

---



Example ADDRESSBOOK window

---

## Required Files

This package automatically loads LOOKUPINFILES.

## Variables

**\*AddressBookFiles\*** [Variable]

**\*AddressBookFiles\*** is a list of files that contain entries to be searched. This is usually set in the INIT.LISP file. In the ADDRESSBOOK package it is initially set to PHONELISTFILES, to make it backwards compatible with PHONE-DIRECTORY. The **\*AddressBookFiles\*** can be any unformatted or TEDIT formatted files, with any number of lines per entry. A typical value for PARC users (as defined for PHONELISTFILES in PARC-INIT) is

```
({ PHYLUM } < REGISTRAR > PARCPHONELIST.TXT
 { INDIGO } < REGISTRAR > ISDPHONELIST.TXT).
```

**\*Address-Book-Pos\*** [Variable]

**\*Address-Book-Pos\*** is the initial POSITION for the ADDRESSBOOK icon. This is defined as an INITVAR in the file, so you can set it before loading the file. The default value is

```
(create POSITION XCOORD ← 970
 YCOORD ← (DIFFERENCE SCREENHEIGHT 90)).
```

This places the icon in the upper right corner of the screen.

\*Address-Book-Region\*

[Variable]

\*Address-Book-Region\* is the initial REGION for the ADDRESSBOOK window. This is defined as an INITVAR in the file, so you can set it before loading the file. The default value is

```
(CREATEREGION 300 (DIFFERENCE SCREENHEIGHT 500)
 400 200).
```

This places the window in the middle of the screen.

## Notes

### Starting or Restarting Address Book

Evaluating (MakeAddressBook) will create an address book window and process. This may be useful if you accidentally close the window.

### Caching Files

When you first open the ADDRESSBOOK window, the program will copy the \*AddressBookFiles\* to {CORE}, significantly speeding up queries. Bugging in the title of the ADDRESSBOOK window with the left or middle mouse button will produce a menu with an option to recache the files on \*AddressBookFiles\*.

### Editing Your Files

To edit the file in which an entry is found, click middle button in the title of the ADDRESSBOOK window, and select the option "Edit file named in window title". A TEDIT process editing the file will be set up. This process is independent of the lookup process. To select the file to be edited, rolloff the above item, and select "Select file to edit". A menu of files used by the Lookup process will be presented to you. Selecting one will cause that file to be edited.

To make editing changes visible to the lookup process, PUT the file in TEDIT; when it is done, recache the the file in core. To recache just the file edited, (the one specified in the title bar of the window), select the option "Recache file named in window title" in the middle button title bar menu. You can recache all files by selecting the option "Recache all files" in the title menu (a subselection of the item "Recache file named in window title").

### Adding to the List of Files

To add to the list of files being used for lookup, select the option "Add new file" in the title bar menu. This file will be added, and cached in core.

### **Deleting a file from the List of Files**

To delete from the list of files being used for lookup, select the option "Delete file from list" in the title bar menu. This file will be deleted from the list of files to be searched.

---

---

## AIREGIONS

### (Active Irregular Regions)

---

---

By: Greg Wexler (Wexler.pasa@Xerox)

and

By: Jim Wogulis (Wogulis@ICS.UCL.EDU)

New Owner: James Turner (Turner.Lexington@Xerox.com)

Uses: FILLREGION, AIREGIONS-DEMO

### INTRODUCTION

The purpose of this package is to provide menu-like operations on irregularly shaped regions within a window and make available general functions that allow users to create their own applications using irregularly shaped active regions. An added feature of AIREGIONS is that multiple IREGIONS may be activated by selecting the intersecting area of those IREGIONS. (Throughout this document an irregularly shaped region will be referred to as an IREGION).

### DESCRIPTION

Virtually all of the features of menu selection have been implemented in this package: ease of menu creation, item-selected shading, quick response to selection, and execution of an associated function. Yet, this package adds one additional feature without any degradation to the quality and efficiency of menu implementation: the selection of any irregularly shaped region from any point within that region, and without any unsightly cosmetic change.

In describing the package by means of an example, picture a map of the world, or better yet, of a particular country broken up into its individual states and/or provinces. Suffice it to say that these regions are not square but irregular in shape and that they are bordered by solid lines, as they are on a common map. Unlike the menu package or ACTIVEREGIONS package, AIREGIONS allows you to select any of these pre-set states/provinces just as if you are making a menu selection of an item. One of the nice aspects of this package lies in the fact that the package does NOT make any cosmetic changes to the irregularly shaped region, like providing some small box within the region to button in. Simply button your mouse within the solidly bordered region, anywhere in the region, and it will shade it to your particular shade and execute your defined function.

### Functionality provided:

The functions in this package allow the user to work with familiar concepts: creating and implementing windows and menus. The examples provided within this documentation should be sufficient for the user to begin setting up irregularly shaped regions.

(CREATEIR *window shade buttoneventfn helpstring region poslist*)

[Function]

*window*: the window which will contain the irregular region.

*shade*: can be either a number between 0 and 65535 for a 4 by 4 shading or a 16 by 16 bitmap (if *shade* is NIL then the default is black, 65535).

*buttoneventfn*: the function called when the region is selected. The arguments that are passed to the function are: the window containing the IREGION, the IREGION record itself, and the button which selected the IREGION.

*helpstring*: the string that is placed in the PROMPTWINDOW when the mouse is held over the item for a few seconds.

*region*: if specified, will be the region relative to *window* in which the IREGION can be found. (If *region* is NIL, the user will be prompted to sweep out a region within *window*.)

*poslist*: If specified, will be either a position or list of positions relative to *window* that are the starting points for the FILLREGION routine (i.e. a point within the desired IREGION). (if *poslist* is NIL, the user will be prompted for a position until he/she selects outside of *region*.)

**Description of use:** This is the first function that is called when actually setting up an irregularly shaped region to become sensitive to button activity. If the *region* argument is not set, then the cursor changes its shape and prompts for a region to completely surround the IREGION within the desired active window. (Note: That it is best to surround the desired IREGION as close as possible since this will save on execution time and memory usage.) A thin box will appear temporarily where the IREGION was scanned. If *poslist* is NIL, then the cursor changes into a TARGET symbol. The user should left-button mouse within desired active IREGION. Note: the IREGION must be surrounded by a border that FILLREGION can use to define the active area. Any gaps in the IREGION will cause the next routine to fill the region and anything outside with the shade provided. Mistakes can be corrected by using the REMOVE.IREGION function described below and PAINTing in the gap to retry. After left-buttoning within the desired active IREGION, the cursor continues to remain in its TARGET state. If the IREGION is split up into many different parts, those parts may be selected with the left-button also making them all active concurrently. However, when one is finished activating that one IREGION, then she/he should left-button outside of *region*. This function must be called for each desired IREGION.

#### Examples:

```
(CREATEIR window 21930 'myfunction "This is the helpstring")
(CREATEIR (WHICH) 1234 'MY.SELECTED.FN "This is the helpstring"
' (0 0 20 30) ' ((12 . 15) (2 . 29))
```

(SURROUNDIR *window shade buttoneventfn helpstring poslist inside.pos*) [Function]

*window*: the window which will contain the irregular region.

*shade*: can be either a number between 0 and 65535 for a 4 by 4 shading or a 16 by 16 bitmap (if *shade* is NIL then the default is black, 65535).

*buttoneventfn*: the function called when the region is selected. The arguments that are passed to the function are: the window containing the IREGION, the IREGION record itself, and the button which selected the IREGION.

*helpstring*: the string that is placed in the PROMPTWINDOW when the mouse is held over the item for a few seconds.

*poslist*: If specified, a list of positions relative to *window* that are the edge points for the FILLREGION routine. If NIL, the user will be prompted to define the outer border of the region desired to be active. Holding the SHIFT key will define the last point used in defining the edge. If this field is non-nil, *Inside.pos* must be specified.

*Inside.pos*: If specified, this would be the inside position in which the Fillregion routine would begin filling from. If *poslist* is non-nil, then this field must be specified.

**Description of use:** Like the CREATEIR function, this function creates IREGIONS. However, the functionality of this routine is quite different. There are times when you do not care what is within a particular region. Say, for example, you have a map of some country and you wish to surround a particular region of the country with an IREGION as you wish to denote an area rich in some mineral deposit or some other characteristic. Such a characteristic is oblivious of the borders of the country's states or provinces, streams, rivers, etc., yet you would like to make active a very general area. Upon calling this function, you are prompted to button around the area of interest. And so, in viewing the crosshairs cursor, you begin buttoning about specifying the border of the area you wish to make active, independent of what is inside it. To stop being prompted for the next edge, simply hold the SHIFT key on the keyboard, (either one will do), as you make your last button selection. At this point, the lisp DRAWCURVE function will take effect and draw the closed region you've defined. Note that the first and last points do not have to touch as the DRAWCURVE routine will connect them for you. You will also be prompted to button within the region you've marked. It is here that the Fillregion routine will begin filling your region from. When complete, this function adds the IREGION to the window and returns the iregion added.

**Examples:**

```
(SURROUNDIR window 21930 'myfunction "This is the helpstring")
(SURROUNDIR (WHICHW) 1234 'MY.SELECTED.FN "This is the helpstring"
'((5 . 5)(6 . 50)(50 . 50)(50 . 7)) '(10 . 10)))
```

(ADD.IREGION *window iregion*) [Function]

*window*: the window to which the iregion is to be added.

*iregion*: the IREGION to be added to window.

**Description:** This function will add iregion to window which will then allow mouse selection of that IREGION.

(REMOVE.IREGION *window iregion*) [Function]

*window*: the window in which the *iregion* exists.

*iregion*: the IREGION you wish to remove from *window*.

**Description:** This function removes the region from a list of active irregular regions which is stored as a window property of the window. The list of irregular active regions can be found by evaluating: (ALL.IREGIONS *window*)).

(INTERSECTING.IREGIONS? *window flg*) [Function]

*window*: a window.

*flg*: either T or NIL

**Description:** This function sets up window to allow selection of intersecting iregions. If two or more iregions overlap and this function had been called with flg = T, then when the overlapping region is selected, all of those iregions will be high-lighted and each IREGIONS BUTTONEVENTFN will be called. If flg is set to NIL, then the last IREGION created in that intersection of iregions will be selected. (Please be aware that intersecting iregions *might generate effects that you do not wish to have*. That is, if you leave the iregion "ON" (the exact same thing you see when you hold the

mouse button down on the iregion, done by inverting that iregion) and create another iregion intersecting with the first, then the mask of the second would have a partial image of the first. At this point, buttoning in an area where both regions intersect might show everything but the intersection of those regions. Sometimes, it all depends on the *order* that they are created and what iregion's mask is left on or off. Shades that are "negatives" or "equals" of one another might make matters more complex than necessary when they are intersected. It is recommend that you play with this function in order to understand how it actually works so that when you work it into your application you'll have a better idea of the functionality and end-results). If this becomes a problem, an EDIT.MASK function has been provided so that you may edit the mask of the iregion by hand. Currently, there are no programmatic methods for doing this.

(ALL.IREGIONS *window*) (Function)

*window*: a window containing IREGIONS.

**Description:** This function returns a list of all the IREGIONS attached to *window*.

(DOSELECTED.IREGION *window iregion button*) (Function)

*window*: the window associated with iregion.

*iregion*: the iregion to be activated

*button*: the button which selected iregion.

**Description:** Applied iregions BUTTONEVENTFN to window, iregion and button. This provides a programmatic way of activating a given IREGION. This does not invert the iregion.

(EDIT.MASK *iregion*) (Function)

*iregion*: the IREGION whose mask you want to edit.

**Description:** This function is provided for buttoning in places where the MASK is not set. More explicitly, TARGETing a region (while creating the regions) specifies the places where the FILLREGION routine is to create a mask. For example, if a US state contains many rivers one pixel wide, the FILLREGION routine will fill around the river, but not the river itself. This means that when the mouse is positioned on the river, the region will not shade because the mask does not have that bit turned on. However, if the mask is edited and the rivers filled in, buttoning on those rivers will activate the IREGION.

(INVERT.IREGION *window iregion*) (Function)

*window*: the window in which the *iregion* exists.

*iregion*: the IREGION targeted for shading.

**Description:** This will highlight the *iregion* with that *iregions* shade. Calling it a second time will low-light it.



(IREGIONP *iregion*) (Function)

*iregion*: the IREGION to be tested.

**Description:** This function returns NIL if *iregion* is not an IREGION datatype and returns *iregion* if it is an IREGION.

(IREGIONPROP *iregion* *prop* *newvalue*) (Function)

*iregion*: the region of which you are setting/requesting the property.

*prop*: the property in which you are interested.

*newvalue*: the new value to be assigned to *prop*.

**Description:** As with WINDOWPROP, if *newvalue* is not specified, it will return the current value of the *iregion*'s property. If *newvalue* is specified, then the property will be reassigned with that value. If a *prop* name is not one of the fields of an IREGION record, it will be stored in property-list format on the USERDATA field of the *iregion* record.

**IREGION** fields:

BUTTONEVENTFN - function called when *iregion* is selected.

USERDATA - property list format for user properties (similar to WINDOWPROP).

REGION - region relative to the window that surrounds the *iregion*.

MASK - a bitmap the same size of REGION that is blackened where the *iregion* is active.

SHADE - the shade number or bitmap used to shade the region.

HELPSTRING - the string that is printed in the *PROMPTWINDOW* when a region is held.

**Examples:**

```
(IREGIONPROP iregion 'SHADE) -- returns shade of iregion
```

```
(IREGIONPROP iregion 'SHADE 21930) - assigns new shade to iregion.
```

(SHOW.ALL.IREGIONS *window* *shade* *delay*) (Function)

*window*: the window in which the IREGIONS exist.

*shade*: the shade with which the iregions will be shown.

*delay*: the time (in milliseconds) between which each IREGION is displayed . (if *delay* is NIL, then a default of 500 is used.)

**Description:** This function will shade and unshade in *shade* (black is used if *shade* is NIL), each IREGION that has been created in the particular window. This is especially useful when the user has lost track of the number of IREGIONS within a window.

(WHICH.IREGIONS *window* *posorx* *y*) (Function)

*window*: the window in which the IREGIONS lie. (if *window* is NIL, default is window to which mouse points).

*posorx*, *y*: the location within the window where the IREGIONS can be found. These points must be local to the window's coordinates...not the screen. (if *posorx* is a position, then it will be used, otherwise if *x* or *y* are not numbers then the current mouse position is used.)

**Description:** Will return either NIL or the list of IREGIONS found in *window* and specified by *posorx*, *y*.

**Examples:**

```
(WHICH.IREGIONS)
(WHICH.IREGIONS MY.WINDOW 50 23)
(WHICH.IREGIONS MY.WINDOW '(50 . 23))
```

**Saving IRegions**

IREGIONS can be saved on a file by setting a variable to be the value returned by ALL.IREGIONS. This variable can be saved by using the file package command, UGLYVARS.

**Example:**

```
(SETQ IRS (ALL.IREGIONS (WHICHW)))
(SETQ SAVEIRSCOMS ' ((UGLYVARS IRS)))
(MAKEFILE 'SAVEIRS)
```

The file SAVEIRS can be loaded and IRS will be set. You can then add IRS to a window by doing:

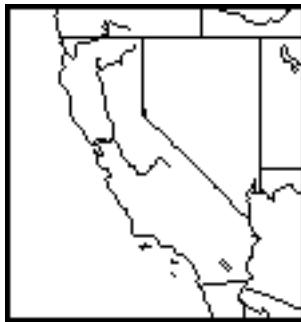
```
(WINDOWPROP (WHICHW) 'IREGIONSLIST IRS)
(WINDOWPROP (WHICHW) 'BUTTONEVENTFN 'IN.CURSOR.REGION)
```

Caution: Some properties on the USERDATA field of an IREGION might not be saved correctly such as a window which can not be saved on a file.

Window images can be saved on a file by creating a bitmap the same size as the window, BITBLT from the window to the bitmap, and then saving the bitmap with the file package command VARS.

**Example use of the AIRegions package:**

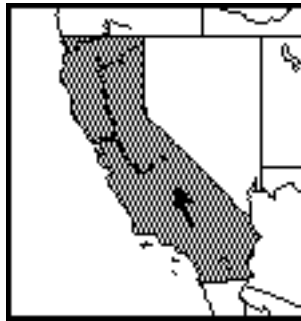
1. Open a window...about 1/4 of a screen.
2. Use the paint function provided when you right-button in the window and paint a picture.



3. With your mouse in this painted window, type in:  
(CREATEIR (WHICHW) 21930)
4. The cursor changes shape and prompts for creating a region similar to the prompt for creating a window. In this case, span a region that contains California.
5. When you are done, and the mouse button is released, the region spanned will remain temporarily on the screen. The cursor changes into a target and now prompts for a left-button within the region.

Select somewhere in California. When done, left-button the mouse outside and away from the temporarily blocked off region. (If you want to continue selecting areas of the same irregular region, in this example, the upper left corner of California, then button that area within the squared off region. As you can see, your irregular region does not necessarily have to be connected).

6. To test it out, simply button anywhere in California and it will fill to a nice shade of grey, as we have just set it up to do:



7. To create more active irregularly shaped regions, follow steps 3 through 5 above. If you want to set the selection of one of the regions to activate the execution of some function that calls RINGBELLS, and have the region shade to black upon selection, type in the following in the top level typescript window keeping the mouse within the painted window.

```
(CREATEIR (WHICHW) 65535 'IR.TESTFN)

(DEFINEQ (IR.TESTFN (LAMBDA (WINDOW IREGION BUTTON)
 (If (EQ (QUOTE LEFT) BUTTON)
 then (RINGBELLS 2))))
```

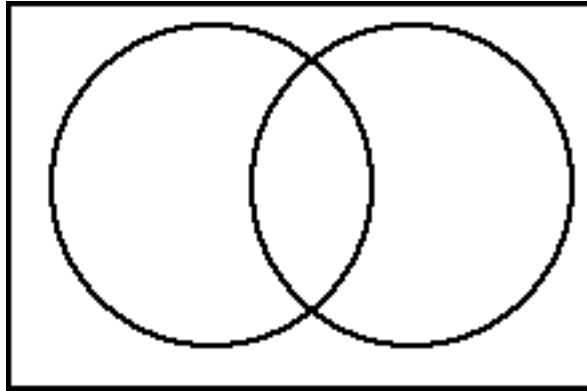
Span the cursor out over another state/region and repeat steps 3-5 above. When you button in this IREGION, the IREGION will temporarily shade black, and call the RINGBELLS function. Note that like menu selection, the function is called only when you release the button within the region. If the mouse button is held down and you move over the created IREGIONS, they will shade and unshade as you enter and exit them.

Note: if you wish to create your own shades but don't know what shades correspond to which numbers, call the function (EDITSHADE) and begin selecting points that you want shaded. When you are done, the function will return the appropriate shade number. You can also use 16x16 bitmaps for the shade of an IREGION (try (EDITBM (BITMAPCREATE 16 16)))

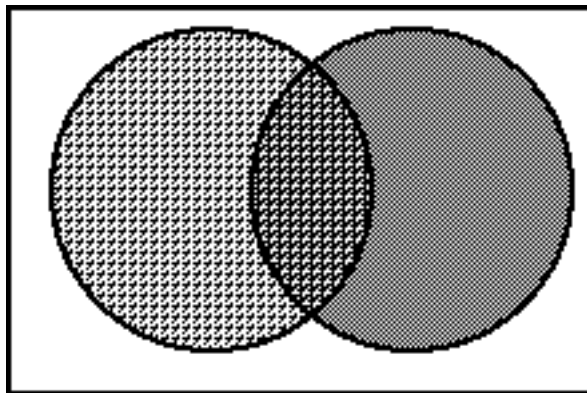
DEMO PACKAGE: To run the demo package, load AIREgions-Demo.

### **Intersecting Iregions**

1. Create a window and paint in the following:



2. Now call CREATEIR passing in this window and a shade of 4747 and surround the left circle and select inside that circle and also in the intersecting area for the area fill. Repeat this for the right circle but use a different shade (say 42405).
3. Now, with your mouse in the window, call the function (INTERSECTING.IREGIONS? (WHICHW) T). When you button in the intersection of the two circles, you should get:



4. When the mouse is released inside of the intersecting region, both IREGIONS BUTTONEVENTFN will be called.

Comments and suggestions are welcome.

---



---

## AISBLT

---



---

By: Nick Briggs (Briggs.pa@xerox.com)

This document last edited on September 21, 1988.

### INTRODUCTION

The AISBLT module provides a fast(er) interface for reading AIS format files into Lisp bitmaps. It does not provide all the arcane features found in the READAIS module.

### CLIENT INTERFACE

The functions provided by the AISBLT module which are intended to be used by clients are

(AISBLT.BITMAP *FILE SOURCE-LEFT SOURCE-BOTTOM DESTINATION DESTINATION-LEFT  
DESTINATION-BOTTOM WIDTH HEIGHT HOW FILTER*) [Function]

The *SOURCE-LEFT*, *SOURCE-BOTTOM*, *DESTINATION*, *DESTINATION-LEFT*, *DESTINATION-BOTTOM*, *WIDTH*, and *HEIGHT* arguments are interpreted in the same way as the corresponding arguments to BITBLT. *FILE* is either an open stream, or a filename. If a filename is provided it will be passed to FINDFILE, which searches the directories specified by the special variable

AISDIRECTORIES [Variable]

which should be a list of directories where the AIS file is likely to be found.

The argument *HOW* should be one of the atoms FSA, :FSA, TRUNCATE, :TRUNCATE. *HOW* is only applicable in the cases where the source and destination are a different number of bits per pixel (source bpp > destination bpp). If *HOW* is not specified, it defaults to FSA. FSA indicates that the source should be reduced to the bits per pixel of the destination by applying the Floyd-Steinberg dithering algorithm, as described in Newman & Sproull, Principles of Interactive Computer Graphics, pg. 226. TRUNCATE indicates that only the high order bit(s) of the source should be used.

The function

(AISFILEHEADER *STREAM*) [Function]

Can be used to determine whether a file has a well formed AIS header, and what the attributes indicated in the header are. The result of the function is a property list describing the AIS attributes:

:RASTER [Key]

The :RASTER property will always be present. The value is also a property list

:SCAN-COUNT [Key]

An integer value indicating the number of scan lines in the image

:SCAN-LENGTH [Key]

An integer value indicating the number of pixels in a scan line of the image

:SCAN-DIRECTION [Key]

An integer, indicating the direction of the scan. Scan direction 3 is top to bottom, left to right, and is the only scan direction that this package will deal with at this time.

:SAMPLES-PER-PIXEL [Key]

An integer, indicating the number of samples per pixel. This package will only deal with files having one sample per pixel at this time.

:CODING-TYPE [Key]

An unsigned integer indicating the coding type of the raster image. A value of 1 indicates uncompressed array format, and is the only type recognized by this package at this time. For convenience, the constant

AIS-RASTER-CODING-UCA [Constant]

is bound to the value 1. If the raster coding types are extended, more constants will be defined.

The rest of the properties are coding type dependent. For the AIS-RASTER-CODING-UCA file, the following properties are present:

:BITS-PER-SAMPLE [Key]

An unsigned integer, indicating number of bits per sample

:WORDS-PER-SCAN-LINE [Key]

An unsigned integer, indicating how many 16 bit words form a single scan line of the image.

:SCAN-LINES-PER-BLOCK [Key]

A signed integer, indicating how many scan lines are present before there is block padding. A value of -1 indicates no blocking.

:PADDING-PER-BLOCK [Key]

A signed integer, indicating how many padding words per block. A value of -1 indicates no blocking.

:PLACEMENT [Key]

The placement property is optional. The value is a property list with keys :LEFT, :BOTTOM, :WIDTH, and :HEIGHT. The values are unsigned integers.

:PHOTOMETRY [Key]

The photometry property is optional. The value is a property list with keys :SIGNAL (integer), :SENSE (integer), :SCALE (integer), :SCALE-A (pair of integers), :SCALE-B (pair of integers), :SCALE-C (pair of integers), :SPOT-TYPE (integer), :SPOT-WIDTH (integer), :SPOT-LENGTH (integer), :SAMPLE-MIN (integer), and :SAMPLE-MAX (integer).

A complete description of the meaning of the photometry parameters can be found on page 38 of the AIS format description, filed on {indigo}<altodocs>aismanual.press.

---

---

## Analyzer

---

---

By: Maxwell (Maxwell.pa@Xerox)

### INTRODUCTION

The Analyzer package is used by the Proofreader (see PROOFREADER). It defines a class of analyzers, of which the proofreader is but one. Later, analyzers will be developed for languages other than English.

---

---

**AUTOSAMEDIR**

---

---

By: Mitchell L Model  
473 Edgell Road  
Framingham, MA 01701

Uses: SAMEDIR.LCOM

**INTRODUCTION**

This package is an extension to the SAMEDIR Library package so that if AUTOSAMEDIRFLG is non-NIL, MAKEFILE will automatically switch to the directory its file argument was originally made to instead of invoking the SAMEDIR dialogue. (It uses the FILEDATES property of the file.)

Notice and Acknowledgement: This package was developed while the author was an employee of Applied Expert Systems, Inc. (Apex), Cambridge, MA. The author thanks the company for its support and assumes full responsibility for the contents and maintenance of this package.

**SOFTWARE REQUIRED**

AUTOSAMEDIR.LCOM

SAMEDIR.LCOM

**FIXES**

Extension to SAMEDIR package so that if AUTOSAMEDIRFLG is non-NIL, MAKEFILE will automatically switch to the directory its file argument was originally made to instead of invoking the SAMEDIR dialogue. (It uses the FILEDATES property of the file.)



---

---

## AUXMENU

---

---

By: David Newman (Newman.pasa @ Xerox.COM.ARPA)

AUXMENU is a Lispusers package that creates a middle-button background menu. This menu acts like the right-button background menu that exists in any Interlisp sysout in most respects. The menu includes commonly used Interlisp functions that require no arguments to be useful.

### Global Variables

|                                           |            |
|-------------------------------------------|------------|
| MiddleButtonBackgroundMenuCommands        | [Variable] |
| DefaultMiddleButtonBackgroundMenuCommands | [Variable] |
| MiddleButtonBackgroundMenu                | [Variable] |

MiddleButtonBackgroundMenuCommands is a list of MENU items (in the same format described in the Interlisp Reference Manual) which is used to create the middle button background menu. DefaultMiddleButtonBackgroundMenuCommands is the default value of MiddleButtonBackgroundMenuCommands. Individual users may reset MiddleButtonBackgroundMenuCommands, or they may add to or change it. If the user changes this variable, MiddleButtonBackgroundMenu should be set to NIL. All variables are initialized by the package as it is loaded. All necessary interaction is performed in the promptwindow, and the result of any menu item is printed to the promptwindow if it is non-nil.

### Default Menu Items

Login - This item performs a (LOGIN) via the promptwindow.

Greet - This item performs a (GREET).

Logout - This item does a (LOGOUT).

Cleanup - This item performs a (CLEANUP) via the promptwindow.

Reclaim - This item executes (RPT 5 (QUOTE (RECLAIM))).

Closeall - This item closes all currently open files by performing (CLOSEALL).

Open Files - This item lists the currently open files to the promptwindow using (OPENP).

Connect - This item prints the currently connected directory to the promptwindow. The subitems of this item make the connected directory the one shown as the item. The 'Default' subitem connects the system to the value of the variable LOGINHOST/DIR. The 'Other' subitem prompts the user for a directory name, makes that the connected directory, and adds it to the menu.

VMem Size - This item prints the current size of the virtual memory file (in pages) to the promptwindow. It uses (VMEMSIZE).

Free Pages - The number of free pages in device DSK are printed to the promptwindow. (DISKFREEPAGES) is the function that provides this information.

Disk Partition - This item prints the name of the current partition to the promptwindow. The printed value is the result of calling (DISKPARTITION).

Volume Display - Turns the volume display window on via (VOLUMEDISPLAY 'ON).

Default Printers - Types the value of DEFAULTPRINTINGHOST to the promptwindow.

File Changes - Lists the result of (FILEPKGCHANGES) to the promptwindow.

---

## BACKGROUNDIMAGES

---

By: Burwell (Burwell.pa@Xerox.com)

Accessory files:

Background-DurerCat.bitmap  
Background-Parc.press  
Background-Rhine.press  
Background-Steinheim.press  
Background-TwoDollar.press  
BackgroundMenu.dfasl  
BitMapFns.lcom

This document last edited on September 8, 1988.

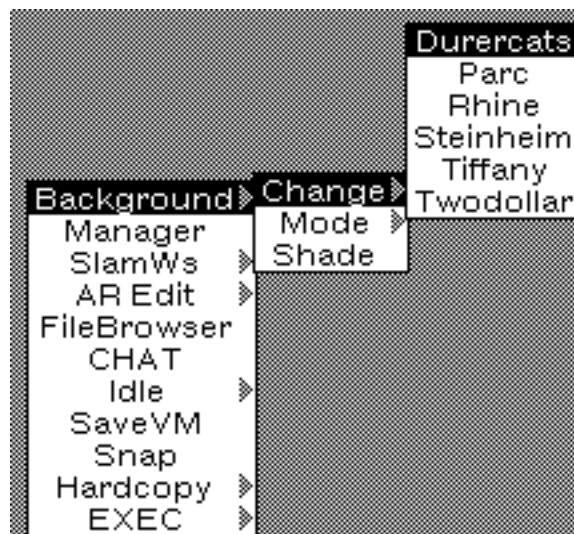
### INTRODUCTION

BackgroundImages is a module which makes it easy to apply graphically interesting static images to the background of one's Lisp screen. To use the package in the simplest way, load it and call

(BACKGROUND.SETUP)

[Function]

This will put an entry called "Background" on your background menu (in a manner compatible with the module BACKGROUNDMENU), so that it will look something like this:



If (as shown) you select one of the subitems of "Background>Change," your background will be painted with the image whose name you selected. The background images currently available are

DurerCats: a reflected picture of a cat from an engraving by Albrecht Durer  
Parc: a picture of the Xerox Palo Alto Research Center

Rhine: a picture of a village on the Rhine river  
 Steinheim: a picture of a relatively unfortified castle  
 TwoDollar: a picture of part of a two dollar bill

If the image you select is a different size than your screen, you may want to control how the image is applied. There are three different image painting modes: "Center," which centers the image on the screen and paints gray in the remaining space; "Tile," which tiles the screen with the image; and "Reflect," which tiles the screen with edge-matched reflections of the image. (This last mode is particularly effective with the DurerCats image.) To change the mode, select one of the subitems of "Background>Mode." To the currently set mode, just select "Background>Mode" itself. Note that once you have changed the mode, to see its effect, you must reapply the background image (by selecting "Background>Change>ImageName").

If you want a less busy background, you can use a plain gray. To apply it, just select "Background." To change the shade of gray, select "Background>Shade." Again, to see the effect of the shade change, you must reapply the background shade (by selecting "Background").

## DETAILS

Background images to be used with this module must be represented in files that can be read by either HREAD or READPRESS. For convenience, they should be named according to the conventions mentioned below under BACKGROUND.FILES.

BackgroundImages does take some pains to reduce user wait time. First, it is very lazy about file interactions, and defers them until it is quite clear they cannot be avoided. And second, when one selects a background, it is cached so that changing back to it will be significantly faster than fetching it the first time. Since the cached background bitmaps consume quite a bit of space, they can be removed by the GAINSPACE mechanism.

The public interface to this package, more fully described, is as follows:

(BACKGROUND.SETUP NAMES) [Function]

Puts an entry on the background menu which enables users to change backgrounds easily. The entry will be labeled "Backgrounds" and if invoked will turn the screen background gray. The entry will have several subitems, each labeled with the name of the background image it will, if selected, put on the screen. The argument NAMES is meant to specify the names of the images; it must be a list either of dotted pairs (whose CAR is the name of an image and whose CDR is the name of the file in which a representation of that image can be found) or of atoms (each of which is the name of an image). If NAMES is NIL, BACKGROUND.SETUP will call (BACKGROUND.FILES) to generate a set of background image names.

(BACKGROUND.FILES WHICH) [Function]

Returns a list of dotted pairs whose CAR is the name of an image and whose CDR is the name of the file in which a representation of that image can be found. Generates this list by looking on LISPUSERSDIRECTORIES for files of the form "background-\*.bitmap" or "background-\*.press"; all such files are taken to be representations of background images. Image representation files that are not named and located according to this convention will have to be specified directly to BACKGROUND.SETUP. If WHICH is T, it will search all the LISPUSERSDIRECTORIES; otherwise it will search till it finds the first directory with background images in it.

(BACKGROUND.FETCH *NAME FILENAME MODE*)

[Function]

Causes the image whose name is *NAME*, and for which there is a representation in file *FILENAME*, to be applied to the screen background. It is this function which the background menu subitems call to apply new images. If *FILENAME* is not specified, BACKGROUND.FETCH will attempt to find an image representation file whose name is either "background-*NAME*.bitmap" or "background-*NAME*.press" on any of the LISPUSERSDIRECTORIES. *MODE* specifies how the image will be applied to the background if it is a different size than the screen. *MODE* should be one of the atoms CENTER, TILE, or REFLECT; it defaults to CENTER. CENTER causes the image to be centered with a white border around it; TILE causes the image to tile the screen; and REFLECT causes the image to tile the screen such that each tile is a reflection of those adjacent to it.

(BACKGROUND.MODE *MODE*)

[Function]

Sets and accesses the mode (as described above) which will be passed to BACKGROUND.FETCH when the latter is invoked from the background menu subitems. *MODE*, if provided, gives the new mode setting. Returns the previous mode setting.

(BACKGROUND.SHADE *NEWSHADE*)

[Function]

Changes the default background shade.

(BACKGROUND.CENTER *BITMAP*)

[Function]

Returns a screen-sized bitmap with *BITMAP* centered in it with a border colored with the default background shade.

(BACKGROUND.TILE *BITMAP*)

[Function]

Returns a screen-sized bitmap that is tiled with *BITMAP* with one of the tiles centered.

(BACKGROUND.REFLECT *BITMAP*)

[Function]

Returns a screen-sized tiled bitmap such that each tile is a reflection of those adjacent to it and such that the center tile is a copy of *BITMAP*.

---

---

**BACKGROUNDMENU**

---

---

By: Mike Dixon  
New Owner: Burwell (Burwell.pa@Xerox.com)

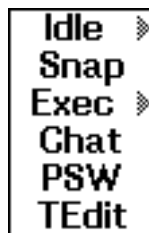
**INTRODUCTION**

If you love to load all those fun LispUsers packages but can't deal with background menus that look like this:



```
Inspecticide
Sketch
DInfo
VStats
DumpCache
AR Edit
FileBrowser
SaveVM
Hardcopy
SendMail
Idle
Snap
Lisp Listener
Chat
PSW
TEdit
Keyboard
```

don't despair! With just a few quick calls you can have a background menu that looks like this:



```
Idle
Snap
Exec
Chat
PSW
TEdit
```

(don't worry, they didn't disappear, they're just hiding under "Exec").

**DESCRIPTION**

BackgroundMenu defines several functions for rearranging your background menu to suit your taste.

|                                                                                                                                                                                                                                                                                                                                                                                      |            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| (BkgMenu.rename.item <i>item newname</i> )                                                                                                                                                                                                                                                                                                                                           | [Function] |
| changes the name of a background menu entry                                                                                                                                                                                                                                                                                                                                          |            |
| (BkgMenu.move.item <i>item superitem atend</i> )                                                                                                                                                                                                                                                                                                                                     | [Function] |
| makes <i>item</i> a subitem of <i>superitem</i> . If <i>atend</i> it is placed after any subitems of <i>superitem</i> ; otherwise it is placed before them. If <i>superitem</i> is NIL <i>item</i> is placed at the top level of the menu.                                                                                                                                           |            |
| (BkgMenu.reorder.items <i>superitem atend</i> )                                                                                                                                                                                                                                                                                                                                      | [Function] |
| just like BkgMenu.move.item but moves a list of items. Useful for changing the order of the items in a menu.                                                                                                                                                                                                                                                                         |            |
| (BkgMenu.remove.item <i>item</i> )                                                                                                                                                                                                                                                                                                                                                   | [Function] |
| throws <i>item</i> out of your background menu.                                                                                                                                                                                                                                                                                                                                      |            |
| (BkgMenu.fixup)                                                                                                                                                                                                                                                                                                                                                                      | [Function] |
| BackgroundMenuTopLevelItems                                                                                                                                                                                                                                                                                                                                                          | [Variable] |
| BackgroundMenuFixupMode                                                                                                                                                                                                                                                                                                                                                              | [Variable] |
| each top level item which isn't on the global BackgroundMenuTopLevelItems is made a subitem of BackgroundMenuSuperItem. If BackgroundMenuFixupMode is 'top they're added before any subitems of BackgroundMenuSuperItem, if it's 'bottom they're added after, and if it's NIL items moved from the top are added at the top and items moved from the bottom are added to the bottom. |            |
| (BkgMenu.subitems <i>item</i> )                                                                                                                                                                                                                                                                                                                                                      | [Function] |
| returns a list of the subitems of <i>item</i> (or the top level items, if <i>item</i> is NIL).                                                                                                                                                                                                                                                                                       |            |
| (BkgMenu.add.item <i>item superitem atend</i> )                                                                                                                                                                                                                                                                                                                                      | [Function] |
| adds a new menu item <i>item</i> as a subitem of <i>superitem</i> . If <i>atend</i> it is placed after any subitems of <i>superitem</i> ; otherwise it is placed before them. If <i>superitem</i> is NIL <i>item</i> is placed at the top level of the menu                                                                                                                          |            |

## EXAMPLES

As an example of using BackgroundMenu, this is what I've got in my init file (which produces the changes shown above) (note that i've already loaded LISTEN):

```
(BkgMenu.rename.item "Lisp Listener" " Exec ")
 (* "Lisp Listener" is just too long. the blanks before and after Exec are just there to
 improve the spacing)

(SETQ BackgroundMenuTopLevelItems '(Idle Snap " Exec " Chat PSW TEdit))

(SETQ BackgroundMenuSuperItem " Exec ")

(BkgMenu.fixup)
 (* Push everything i don't use regularly under the now-renamed Lisp Listener)

(BkgMenu.reorder.items BackgroundMenuTopLevelItems)
 (* and put the top level items in the order i prefer)
```

If I later add more packages which add junk to the top level of my background menu, just calling (BkgMenu.fixup) again will hide anything new under " Exec " with the rest of the junk.

When any of the above functions (except `BkgMenu.add.item`) require you to specify an item, you can usually just give a string with the menu entry (or an atom, which is coerced to a string). The case has to be correct, and blanks have to be in the right place. The function will do a breadth first search of the background menu and all its submenus to find such an entry. If for some reason you have the same entry in more than one menu, you'll have to disambiguate it. To do this, you pass a list for the item, where the first thing in the list is the menu entry, and the rest of the list is a path through the tree to find it. For instance, the item `(one two three)` means find an entry whose text is "three", then find an entry in the tree underneath it whose text is "two", and then find an entry under that whose text is "one".

The item argument to `BkgMenu.add.item` is a standard menu item, i.e. a list of `(label form help.string)`.

All of the functions return `T` if they were able to do as asked and `NIL` otherwise (you tried to do something with a menu entry which isn't there, or you tried to make a circular menu structure). The only exception to this rule is `BkgMenu.subitems`, which as previously mentioned returns a list of the subitems, or the atom `NotAnItem` if it's given a nonitem.



---



---

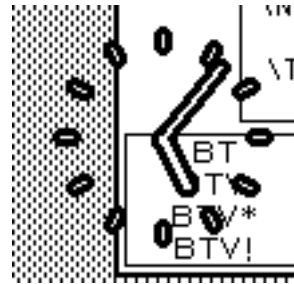
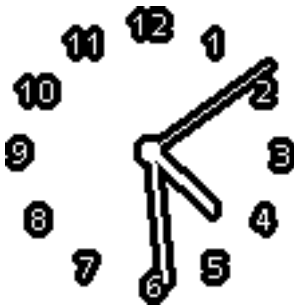
## BICLOCK

---



---

By: Bernt Nilsson (Bernt Nilsson:Ida:LiTH or BKN%LiUIDA.UUCP@Seismo.ARPA)



### INTRODUCTION

BICLOCK, a realtime screen clock, with hour, minute and second hands. It behaves as a kind of icon, in that it is partly transparent. The snapshot to the right above shows that the greater part of the window's area is transparent, in this case the gray background and part of the PSW window show through. The clock-image may be either black or white, and the shadow around the image-parts the opposite color: The reason for having a shadow is to make the clock easier to read. Markers and/or digits may be chosen.

### Seconds

The seconds hand is optional. When on, the clock process consumes a lot of time, but because (1) it blocks very frequently and (2) shuts itself off temporarily, if the "load" is higher than usual, that does not seem to irritate at all. The "load" is measured by the continuous average of time spent in block, i.e. each round robin. A "usual load" is computed by a second level of average over the "load" value. The "usual load" is limited by upper and lower "reasonable" constants. If "load" is more than 10% above "usual load" then the seconds hand is shut off. This is a rather heuristic method, but works reasonably well. When the seconds hand is shut off, the image is updated approximately once per minute.

### WINDOW COMANDS

Most window commands are applicable. The clock will preserve its "square" form when shaped. More details of the hour/minute markers/digits will appear only when the window is large enough, less if the window is smaller. The digits' fonts are chosen among fonts already "in core".

### Left button

Left buttoning the window will print the current date (and alarmtime) in the promptwindow. Holding down left shift key while buttoning the window will copyselect current date, by BKSYSBUF.

### Middle button, Alarm

There is a simple alarmclock facility built into the clock. That and some other options may be accessed by middlebuttoning the window and selecting the appropriate command from the menu that pops up.

Choosing the command "Set Alarm" will change the clock into showing current alarmtime and attach an adjust menu below the clockwindow. If no alarm time is known, current daytime plus 1 minute is used as default. Any change to the alarmtime is shown both by the clock and printed in the promptwindow. Buttoning "OK!" in the attached menu exits the adjust mode. When the alarmtime is reached, you will be aware of that fact... to shut the alarm off, use the "Alarm Off" in the popup menu.

## FUNCTIONALITY

One clockwindow is usually created automatically when the file is loaded. More clockwindows may be created by the following function call:

(BICLOCK *props*) [Function]

Creates a new window. Value is the new window. The call may be done either with a free property list as first arg or spread. See properties below. The different properties are also controlled by the variables below. It is allowed to have more than one clock running, but see "bugs" below.

When the BICLOCK file is loaded, the following variables are initialized, some through INITVARS, so that you can set them before loading the file:

BICLOCKUSERPROPS [Variable]

Change this to customize the overall behavior of the clock. Value must be a free property list. Defaults to NIL.

BICLOCKDEFAULTPROPS [Variable]

These are the default properties, don't modify this, modify BICLOCKUSERPROPS above.

Properties are searched for in the following order.

1. those specified in the function call.
2. those specified by the BICLOCKUSERPROPS variable.
3. those specified by the BICLOCKDEFAULTPROPS variable.

BICLOCKINITIALPROPS [Variable]

Modify this if you like some specific property for the initial clock. BICLOCK is called with these properties when the file is loaded, defaults to NIL. If you want no initial clock, set it to (CREATE NIL).

BICLOCKWINDOW [Variable]

The window of the clock that is created when the file is loaded.

BICLOCKIDLEPROPS [Variable]

(IDLE.BICLOCK) [Function]

BICLOCK is called with BICLOCKIDLEPROPS from the Idle function IDLE.BICLOCK, defaults to start bouncing at center of screen.

## Properties

Recognized properties and allowed values are:

|         |                                                                                                      |
|---------|------------------------------------------------------------------------------------------------------|
| SECONDS | T (default), if seconds hand shall be used, or NIL, if no seconds                                    |
| COLOR   | interior color, one of WHITE (default) or BLACK                                                      |
| MARKS   | NIL (default), if no marks should be used, or one of HOURS, HOUR&MINUTE, 3/6/9/12 or a modulo number |

|             |                                                                                                               |
|-------------|---------------------------------------------------------------------------------------------------------------|
| DIGITS      | NIL, if no digits, or one of HOUR (default), 3/6/9/12 or a modulo number                                      |
| DIGITFORMAT | one of ARABIC or ROMAN                                                                                        |
| CHIME       | NIL (default), no chime, or one of HOUR, QUARTER or a modulo number                                           |
| ALARM       | NIL (default), no alarm, or a the standard string representation of a date                                    |
| SIZE        | a number describing both width and height of the clock window, defaults to 119                                |
| HORIZONTAL  | generic place on screen, one of LEFT, CENTER, RIGHT (default), or a number to specify left border of window   |
| VERTICAL    | generic place on screen, one of BOTTOM, CENTER, TOP (default), or a number to specify bottom border of window |
| REGION      | if nonNIL, a region, overrides SIZE, HORIZONTAL and VERTICAL for clockwindow                                  |
| WINDOW      | if nonNIL, a window, overrides REGION, SIZE, HORIZONTAL and VERTICAL                                          |
| CREATE      | T (default), create a clock, or NIL, do not create, probably only useful if you want no initial clock...      |

The nonformal meaning of the Modulo numbers mentioned above are: "1" means: on each possible place. "2" means: on even places and so on...

**KNOWN BUGS:**

1. If the window is reshaped when in "set alarm" mode, the whole window group will be square.
2. Running more than one clock concurrently with seconds hand on, may behave a bit round robin because the load sensor is local for each clock and "senses" the other clocks.
3. Overlapping two clock windows creates a funny image in the intersecting area.

---

---

**BITMAPFNS**

---

---

By: Larry Masinter (Masinter.PA@Xerox.COM)

This document last edited on 4-mar-87

(READBINARYBITMAP *WIDTH HEIGHT FILE*) [Function]

reads a series of bytes from *FILE* and creates a *WIDTH* times *HEIGHT* bit map with contents. Note that each scanline of the bit map is rounded up to the nearest multiple of 16 bits (two bytes).

(WRITEBINARY BITMAP *BITMAP FILE*) [Function]

writes out *BITMAP* to *FILE* in format read by READBINARYBITMAP. Please note that READBINARYBITMAP must be supplied with width and height.

(WRITEBM *FILE BITMAP*) [Function]

writes *BITMAP* on *FILE* first preceding with width and height (in binary) such that it can be read in with READBM.

(READBM *FILE*) [Function]

reads width, height, and then appropriate size bit map.

(WRITEBMLST *FILE LST*) [Function]

writes a list of bit maps on *FILE*.

(READBMLST *FILE*) [Function]

reads a list of bit maps.

The following functions open and close *FILE*.

(READPRESS *PRESSFILE*) [Function]

reads press file *PRESSFILE* and returns a bit map. Can only handle press files generated by PRESSBITMAP and a couple of other utilities. Has no smarts, and is not easily extended.

(WINDOWBM *BITMAP POSITION*) [Function]

creates and returns a window containing image of *BITMAP*. Will be at *POSITION* or (GETPOSITION).

# XEROX

Xerox Corporation  
Palo Alto Research Center  
3333 Coyote Hill Road  
Palo Alto, California 94304

## Bitmap-Gallery

Sampler and Documentation for bitmaps, especially those useful as screen background.

Eventually I'll rationalize the names and packages. Most of these bitmaps have been snapped from the screen and are smaller than they might appear. They usually look better on screen than they do on paper. Please send suggested additions to Foster.PA.

*WARNING: This file is only usable in Lyric.*

Shades

Author: Gregg Foster et al <Foster.pa>  
Filed on: {FS8:PARC:XEROX}<FOSTER>LISP>USERS>BITMAP-GALLERY.TEDIT;5  
Last filed: March 23, 1988 18:07 PDT (Wednesday)  
Copyright © 1987 by Xerox Corporation. All rights reserved.

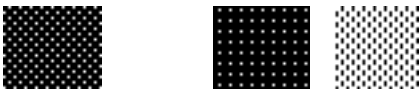
**BITMAP-GALLERY**

## Some Basic Shades

Use (IL:EDITSHADE) to create your own.



IL:PLAINSHADE, IL:GRAYSHADE, IL:GRAYSHADE1, IL:GRAYSHADE2



IL:GRAYSHADE3, IL:GRAYSHADE4, IL:DEFAULTSCREENS SHADE

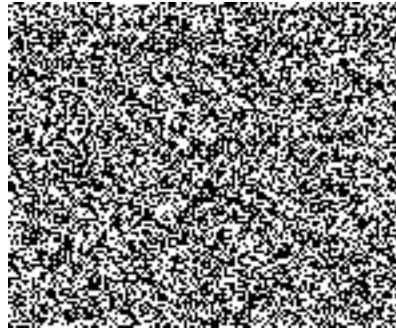
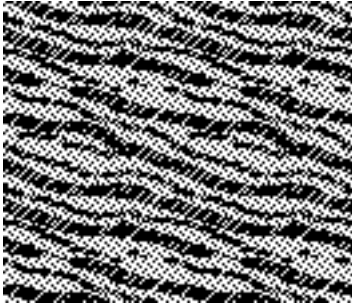


IL:WAVE-TEXTURE, IL:WAVE2-TEXTURE, IL:MESH-TEXTURE

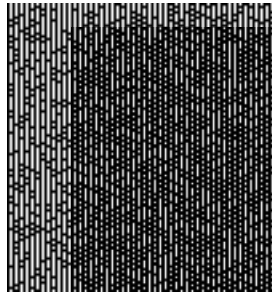
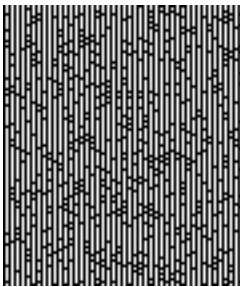


IL:DI-TEXTURE, IL:DARK-DI-TEXTURE

## From Gregg Foster



XCL-USER::~\*TESSEL-BM\*, XCL-USER::~\*RANDOM-BM\*

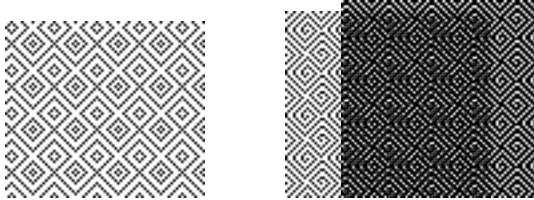


XCL-USER::~\*GRANITE-LIGHT-BM\*, ...-MEDIUM-..., ...-DARK-...

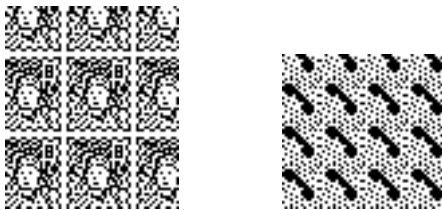
**From {PHYLUM}<Foster>Lisp>Users>GRANITE**, using the function `il:|MakePseudoRandomBitmap|` you get bitmaps that look something like the above (they come in three shades, LIGHT, MEDIUM, and DARK (symbols for 3, 2, 1, respectively); and large variety. The three bitmaps above are examples of the kind of thing you get.

[Example usage: `(setq my-bitmap (il:|MakePseudoRandomBitmap| NIL 64 64 3))`, makes a 64x64 light PseudoRandom bitmap.]

## From Stanley's Tool Works



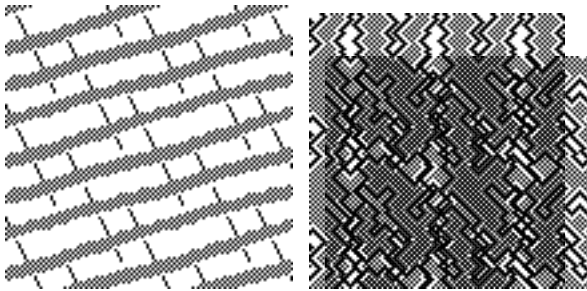
IL:LIGHTWALLPAPER, IL:WALLPAPER, IL:DARKWALLPAPER



IL:\*STAMP-BITMAP\*, IL:\*PHONE-BITMAP\*

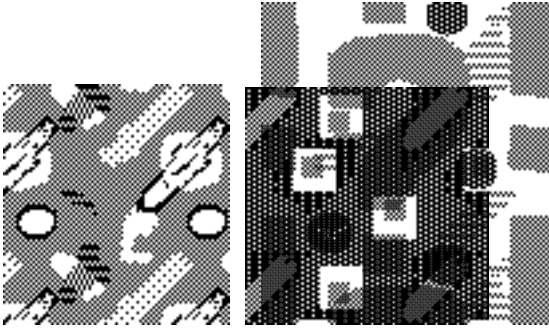
Wickberg

## From Andreas Wickberg



IL:AVANTBACKGROUND0, ...1, ...2





IL:AVANTBACKGROUND3, ...4, ...5

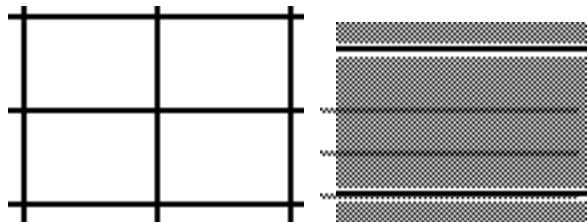


IL:AVANTBACKGROUND6, ...7, ...8

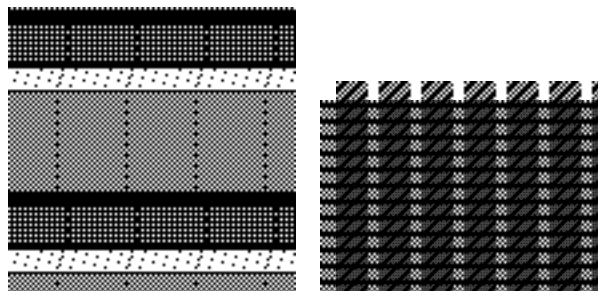


IL:AVANTBACKGROUND9, ...10, ...11

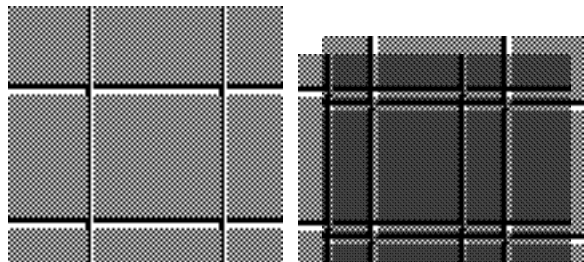
From Stu Card



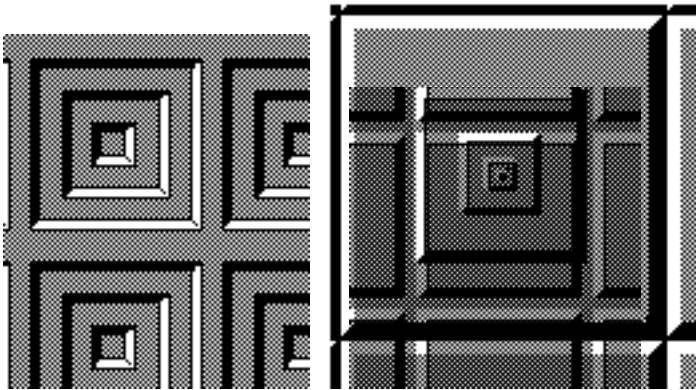
IL:ROOM.BM, IL:LINE1.BM, IL:LINE2.BM,



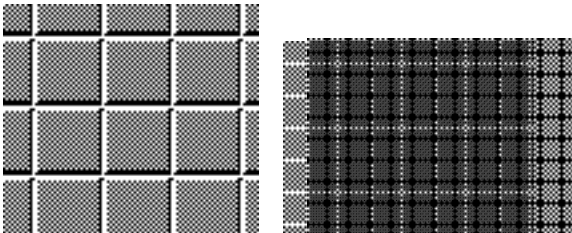
IL:LINE3.BM, IL:LINE4.BM, IL:LINE5.BM



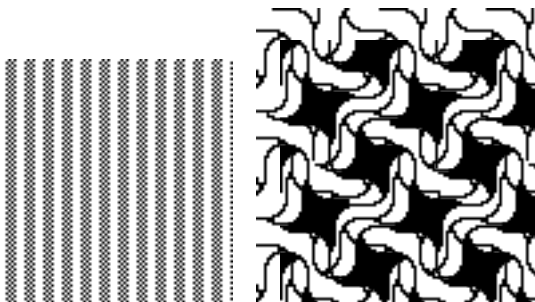
IL:SQUARE1, IL:SQUARE2, IL:SQUARE3.BM, ■



IL:SQUARE4.BM, IL:SQUARE5.BM, IL:SQUARE6.BM

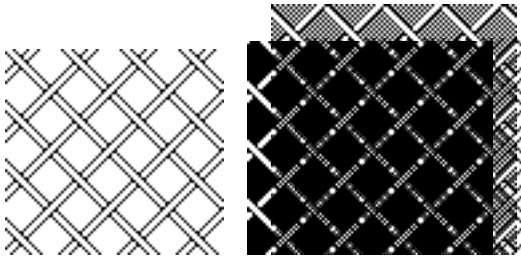


IL:SQUARE7.BM, IL:SQUARE8.BM, IL:SQUARE9.BM



IL:SHIRT1.BM, IL:CURLY, IL:CURLY1

## BITMAP-GALLERY



IL:WOVEN, IL:WOVEN1, IL:WOVEN2, IL:WOVEN3

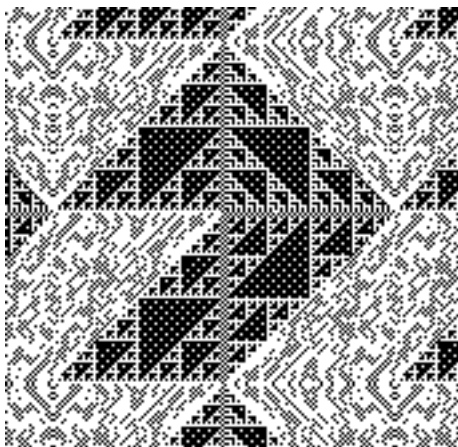
## From Harley Davis



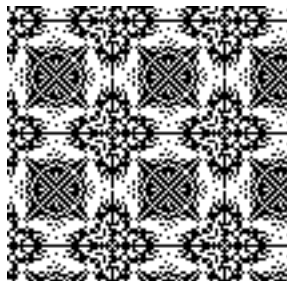
XCL-USER::\*EYE-BM\*

Corbett

## From John Corbett



XCL-USER::\*FRACT-BM\*



XCL-USER::\*MANDALA-BM\*



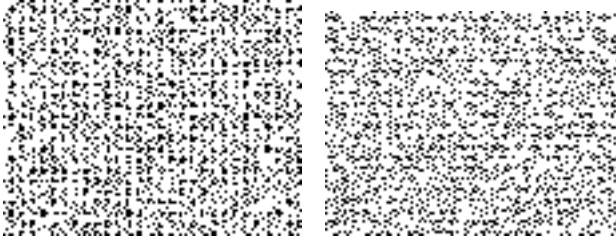
XCL-USER::\*STATIC1-BM\*



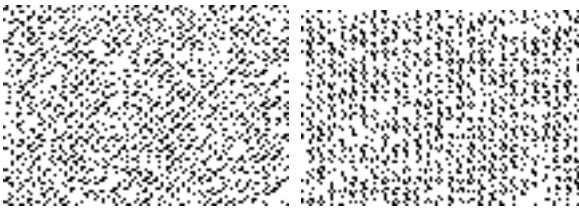
XCL-USER::\*STATIC2-BM\*

Fabrics

## Fabrics ( from John Corbett and Gregg Foster)



XCL-USER::\*TWEED-BM\* XCL-USER::\*CHAMBRAY-BM\*



XCL-USER::\*CANVAS-BM\* XCL-USER::\*CORDUROY-BM\*

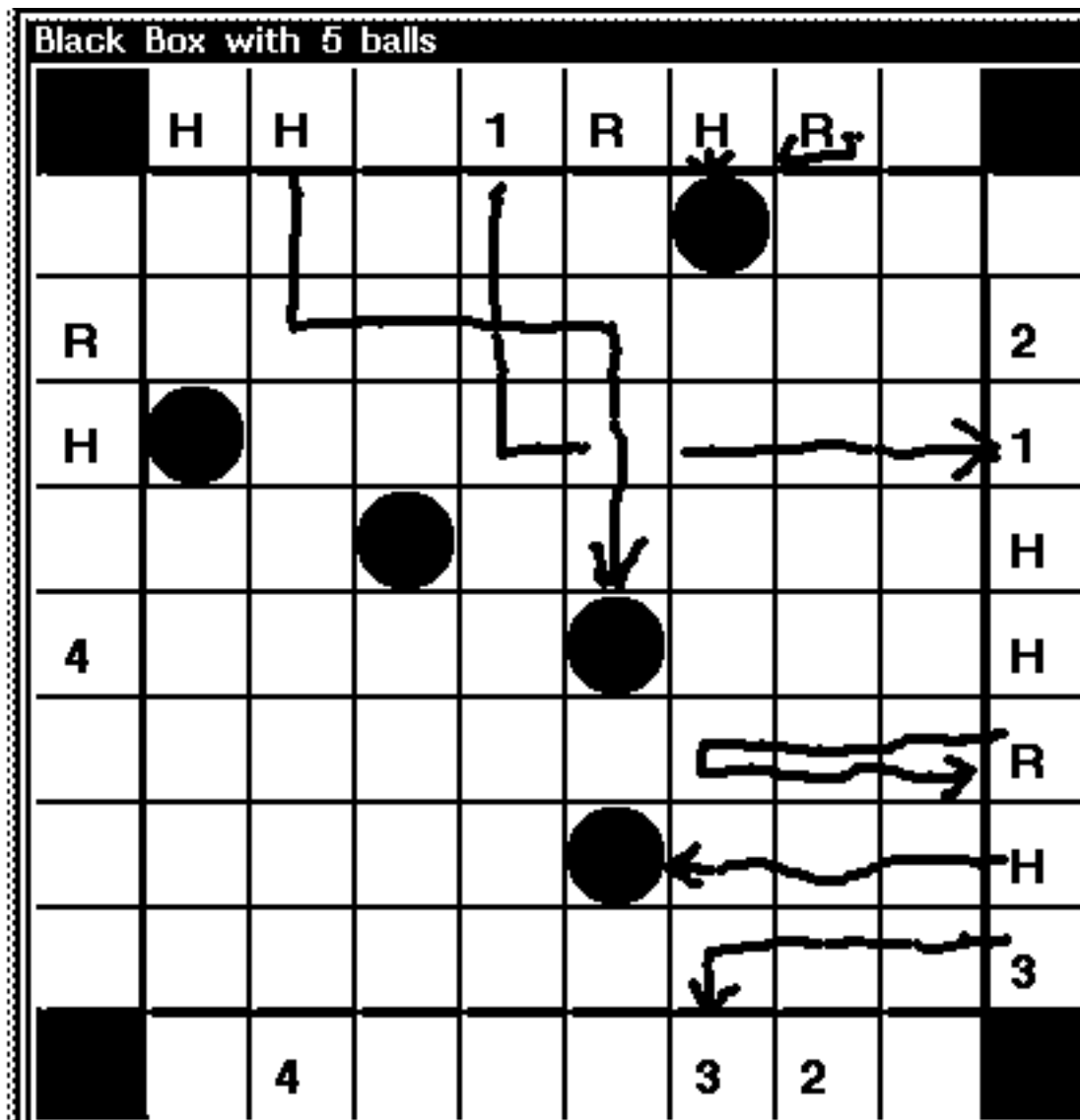


XCL-USER::\*SEERSUCKER-BM\* XCL-USER::\*BURLAP-BM\*

You can use the function FABRICIZE (included in this file) on an arbitrary bitmap to return a fabric-like bitmap (the original bitmap is unaltered).

USAGE: (FABRICIZE BITMAP)

# Black Box A Game



|                                                                                   |  |  |  |   |
|-----------------------------------------------------------------------------------|--|--|--|---|
|  |  |  |  | H |
|                                                                                   |  |  |  | R |
|  |  |  |  | H |



---

**BUTTONS**

---

Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox)

Created: November, 1986  
Last Edit: December 2, 1988

**DESCRIPTION**

BUTTONS is a facility for creating icons which will trigger actions when they are clicked in. Each button has a label and an action associated with it. There are three different things which one can do with buttons: Trigger the action, move the button, and bring up a button command menu. These are initiated by use of the left, middle, and right mouse buttons within each button. The command menu is also available through the background menu entry "Button Control." The button world can be tailored (somewhat) using button properties.

Clicking with the Left mouse button on a button and then letting up causes the action associated with the button to be taken. If the action is a list it will be evaluated, otherwise it is stuffed into the system read buffer. The button inverts while the action is being taken.

Clicking with the Middle mouse button allows one to move the button on the screen. The button moves on a grid, unless the left shift key is down.

Clicking with the Right mouse button brings up a menu with the following commands:

- Redisplay** -- redisplay this button
- Move** -- same as clicking with the middle button
- Copy** -- make a copy of this button and move it
- Edit** -- invoke the structure editor on the label and the action of this button
- Close** -- close this button (but keep it)
- Rollout: **Close All Buttons** -- close all open buttons
- Delete** -- delete this button
- Rollout: **Delete All Buttons** -- delete all existing buttons
- Create Button** -- make a new button and move it

**Expose Buttons** -- redisplay all buttons (including previously closed ones)  
**Align Buttons** -- prompts for alignment axis, then for successive buttons to line up  
**Save Buttons** -- save current buttons and button properties in default data file  
Rollout: **Save Some Buttons** -- prompt for file and for which buttons to save  
**Restore Buttons** -- discard current buttons, restore saved buttons and properties  
Rollout: **Load Some Buttons** -- prompt for file to load, keep/discard current buttons

When BUTTONS is loaded, a single "Create Button" button is placed in the lower left corner of the screen. See RESTORE.BUTTONS below for setting up your buttons programmatically.

## FUNCTIONAL INTERFACE

(CREATE-BUTTON *action label location noopenflg*) [Function]

Creates a button with indicated action and label at the given location and displays it unless *noopenflg* is non-NIL. If *action* is NIL, *label* will be used for *action*. If *label* is NIL, (CAR *action*) will be used for *label* if *action* is a list, *action* otherwise. If both *action* and *label* are NIL, the values of the button properties DEFAULT-ACTION and DEFAULT-LABEL will be used instead. If *location* is not a POSITION or a REGION, the user is prompted for a location.

(BUTTONSPROP *propname {newvalue}*) [Function]

Returns the current value of the button property *propname*. If *newvalue* is given, it becomes the new value. The following properties (and their initial values) are currently in use :

|                |                  |
|----------------|------------------|
| DEFAULT-LABEL  | "Create Button"  |
| DEFAULT-ACTION | (CREATE-BUTTON)  |
| MENU-FONT      | (MODERN 12 BOLD) |
| LABEL-FONT     | (MODERN 10 BOLD) |
| GRID-ORIGIN    | (15 . 15)        |
| SAVE-DIRECTORY | NIL              |
| EDIT-SHADE     | 4104             |
| EXEC-SHADE     | 65535            |

The value of SAVE-DIRECTORY must be acceptable to the function DIRECTORYNAME (i.e., either NIL for login host & directory, or T for current directory, or a standard host & directory spec).

(RESTORE-BUTTONS *filename keep-current-buttons?*) [Function]

Reinstalls the buttons stored on *filename*, which defaults to SAVED-BUTTONS.DATA on the directory indicated by the button property SAVE-DIRECTORY. Existing buttons are discarded unless *keep-current-buttons?* is non-NIL.

(SAVE-BUTTONS *filename buttons*)

[Function]

Saves the given *buttons* in *filename*, which defaults to SAVED-BUTTONS.DATA on the directory indicated by the button property SAVE-DIRECTORY. If *buttons* is NIL, all current buttons will be saved.

(SAVE-SOME-BUTTONS *filename buttons*)

[Function]

Saves the given *buttons* in *filename*. If *buttons* is NIL, you are prompted to indicated the buttons to be saved. If *filename* is NIL, you are prompted to supply a file in which to save the indicated buttons.

(LOAD-SOME-BUTTONS *filename*)

[Function]

Loads the buttons in *filename*. You are prompted to indicate whether to keep the current buttons or discard them. If *filename* is NIL, you are prompted to supply a file from which to load the buttons.

---

---

## CALENDAR

---

---

By: Michel Denber (Denber.WBST @ Xerox.COM)

Uses: TABLEBROWSER, TEDIT

### INTRODUCTION

CALENDAR is a program which can be used to display a calendar on your screen, and keep track and remind you of events and appointments. Calendar 2.04 (the current distributed version) runs in the Koto or Lyric releases of Lisp. The version number appears in the title bar of each Calendar window. Calendar needs the Lisp Library package TABLEBROWSER, which it loads automatically. It also uses TEdit. Various font sizes (from 8 to 36) in the families TimesRoman and Helvetica may be needed, depending on the size chosen for month windows. Reminder files created by earlier versions of Calendar are incompatible with this version.

### I. STARTING CALENDAR

Load CALENDAR.LCOM from your favorite LispUsers directory, eg.

```
LOAD ({ERIS}<LISPUSERS>CALENDAR.LCOM]
```

and then type (CALENDAR). You will get a menu of years (the menu always shows five years starting with last year). If you select a year with Left, it will create a Year window containing a calendar for that year. Each month in the Year window is also a menu item. If you now select a particular month with Left, CALENDAR will create a Month window showing a calendar for that month. You can now select a particular day within the month to bring up a Day browser (described in the next section). The Month window also shows small calendars of last month and next month. You can bring up those months in the current month window by selecting them with Left. If you select them with Middle, the program will create a new window for that month. The Year menu has an entry labelled "Other". If you select this, it will prompt you to type in a year, if you want one that isn't on the menu.

You can have as many year and month windows open at the same time as you like. Month and day windows can also be reshaped to occupy less room on the screen. You can Shrink any of the CALENDAR windows to an appropriate icon, or close them when they are not needed. The reminder facility remains active. If you close your last year window, call (CALENDAR) again to get a new one. CALENDAR uses the Lisp Prompt Window to display informative messages.

Please send your comments, suggestions, and bug reports to me - Denber.WBST (ARPA: Denber.WBST@Xerox.COM). Thanks.

### II. REMINDERS

#### The Day Browser

Clicking Left in any day in a month window will open a browser on that day. The browser displays each reminder for the day, along with its event time if it is a timed reminder. You may have more than one browser open at the same time. When you close a month window, it will automatically close all day browsers for that month. There is a menu across the top of the browser with the following items:

*Add:* Lets you create a new reminder in this day. If you select Add, the program will bring up a TEdit window containing a template for the new reminder. The template contains several fields you can select and fill in. These are described in Creating Reminders, below.

*Display:* Brings up the full contents of the reminder in a TEdit window.

*Delete:* Useful for deleting reminders that you no longer need. By default, timed reminders are deleted automatically after they "fire"; untimed reminders do not fire and are never deleted automatically. Calendar will immediately remove reminders which you delete from the month window (and the reminder's line in the day browser is crossed out), however it will leave reminders that have fired visible in the month window until you redisplay it (eg. September is visible, you select Redisplay from the right-button menu in the title bar or select September again in the Year window, and all fired September reminders will be purged from the month window when it redraws).

*Update:* Saves your reminders to disk (see the section on Saving reminders below).

*Send Mail:* Prompts you for a name to send to. The selected reminder will be mailed to that person when it activates, rather than displaying on your screen. Note that no validity checking is done when you enter a name, so your message could conceivably not be delivered if you typed the name wrong, for example. The message is mailed when the time arrives. Of course, this assumes that your system is running at that time, that you have Lafite active, and that Lafite is running in the mode (GV or NS) corresponding to your intended recipients.

*Period:* Brings up a menu with the choices Daily, Weekly, Monthly. The selected reminder will be made periodic and will appear at the selected intervals.

### **Creating Reminders**

You can create a new reminder either by clicking Add in a day browser, or by clicking the middle button in a day box in the month window. This opens a new reminder form with the following fields:

*Title:* The reminder title should not exceed one line in length. This field will be displayed in the Day browser and the month window. This field may not be omitted; all others are optional.

*Event time:* The scheduled time for the event. By default, this is also the time at which the reminder will be activated. If this field is omitted, the reminder is "untimed". Untimed reminders do not alert you. When a timed reminder activates, it beeps and brings up a TEdit window containing the full reminder text.

*Alert time:* The time at which you would like the reminder to activate. You might want to be reminded of a meeting 10 minutes early, for example. The alert time can be set to any time, before or after the event time, as long as it is in the same day. If this field is omitted, it defaults to the value of the event time.

*Alert:* Edit this field to contain just the word Yes or No. If you choose No, the reminder will not alert you, even if it is a timed reminder. If this field is omitted, it defaults to the value set in the Options menu (see Programming below).

*Duration:* The expected length of the event. Version 2.04 makes no use of this field.

*Message:* The actual message you want to save. This may be any TEdit text or omitted entirely.

The new reminder form includes a menu with the choices Save and Abort. After filling in the fields you want, clicking Save will add the reminder to the system and close the form. Clicking Abort at any point cancels the reminder being created.

The time can be entered in almost any reasonable format, eg. 9:00 AM, 9 AM, 9 a.m., 2:30 PM, 2:30 P.M., 1430, or can be left out by skipping over the field. Times are "AM" by default, so if you only type 8:30, it will assume 8:30 AM. A heuristic is included to ask "Are you sure?" if you type a time earlier than 9 without an AM/PM qualifier (this value is controlled by CALDAYSTART, see Programming, below). Times of noon and midnight are special cases. There is no generally accepted meaning for the expressions "12:00 AM" and "12:00 PM". If you want a reminder at noon, enter the time as "12:00" or just "1200". Because reminders are added to a particular day, midnight is ambiguous; there is no provision for entering a time of midnight.

If you add a reminder for a time that is already in the past (for example, to keep a historical record of an event after the fact), the program will save the reminder but will warn you that the reminder time has already passed.

Expired timed reminders are automatically deleted upon expiration by default. Setting the variable CALKEEPEXPIREDREMS (see Programming, below) will cause timed reminders to be retained after firing.

Reminders which are scheduled for a time when your machine is not running will not be activated the next time you login. This avoids having a possibly long sequence of "dead" reminders popping up at login time.

### **Saving and loading reminders**

You can save your reminders in a file at any point. The first time you start Calendar, it will ask you to provide a default host and directory for reminder files. You should enter this in the usual format, for example {DSK}<Lispfiles> or {ERIS}<your-name>LISP>. This will become the new value of CALDEFAULTHOST&DIR (it is initially NIL). To save your reminders, select Update from any day browser. This will open a pop-up menu of currently loaded files, plus an "other" item for giving a new file name. If you enter a new name, all currently unsaved reminders will be stored under that name. If you select an existing file, the contents of that file will be updated and any new reminders created since the last update will be added to it. If you abandon your sysout or if your machine crashes, you can have Calendar automatically reload your reminders file when you restart (see CALDEFAULTHOST&DIR and CALLOADFILE in Programming, below). You can also load a reminder file at any time by holding the middle button down in the title bar of a month window. This will open a pop-up menu of files that have already been loaded, plus an "other" item to specify a new file. In this version of Calendar there is never any need to load a reminder file more than once. The menu is useful, however, to show which files have already been loaded.

An "almanac" reminder file is distributed along with Calendar. It contains a variety of holidays and notable dates for the year. The file is called CALMANACnn, where nn is the last two digits of the year. For example, the file for 1986 is called CALMANAC86. You can load this file by selecting Other from the middle button menu and typing CALMANAC86.

By default, the program will only save your reminders when you select Update. You may control file updating by changing the Auto File Update option available under the Options menu item in the month window. See Programming, below.

### **III. PROGRAMMING**

A programmatic interface is provided to let you create day, month, or year windows from your own programs.

If your reminder text is a Lisp list (anything inside parentheses), when the reminder fires the program will evaluate the list rather than displaying the reminder in a window and beeping.

**Functions**

(CALENDAR *m d yr*) [Function]

*m*, *d*, and *yr* are integers specifying a month, day, and year, respectively. Arguments are specified as follows:

If only *yr* (must be 4 digits) is supplied, brings up a year window for that year and returns *yr*.

If *m* and *yr* are supplied, brings up a month window for that month and returns *m*.

If *m*, *d*, and *yr* are supplied, brings up a day window for that day and returns *d*.

For invalid combinations (missing *yr*, *d* and *yr* only), returns NIL. Also returns NIL if *yr* is out of range (the calendar algorithm is only valid for years between 1700 and 2100).

**Examples:**

(CALENDAR NIL NIL 1984) shows a calendar for 1984 and returns 1984.

(CALENDAR 10 NIL 1984) shows a calendar for October 1984 and returns 10.

(CALENDAR 10 NIL 84) returns NIL (out of range).

(CALENDAR 10 21 1984) shows October 21st, 1984 and returns 21.

You can also call Calendar with the keywords TODAY, THISMONTH, and THISYEAR.

**Examples:**

(CALENDAR 'THISYEAR) shows a Year window for 1986, if this year is 1986. This might be used in an init file, to always start a Calendar of "this year".

(CALENDAR 'TODAY) opens a Day browser for today, containing all of today's active reminders.

(CALLOADFILE *file-name*) [Function]

Loads the file *file-name* into the reminder system and returns T. Returns NIL if the file is not found or is not a valid reminder file.

**Example:**

(CALLOADFILE '{DSK}<LISPFILES>CALREMINDERS)

**Variables**

CALALERTFLG [Variable]

Initially T. This controls whether or not reminders whose Alert field is not specified should alert you when they fire. T means they will. NIL means they won't.

CALDAYDEFAULTREGION [Variable]

Initially (32 200 350 100). This specifies the default size for day browsers. The location is only used for day browsers opened programatically.

CALDAYSTART [Variable]

Initially 900. This represents the time (in 24 hour format) at which your regular day starts. The system will use it to confirm times you enter without a "PM" indicator if they are less than this value. For example, it is more likely that 4 means 4 PM than 4 AM.

CALDEFAULTALERTDELTA [Variable]

Initially 0. This represents the time (in minutes) before or after the event time you want reminders to be activated, if no explicit alert time was given for them. To be reminded before the event, make this value negative. The resulting time must still be in the same day as the event.

CALDEFAULTHOST&DIR [Variable]

Initially NIL. This is the host and directory on which your reminder files will be saved if you type the file name without a directory specification. The system will prompt you to enter a value for this the first time you start it.

CALFLASHTIMES [Variable]

Initially 0. Specifies the number of times to flash the destination given by CALFLASHTYPE when a reminder is activated.

CALFLASHTYPE [Variable]

Initially 'None. Specifies which window should be flashed when a reminder is activated. Can be set to 'WINDOW, to flash the reminder display window, or 'SCREEN to flash the entire screen. CALFLASHTIMES (above) should be set to the desired number of flashes.

CALFONT [Variable]

Initially 'TimesRoman36. This variable controls the font used to display the Month Window. You can change it for example, by saying (SETQ CALFONT (FONTCREATE 'HELVETICA 18)). The change takes effect the next time you display a month. If you reshape a month window, the program will try to find a smaller font to fit the new window size, but the value of CALFONT will not be changed.

CALHARDCOPYPOMFLG [Variable]

Initially T. This variable controls the printing of the phase-of-the-moon icons when you hardcopy a month window. Setting it to NIL suppresses this printing. Month windows are hardcopied at printer resolution in Koto, screen resolution in Lyric.

CALHILITETODAY [Variable]

Initially 'CIRCLE. This variable determines how today's date will be highlighted in a month window. The default is to draw a circle around it. If you set this to 'BOX, a light gray grid will be placed over the date. Setting this to NIL suppresses all date highlighting.

CALKEEPEXPIREDREMSFLG [Variable]

Initially NIL. If you set this to T, Calendar will not automatically delete reminders when they fire (they can still be deleted using the Delete menu command, above). The default action is to delete reminders when they fire, although they will remain visible until the window is redisplayed.



CALMONTHDEFAULTREGION

[Variable]

Initially (32 32 868 700). This specifies the default position and size for month windows. If you set the size to a value small enough to allow several month windows side by side, the windows will tile left to right, bottom to top.

CALREMDISPLAYREGION

[Variable]

Initially (200 400 300 400). This specifies the default position and size for reminder display windows.

CALTUNE

[Variable]

When a reminder is activated, it will play the tune stored here (in PLAYTUNE format). This is initially a two-note "ding-dong". Set this to NIL if you want no audible warning. 1100's and 1132's have no hardware for sound.

CALUPDATEONSHRINKFLG

[Variable]

Initially 'Never'. This means that Calendar will save your reminders on a file only when you explicitly click Update from a Day Browser. If set to 'Shrink', it will cause Calendar to save your reminder file automatically only when you shrink the Month window. This is useful when you are entering many reminders at the same time, but it means you must remember to explicitly shrink the month window or your reminders will be lost if your machine dies. If set to 'Always', causes Calendar to immediately save each reminder as soon as it is created.

You can also set these variables interactively by clicking on the box marked "Options" in any Month window. This brings up a freemenu similar to the TEdit expanded menu.

```

Calendar Options
Alert: Yes No
Keep expired rems.: Yes No
Auto. file update: Always Shrink Never
Alert delta: 0
Host & dir.: {DSK}<LISPFILES>
Apply!

```

*Alert:* Specifies the default for the Alert field in the new reminder form. Sets the value of CALALERTFLG (described above).

*Keep expired rems.:* If set to No, the system will automatically delete reminders when they fire (although they remain listed in the month window until the next time you redisplay it). Sets the value of CALKEEPEXPIREDREMSFLG.

*Auto. file update:* Always means that the system will update the reminder file every time you create a new reminder. Shrink means update only when a month window is shrunk. Never means updates will be done only when you explicitly select Update from a Day browser. Sets the value of CALUPDATEONSHRINKFLG.

*Alert delta:* Sets the value of CALDEFAULTALERTDELTA.

*Host & dir.:* Sets the value of CALDEFAULTHOST&DIR .

After you have made the selections you want, click Apply! This sets the selections and closes the menu. If you don't want to make any changes, just close the menu (like closing any window). This

preserves the previous settings even if you changed them in the menu. Any changes you make to these variables are not saved automatically in reminder files.

#### **IV. LIMITATIONS**

Day groups must begin and end in the same month.

The calendar algorithm is valid only for years between 1700 and 2100.

#### **V. KNOWN BUGS**

Today-circling function occasionally fails to erase the old day.

#### **VI. FUTURE PLANS**

Automatic scheduling.

Automatic communication with other Calendars.

---

---

**CANVASCONVERTER**

---

---

By: Stephen Knowles (Stephen Knowles:49/89/636/13:Siemens AG)

Partly based on work by:

Matthias Schneider-Hufschmidt (Matthias Schneider-Hufschmidt:ZTISOF:SIEMENS)

Giselbert Schramm (Giselbert Schramm:ZTISOF:SIEMENS)

Uses: BITMAPFNS

This document last edited on 19-Sep-1988 13:32:21.

## INTRODUCTION

This module enables the transfer of bitmaps between the Envos Lisp and Xerox ViewPoint environments. The medium used for the transfer is an NS file server (i.e. a file drawer which can be accessed by both environments). The possibility of transferring Lisp bitmaps into the ViewPoint environment is particularly useful for documenting Lisp applications.

## MODULE EXPLANATIONS

There are essentially two major functions:

(IL:WRITECANVAS *BITMAP FILE*) [Function]

This function writes the *BITMAP* on to *FILE* and makes *FILE* of type ViewPoint Canvas, whereby *FILE* must be on an NS file server.

(IL:FETCHCANVAS *FILE*) [Function]

This function reads *FILE* into a Lisp bitmap, whereby *FILE* must be on an NS file server.

Additionally there are two auxiliary functions to aid in the use of the above two functions.

(IL:SNAPBM) [Function]

and

(IL:CANVAS-FROM-WINDOW *WINDOW FILE*) [Function]

## EXAMPLES

All examples must be typed into an INTERLISP exec.

To write a canvas of a Lisp screen region:

```
(WRITECANVAS (SNAPBM)
```

```
' {NSFileServer:Domain:Organization}<FileDrawer>Folder>TESTFILE)
```

To write a canvas of a Lisp window:

```
(CANVAS-FROM-WINDOW (WHICHW)
```

```
' {NSFileServer:Domain:Organization}<FileDrawer>Folder>TESTFILE)
```

To read a canvas into a Lisp bitmap:

```
(SETQ X (OPENSTREAM
```

```
' {NSFileServer:Domain:Organization}<FileDrawer>Folder>TESTCANVAS 'INPUT))
```

```
(EDITBM (SETQ LISPBITMAP (FETCHCANVAS X)))
```

```
(CLOSEF X)
```

### CAVEAT

When fetching a canvas, there is a 50-50 chance that the Lisp bitmap will be O.K. It could, however, come out distorted (this is due to the differing ways in which ViewPoint and Lisp handle bitmaps, Lisp uses 16 complement, ViewPoint 32 complement - or something like that). If this should be the case, simply increase the canvas width in ViewPoint by 5 millimeters (approx. 16 pixels) and repeat the fetching process.

Unfortunately in the Lyric version if one repeatedly wrote a canvas with the same name, the file server somehow got mixed up and set the file-info of the folder above the canvas into "type = canvas"! One could put this right with the (SETFILEINFO...) function in Lisp, although under normal circumstances one does not write out a canvas repeatedly with the same name any way. I have been unable to test the behaviour in MEDLEY.

Compatibility has only been tested up to ViewPoint 1.1.

---



---

## CD

---



---

By: Henry Thompson (HThompson.pa@Xerox.com)

This document last edited July 6, 1988.

### INTRODUCTION

The file CD implements a UNIX\*-style facility for manipulating the connected directory. It also insures that the connected directory is always displayed.

CD *PATTERN*

[Exec command]

### MODULE EXPLANATIONS

CD is defined as a command which allows low-overhead means of effecting many common changes of connected directory. Its behaviour is partly conditioned by three global variables:

CD.DEFAULT.HOST [Variable]

CD.DEFAULT.PREFIX [Variable]

CD.DEFAULT.USER [Variable]

CD.DEFAULT.HOST defaults to DSK. CD.DEFAULT.PREFIX defaults to the name (e.g. DSK) of the local disk volume on a Dandelion, otherwise NIL. CD.DEFAULT.USER defaults to the value of USERNAME, and is updated automatically after GREETing.

The value of CD is always a CONS-pair of the old and new connected directories.

On hosts which support some form of sub-directory, CD needs to know the character which is used to separate sub-directories. The table CD.OS.SEPRS is an a-list which determines this mapping - it is initialised to map UNIX\* and VMS to "/" and DSK, NS and IFS to ">". To enter this table it looks up the host first in CD.OS.SEPRS directly, then via NETWORKOSTYPES. In the documentation which follows, ">" means whatever the separator is for the relevant host.

The possibilities for *pattern* are as follows:

*empty*

Connects to the directory determined by the conjunction of CD.DEFAULT.HOST, CD.DEFAULT.PREFIX and CD.DEFAULT.USER.

*{anything}*

Interprets *pattern* as a complete directory specification, and connects to it.

*<anything*

Interprets *pattern* as a directory specification to be qualified by CD.DEFAULT.HOST and CD.DEFAULT.PREFIX, and connects to it. For example if CD.DEFAULT.HOST is {server} and CD.DEFAULT.PREFIX is NIL, then CD <dir>sdir> is equivalent to CD {server}<dir>sdir>, whereas if CD.DEFAULT.PREFIX was /user and server was known to be running UNIX\*, then CD <dir/sdir> would be equivalent to CD {server}</user/dir/sdir>.

*.>rest*

Equivalent to CD rest. This is purely for compatability with UNIX\*.

*..>rest*

Equivalent to peeling off one (sub-)directory from the currently connected directory, followed by CD rest. For example, if connected to {server}<dir>sdir>, then CD ..>sdir1 is equivalent to CD {server}<dir>sdir1>. Note that because of common lisp reader peculiarities, you cannot use .. alone under a common lisp read-table. The synonym << can be used instead.

*otherwise*

Treat *pattern* as a further specialisation of the current directory, and connect to the resulting sub-directory. For example, if connected to {server}<dir>sdir>, then CD sdir is equivalent to CD{server}<dir>sdir>sdir>.

Note that throughout, the closing ">" is optional.

### **Menu Interface**

At any time you can left button in the window displaying the current connected directory, and see a menu of all the directories you have yet been connected to. Selecting one will move you there. You can also shift-select out of this menu into the current input stream. This latter is very useful when typing file names.

Middle buttonning in the directory display window will give you a menu of directories, followed by a menu of Connect/Browse/Delete. Connect does so, Browse brings up a file browser and Delete removes the directory from subsequent menus.

---

\*UNIX is a trademark of Bell Laboratories.

---

---

## CHATEMACS

---

---

By: Randy Gobbel (Gobbel.pa)

requires:CHAT, chatemacs.elc (GnuEmacs Lisp program)

This document last edited on August 24, 1987

### INTRODUCTION

ChatEmacs, in conjunction with the **chatemacs.elc** module for GnuEmacs, enables use of the mouse for scrolling and selection in GnuEmacs. It also allows use of the META key for escape-prefix commands and automatically switches Chat in and out of Emacs mode when entering and leaving the editor.

### DETAILS

After loading ChatEmacs, typing META-char will send an ESCAPE character, followed by the vanilla character. CTRL-META-char sends an ESCAPE followed by CTRL-char. Once ChatEmacs is active, most Emacs commands should require only one keystroke. Since Emacs was originally designed for terminals with a META shift key, this makes the Emacs command set somewhat more regular and easier to remember. For example, scrolling forward and backward will be on CTRL-V and META-V, respectively.

In order to enable mouse actions, first load CHATEMACS.LCOM into Interlisp. After opening a Chat connection and running GnuEmacs, either load chatemacs.elc manually (by giving the ^Xload command), or add the following line to your GnuEmacs init file (.emacs):

```
(load "chatemacs")
```

After loading chatemacs.elc, the title bar on your Chat window should say "Emacs ON". If not, middle-buttoning the "Emacs" menu item in your Chat window will enable mouse events to be sent to Emacs. After ChatEmacs has been activated for the first time, the Chat window's title bar will always indicate whether Emacs mode is on or off. If your mouse clicks don't seem to be taking effect, check the title bar first!

Automatic switching frees the user from having to manually turn ChatEmacs on and off when using Emacs. In most circumstances (see exceptions below) automatic switching will not interfere with other Chat operations, and can be left enabled. Auto-switching is controlled by:

CHATEMACS.SWITCH.ENABLED

[Variable]

When this variable is non-NIL, Chat will respond to a sequence of two consecutive ESCAPEs by toggling the flag that controls mouse event sending. The state of the flag is noted in the window's title bar, just as if the menu command had been executed. CHATEMACS.SWITCH.ENABLED is defaulted to NIL.

auto-switch-enabled

[GnuEmacs Variable]

This variable controls auto-switching on the GnuEmacs side of the Chat connection. If it is non-nil, GnuEmacs will send a switch command when chatemacs.elc is loaded, and another when exited via a ^X-^C command.

### Using Emacs with the mouse

The chatemacs.elc module, at the GnuEmacs end of the connection, determines the interpretation of mouse clicks. The current user interface is more complicated than I would like, and suggestions for improvements are welcome.

The most basic operations are fairly simple: left button in a text buffer moves the Emacs "point" to wherever the cursor is pointing. Right button moves the mark (the typein cursor will move for a couple of seconds just to show you where you've just put the mark), and copies the new region to the kill buffer (for use with "shift-select," see below).

Scrolling with the mouse works more or less as in Interlisp, with the scrollbar being the right-hand part of the screen past column 80. Alternatively, holding down the META key makes the entire text area act as a "scrollbar". As in most Envos environments, left button scrolls the line that the mouse is pointing to to the top of the window, right button moves the top line down to the mouse cursor, and middle button "thumbs", taking the vertical displacement of the mouse cursor as an offset into the file (i.e., top line = beginning of file, bottom line = end of file).

Shift- and control- mouse clicks perform editing operations: shift-left copies the contents of the kill buffer to wherever the mouse is pointing (the closest thing to Interlisp shift-select I could come up with). Control-left and control-right kill from point to where the mouse is pointing (sort of like control-select). Control-shift-left moves the mark without copying anything to the kill buffer.

Mouse clicks in the mode line and minibuffer do things that were inherited from il-mouse's ancestor, a package for the BBN Bitgraph terminal. Maybe you will find them useful. They are: The modeline acts like a sideways scrollbar, left=top. In the minibuffer, left button is equivalent to typing META-X, middle button evals an expression you type in, and (beware!) right button suspends Emacs (equivalent to typing ^X^Z).

As mentioned above, the current user interface is sort of, how shall I say, "gnarly." If you have better ideas, please let me (Gobbel.pa) know.



---

---

## CHATSERVER

---

---

By: Larry Masinter (Masinter.PA@Xerox.COM)

This document last edited on September 7, 1988.

### REQUIREMENTS

CHATSERVER-NS Requires: CHATSERVER and COURIERSERVE.

CHATSERVER-RS232 requires: CHATSERVER and (DLTTY or DLRS232C). As of this date, CHATSERVER-RS232 hadn't been tested with Medley.

CHATSERVER-TCP requires: CHATSERVER and TCP. As of this date, CHATSERVER-TCP is unreliable: the chat server sometimes leaves open the connection and will not open another one.

In general, a protocol chatserver requires CHATSERVER and a protocol converter. Sources for TCP server available.

CHATSERVER also loads LispUsers modules CL-TTYEDIT and SIMPLECHAT.

The module PREEMPTIVE is useful in conjunction with CHATSERVER but not required.

### INTRODUCTION

CHATSERVER is a general facility that allows a Lisp workstation to be controlled from a dumb terminal. In addition to CHATSERVER, you will need a protocol driver: something that connects the CHATSERVER to a communication protocol. The various protocol drivers are the mechanism by which CHATSERVER can be controlled; versions include using XNS via CHATSERVER-NS, TCP/IP TELNET protocol via CHATSERVER-TCP, and RS232 via CHATSERVER-RS232. CHATSERVER-NS is the most reliable, although CHATSERVER-RS232 has worked reliably in the Lyric release.

The server implements password protection using the same mechanism as IDLE. There is another variable, CHATSERVER.PROFILE, which gets searched first for ALLOWED.LOGINS so that you can have a different setting.

IL:CHATSERVER.PROFILE [Variable]

The value of the variable CHATSERVER.PROFILE appended to the front of IDLE.PROFILE when determining login options etc for the chatserver. The property IDLE.ONLY is also consulted; if T, chatserver only allows connections when machine is in idle mode.

### Example:

```
(SETQ CHATSERVER.PROFILE ' (ALLOWED.LOGIN (T) IDLE.ONLY T))
```

means to allow only the previously logged in user, and then, only when in IDLE mode.

QUIT [Exec Command]

The QUIT exec command exits a chatserver session. It signals an error if you are not running in a chatserver session.

**Documentation for CHATSERVER-NS:**

CHATSERVER-NS implements the Xerox Network Systems GAPTELNET protocol. It allows connection to a machine running Medley from other machines that implement this protocol, including Viewpoint (using ViewpointChat), External Communication Service servers (which allow dial-in from remote terminals into an XNS network), XDE workstations and other Interlisp-D implementations (Koto, Lyric, Medley.)

Gaptelnet is a courier server program, and so requires the COURIERSERVE lispusers module (which it loads automatically). The following function is part of COURIERSERVE

(COURIER.START.SERVER) [Function]

This "starts" the courier server process which listens for connection attempts. It is necessary to call this (once) before you can CHAT to your Lisp workstation. (If the process dies for some reason, you will not be able to CHAT until you restart the process.)

**Documentation for CHATSERVER-TCP:**

This module was an attempt to implement a TCP/TELNET server. Unfortunately, the mechanism by which it waits for a connection is buggy, and it does not negotiate terminal characteristics properly with the client calling workstation. It is therefore unreliable & may need to be restarted. Using telnet from a Sun to Lisp I've found it was necessary to explicitly tell the Sun not to echo, to send character at a time, etc.

(TCPCHATSERVER) [Function]

It is necessary to call this function to spawn the process that waits for TCP connections.

**Documentation for CHATSERVER-RS232:**

The CHATSERVER-RS232 module attempts to allow for connections on the RS232 or TTY port on an 1186 or 1108. It uses DLRS232.

(RS232CHATSERVER) [Function]

Spawns a process waiting for a character to be typed on the RS232 port, and then starts a chatserver session.

**Other notes:**

The server runs a standard (XCL) exec. Note that you can't do graphics; the debugger will not attempt to open a window, only the type-in commands are available, ED will give you the "teletype" editor. Interrupt characters enabled are ^E, ^D, DEL, ^B, ^H and ^T. (Note that currently interrupts are only processed when they are read, and there is no way to interrupt a run-away process.)

Typeout uses a "---more---" style: after (PAGEHEIGHT) lines, the system will prompt you with a "---more---". Type any character, and the more will be erased.

Chatserver assumes you are chatting from a DM2500 emulator, and treats font changes as a switch between bold and regular as appropriate.

The PREEMPTIVE Lispusers module is useful when running chatserver, because it will keep the running process from blocking out the typein process. For some protocol drivers (and the NS server in particular), this is necessary to avoid timeouts.

CHATSERVER advises various facilities in the environment that normally create menus to check to see if the "controlling" keyboard is not the workstation console; these facilities include TTYIN, the editor, the debugger, CHAT. Thus, calls to the editor use the teletype-style editor from Interlisp, while FIX does not generally allow character editing.

---

---

## CHECKSET

---

---

By: >>Your Name<< (>>Your net address<<)

>>Other packages necessary to run this one<<

This document last edited on >>DATE<<

This package checks source files against compiled files in a directory and prompts you about whether you want to (RE)COMPILE the files that need it. It compares the FILECREATED expressions, and determines whether a BRECOMPILE with CHANGES will suffice or if it is necessary to BCOMPL the file.

(CHECKSET FILES COMPFLG)

[Function]

FILES is a list of files. If FILES is NIL, CHECKSET is driven by the variable FILESETS.

COMPFLG can be:

N                    don't compile, just return list. List can be passed to COMPFILES.

Y or NIL            compile

ASK                ask, for each file, whether to compile it.

FILESETS

[Variable]

Used by (CHECKSET NIL): FILESETS is a list of variables, each of which has a value that is a list of files. (CHECKSET NIL) performs (for X in FILESETS join (CHECKSET (EVALV X))). For example,

if FILESETS = (0LISPSET 1LISPSET) and 0LISPSET = (ATERM LLREAD BREAK), 1LISPSET = (WINDOW EDIT) then (CHECKSET) will check if each of those files in turn need recompiling.

(COMPFILES LST)

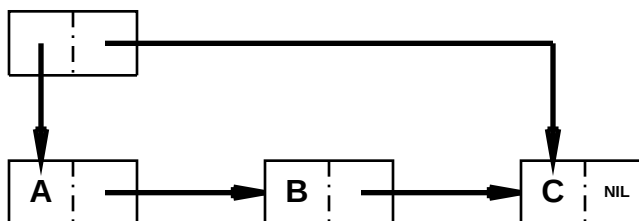
[Function]

takes a list of elements of the form (RECOMPILE FILE) (COMPILE FILE) as returned by CHECKSET with COMPFLG=N and performs the corresponding operation.

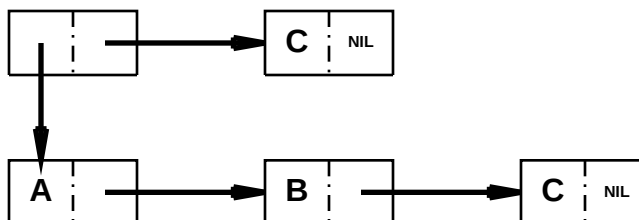
## CIRCLPRINT

HPRINT is designed primarily for dumping circular or reentrant list structures (as well as other data structures for which READ is not an inverse of PRINT) so that they can be read back in by Interlisp. The CIRCLPRINT package is designed for printing circular or reentrant structures so that the user can look at them and understand them.

A reentrant list structure is one that contains more than one occurrence of the same EQ structure. For example, TCONC makes use of reentrant list structure so that it does not have to search for the end of the list each time it is called. Thus, if X is a list of three elements, (A B C), being constructed by TCONC, the reentrant list structure used by TCONC for this purpose is:

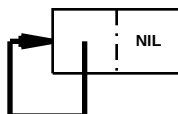


This structure would be printed by PRINT as ((A B C) C). Note that PRINT would produce the same output for the nonreentrant structure:

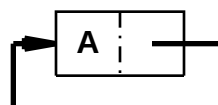


In other words, PRINT does not indicate the fact that portions of the structure in the first figure are identical. Similarly, if PRINT is applied to a circular list structure (a special type of reentrant structure) it will never terminate.

For example, if PRINT is called on the structure:



it will print an endless sequence of left parentheses, and if applied to:



will print a left parenthesis followed by an endless sequence of *A*'s.

The function `CIRCLPRINT` described below produces output that will exactly describe the structure of any circular or reentrant list structure. This output may be in either single- or double-line format. Below are a few examples of the expressions that `CIRCLPRINT` would produce to describe the structures discussed above.

First figure, single-line:

```
((A B *1* C)1)
```

First figure, double-line:

```
((A B C) 1)
1
```

Third figure, single-line:

```
(*1* 1)
```

Third figure, double-line:

```
(1)
1
```

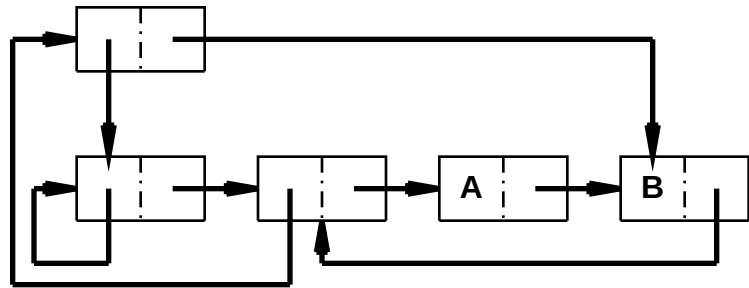
Fourth figure, single-line:

```
(*1* A . 1)
```

Fourth figure, double-line:

```
(A . 1)
1
```

The more complex structure:



is printed as follows:

Single-line:

```
(*2* (*1* 1 *3* 2 A *4* B . 3) . 4)
```

Double-line:

```
((1 2 A B . 3) . 4)
21 3 4
```

In both formats, the reentrant nodes in the list structure are labeled by numbers. (A reentrant node is one that has two or more pointers coming into it.) In the single-line format, the label is printed between asterisks at the beginning of the node (list or tail) that it identifies. In the double-line format, the label is printed below the beginning of the node it identifies. An occurrence of a reentrant node that has already been identified is indicated by printing its label in brackets.

```
(CIRCLPRINT LIST PRINTFLG RLKNT)
```

[Function]

Prints an expression describing *LIST*. If *PRINTFLG*=NIL, double-line format is used, otherwise single-line format. CIRCLPRINT first calls CIRCLMARK, and then calls either RLPRIN1 (if *PRINTFLG*=T) or RLPRIN2 (if *PRINTFLG*=NIL). Finally, RLRESTORE is called to restore *LIST* to its unmarked state. Returns *LIST*.

(CIRCLMARK *LIST* *RLKNT*) [Function]

Marks each reentrant node in *LIST* with a unique number, starting at *RLKNT* plus one (or one, if *RLKNT* is NIL). Value is *RLKNT*. Marking *LIST* physically alters it. However, the marking is performed undoably. In addition, *LIST* can always be restored by specifically calling RLRESTORE.

(RLPRIN1 *LIST*) [Function]

Prints an expression describing *LIST* in the single-line format. Does not restore *LIST* to its unCIRCLMARKed state. *LIST* must previously have been CIRCLMARKed, or an error is generated.

(RLPRIN2 *LIST*) [Function]

Same as RLPRIN1, except that the expression describing *LIST* is printed in the double-line format.

(RLRESTORE *LIST*) [Function]

Physically restores list to its original, unmarked state.

Note that the user can mark and print several structures that together share common substructures, e.g., several property lists, by making several calls to CIRCLMARK, followed by calls to RLPRIN1 or RLPRIN2, and finally to RLRESTORE.

(CIRCLMAKER *LIST*) [Function]

*LIST* may contain labels and references following the convention used by CIRCLPRINT for printing reentrant structures in single-line format, e.g., (\*1\* . 1). CIRCLMAKER performs the necessary RPLACAs and RPLACDs to make *LIST* correspond to the indicated structure. Value is (altered) *LIST*.

(CIRCLMAKER1 *LIST*) [Function]

Does the work for CIRCLMAKER. Uses free variables LABELST and REFLST. LABELST is a list of dotted pairs of labels and corresponding nodes. REFLST is a list of nodes containing references to labels not yet seen. CIRCLMAKER operates by initializing LABELST and REFLST to NIL, and then calling CIRCLMAKER1. It generates an error if REFLST is not NIL when CIRCLMAKER1 returns. The user can call CIRCLMAKER1 directly to "connect up" several structures that share common substructures, e.g., several property lists.

---

---

**CL-TTYEDIT**

---

---

By: Larry Masinter (Masinter.PA@Xerox.COM)

This document last edited on November 24, 1987.

**INTRODUCTION**

This file patches the TTY editor so that it is a little more usable in Lyric/Medley for non-Interlisp sources. In particular, it changes the TTY editor so that EDITRDTBL is no longer used; the read table in effect at the time the ttyeditor is invoked is used instead (\*READTABLE\*).

It patches the main editor loop (EDITCOM) so that the package and case of edit commands are ignored, i.e., if you type in the P command, it doesn't care whether it is XCL-USER::P or IL:P or |p|.

It patches EDITFPAT (which takes "find" patterns) so that you can specify patterns with --, &, ==, \*ANY\* in any package, and use --- instead of .. (since symbols consisting entirely of dots are not allowed in CL readtables.)

This file is especially useful if you are talking to another machine using CHATSERVER and need to edit something on the remote machine; since the CHAT connection is character only, you can't run (and the system doesn't attempt to run) SEDIT.



---

---

## COLORDEMO

---

---

Maintained By: Frank Shih (Shih.envos@Xerox.com)

Uses: Color, Peano, ColorPolygons

This document last edited on 8-Nov-88

### Color Demonstration Programs

The following functions are on file on COLORDEMO.LCOM.

(COLORDEMO) [Function]

brings up a menu of color demonstration programs. The system cycles through the entries on the menu automatically, allowing each to run for a small, fixed amount of time (typically 40 seconds). Selecting one of the entries in the menu causes it to start that program.

(COLORKINETIC *Wait Window*) [Function]

runs a color version of the standard *Kinetic* demo in *Window*. The demo works by BLTSHADEing random textures to *Window* to random regions of *Window* using random OPERATIONS with a bias towards OPERATION='REPLACE.

(VINEDEMO *Wait Window*) [Function]

draws a twisting vine that changes in thickness, direction, and color as it grows inside *Window*.

(RAINING *Wait Window*) [Function]

drops of rain appear to splash on to *Window* causing concentric circular ripples of color to spread outward on the surface of *Window*.

(MODARTDEMO *Wait Window*) [Function]

some of the art produced by this demo is at least as good as some that you will see in art galleries. The demo actually works by BITBLTing *Window* on to itself with a displacement with random SOURCETYPE and OPERATION, mixed in with some BITBLTed random textures.

(STARBURSTDEMO *Wait Window*) [Function]

far far away in a galaxy somewhere in the future, an unexplained physical force sweeps over peaceful stars turning them at once into brilliant exploding novas which are safely viewed at a distance through the rear view porthole of our fleeing spaceship.

(COLORPEANODEMO *Wait Window*) [Function]

the Peano fractal curve in color.

(BUBBLEDEMO *Wait Window*)

[Function]

the *Window* fills with brilliantly colored soap bubbles. This demo works by calling FILLCIRCLE.

(OVERPAINTDEMO *Wait*)

[Function]

uses masking techniques to print over the lower right color demo window. Notice that just the pixels of the character images get printed and not the white pixels that normally surround the character images.

(TILEDEMO *Wait*)

[Function]

takes what currently appears in the four color demo windows and adds their images to a growing list called TILEBITMAPS. The demo then tiles the color screen background followed by repeatedly tiling the four color demo windows with randomly chosen tiles.

(TUNNEL *Speed*)

[Function]

draws a series of concentric rectangles of increasing size in increasing color numbers. *Speed* determines the size of the rectangles. This can then be “run” by calling ROTATEIT, which is described below.

(MINESHAFT *N OutFlg*)

[Function]

draws a series of concentric rectangles of size *N* in increasing color numbers. *OutFlg* determines whether the color numbers increase or decrease. This can then be “run” by calling ROTATEIT, which is described below.

(WELLEDemo *Wait*)

[Function]

draws a series of concentric circles on the color demo windows in increasing color numbers. The circles are then “run” by rotating the color map.

(ROTATEIT *BeginColor EndColor Wait*)

[Function]

goes into an infinite loop rotating the screen color map. The colors between *BeginColor* (default zero) and *EndColor* (default maximum color) are rotated. If *Wait* is given, (DISMISS *Wait*) is called each time the color map is changed. This provides an easy way of “animating” screen images.

(COLORPOLYDEMO *ColorStream*)

[Function]

is on the file COLORPOLYGONS.DCOM. It runs a version of the *Polygons* program on the color screen.

---

---

## COLORNCC

---

---

By: >>Your Name<< (>>Your net address<<)

>>Other packages necessary to run this one<<

This document last edited on >>DATE<<

### INTRODUCTION

This package is the Xerox Lisp software driver for Number Nine Computer Corporation's Revolution 512 x 8 color card. The IJCAI 1985 show featured an Interlisp-D color window system running on a Xerox 1108 attached to a Busmaster and color display. That color window system was based partly on a version of this package. This advanced high level software has become available with the Lyric release or Xerox Lisp.

### NECESSARY HARDWARE

You need a Xerox 1108 with Extended Processor Option (CPE), Xerox Busmaster card, an IBM PC expansion chassis, a Number Nine Computer Corporation Revolution 512 x 8 color card, and a third party color display. Please contact your Xerox representative for details concerning acquiring and setting up all the required hardware.

Assuming you have all the hardware you need, turn it all on. This means

- (1) Your 1108 is running Xerox LISP.
- (2) Your PC expansion chassis is plugged in and powered on.
- (3) A cable connects between your 1108 CPE board and your Busmaster board. (The Busmaster board does not go into the 1108, but should rest outside the 1108.)
- (4) Another cable connects between your PC expansion chassis and your Busmaster board.
- (5) A pair of purple and orange wires connects your Busmaster board to the +5V/Gnd power supply terminals on the side of your 1108.
- (6) Your Number Nine Revolution 512 x 8 board is plugged into the PC expansion chassis.
- (7) Your color display is plugged in and powered on.
- (8) Three cables for red, green, and blue signal connect your Number Nine card to your color display.

Any reconnections that involve (3), (4), or (5) should be made while your 1108 is off. Until you issue some software commands, a black display is normal.

Check that your hardware is set up correctly by typing

```
(\COLORNNCC.STARTBOARD)
```

[Function]

Taking less than a second to execute, there should be a noticeable flicker on your color display, followed by what can be taken to be a stable abstract pattern representing the contents of the Number Nine card's RAM when the PC chassis was turned on. If \COLORNNCC.STARTBOARD doesn't return and simply waits, there could be something wrong with the BusMaster card. If \COLORNNCC.STARTBOARD does return, but the image is noisy instead of stable, it could be that you have to adjust the frequency selector on your color display.

## **COLORNNCC SOFTWARE**

The COLORNNCC package provides the machine dependent portion of software that is needed to drive your color display assuming you are using an 1108 with COLORNNCC card and Busmaster. Other than LOADING the COLORNNCC package and turning the COLORNNCC package on using the function COLORDISPLAY, all additional functionality is provided by and documented with the COLOR package. There are no COLORNNCC functions that the user needs to call directly. The user calls functions described in the COLOR documentation.

Once your hardware is on, you can proceed to issue COLOR commands to your hardware. You should have the COLORNNCC package already LOADED from your LIBRARY directory. That is, you've already done something like (LOAD '<LIBRARY>COLORNNCC.DCOM). At this point it may be convenient to follow this documentation along with the documentation for COLOR in the Lisp Library Packages Manual. If you now type

```
(COLORDISPLAY 'ON 'REV512X8)
```

your display will now change from total black to a color test pattern with horizontal and vertical stripes. The sequence of events is that there should be a noticeable flicker on your color display, followed by what can be taken to be an abstract pattern representing the contents of the Number Nine card's RAM when the PC chassis was turned on, followed by a white wall covering up this abstract pattern, followed by the painting of this white wall with horizontal and vertical stripes of color woven together. There are now some simple tests you can do to satisfy yourself that your hardware is working. Here is a small list of things to try:

```
(SETQ CSBM (COLORSCREENBITMAP))
(BLTSHADE 'WHITE CSBM)
(BLTSHADE 'RED CSBM)
(BLTSHADE 'GREEN CSBM)
(BLTSHADE 'BLUE CSBM)
(SETQ DS (DSPCREATE CSBM))
(DRAWLINE 0 0 500 500 10 'REPLACE DS 'YELLOW)
(DRAWLINE 500 0 0 500 10 'REPLACE DS 'CYAN)
```

Assuming all has gone well to this point, you should now be able to try all the functions described in the COLOR package documentation. The COLORDEMO package is a good source of test programs to try — (IL:LOAD 'COLORDEMO.LCOM) to get this package. Both COLOR and COLORDEMO documentation are in your LispUsers' Manual.

---

**COMMWINDOW**

---

By: Larry Masinter, Stan Lanning, Nick Briggs, Richard Burton (Masinter.pa@Xerox.com, Lanning.PA@Xerox.COM, Briggs.PA@Xerox.com, Burton.pa@Xerox.com)

Uses: COURIERSERVE

This document last edited on 2-Apr-87 17:49:35

## INTRODUCTION

This is a demonstration of communication capabilities. The COMMWINDOW module implements a "remote window" capability, where one user can watch (with slow update) a region of another users screen.

Both participants need to have COMMWINDOW loaded.

To show someone else a piece of your screen, call

(SEND-BITS *partner* &OPTIONAL *frame*)

[Function]

The *partner* is the name of the machine you want to talk to. (The watcher has to be registered in the clearinghouse database; this is a NS protocol name.)

*frame*, if supplied, is a screen region to show. If it is omitted, send-bits will prompt the (sender) for a frame of the screen.

The sender has complete control over the frame. The frame appears as a gray frame around the area shown, like this:



(The frame consists really of 4 thin windows, so that you can really type/button inside it.)

The frame can be moved by left buttoning anywhere on it. Right buttoning on it makes it go away temporarily.

(Reshaping the frame can be accomplished by left buttoning the frame while the shift key is depressed. This doesn't reshape the remote window currently, so this isn't very useful.)

Since this is an experiment, there are a couple of parameters you can vary. The function (mode-menu) will bring up a window that lets you control any and all of these parameters.

The first is the shape of the areas sent in each packet.

**Currently available shapes are:**

square: a square of bits as big as will fit in a single packet  
rectangle: 4 times wider than it is high.  
horizontal: as wide as the frame, and then as high can be  
vertical: as high as the frame and as wide as can be  
h3: this is a funny mode, where the update is in horizontal chunks, but spread out.

**PARAMETERS**

The parameter `\ETHERLIGHTNING` lets you tell the low-level ethernet code to randomly drop packets. This parameter can be used to study the effects of noisy communication lines on transmission protocol.

Another parameter gives you some local control over the region within the frame that is updated. There are three available values:

*Sender:* Update near the sender's cursor, when it has moved, ASAP.  
*Viewer:* Update near the viewer's cursor, when it has moved, ASAP.  
*NIL:* don't do anything special.

Another parameter lets you prune out sending packets that update regions of the display that have not changed. The tradeoff here between the cost of sending the packets vs testing a region to see if it has changed.

A recent addition was the following part of the protocol: when the mouse on the sender is moving, it will send a square around the mouse interleaved with any other transmission ongoing. That means you can make sure a remote area is being updated by wiggling the mouse.

The sender's mouse is also tracked in the remote viewers viewpoint (the cursor shape is currently not tracked, however). The mouse coordinates are sent every packet, so that mouse position always updates.

The viewer has a limited option to place a pointer back on the sender's screen: if the viewer holds the

shift key down while the viewer's mouse is in the view point window, a little pointer () will appear in the senders window and track the viewers cursor. (The pointer is in fact a little icon window.) When the viewer releases the shift key, the pointer icon disappears.

**CAUTIONS:**

Don't move the frame off the screen. It will just signal an error.

The "don't change unsent tiles" parameter doesn't seem to work.

---



---

**COMPAREDIRECTORIES**


---



---

By: Larry Masinter and Ron Kaplan

This document edited on

December 2, 1987

December 28, 1998 (Ron Kaplan)

April 7, 2018 (Ron Kaplan)

Rewritten August 25, 2020 (Ron Kaplan)

COMPAREDIRECTORIES compares the contents of two directories, identifying files according to their creation dates and lengths. It is called using the function

(COMPAREDIRECTORIES *DIR1 DIR2 SELECT FILEPATTERNS EXTENSIONSTOAVOID  
USEDIRECTORYDATES OUTPUTFILE ALLVERSIONS*) [Function]

Compares the creation dates of files with matching names in the lists that CDFILES returns for DIR1 and DIR2. Collects or prints CDENTRIES for those files that meet the SELECT criteria. May also collect or print entries for relevant files that exist in DIR1 or DIR2 but not both.

SELECT specifies which the match/mismatch criteria for filtering the output. If SELECT is or contains

AFTER or >: select entries where file1 has a later date than file2

BEFORE or <: select entries where file1 has an earlier date than file2

SAMEDATE or =: select entries where file1 and file2 have the same date

-\*: exclude entries where file1 does not exist

\*-: exclude entries where file2 does not exist

~=: exclude entries where file1 and file2 are byte-equivalent

SELECT = NIL is equivalent to (< > -\* \*-). Excludes files with matching dates, a useful default for identifying files that may require further attention.

SELECT = T is equivalent to (= < > -\* \*-). Includes all files for processing by other functions or later filtering by CDSUBSET (below).

SELECT may also contain the token AUTHOR to indicate that authors should be provided in the printed output (see CDPRINT below).

Unless USEDIRECTORYDATES, the FILECREATED date is used for the date comparison of Lisp source and compiled files, otherwise the file-system CREATIONDATE is used.

If OUTPUTFILE=NIL, then a list of the form (Parameters . entries) is returned. Parameters is a list (DIR1 DIR2 SELECT DATE) that records the parameters of the comparison. Entries contains one entry for each of the file-comparisons that meets the SELECT criteria. Each entry is a CDENTRY record with fields

(matchname info1 daterel info2 equiv)



where matchname is the name.extension shared by the two files, and each file info is either NIL (for nonexistent files) or a CDINFO record with fields

(FULLNAME DATE LENGTH AUTHOR TYPE EOL)

TYPE is SOURCE for Lisp source (filecreated) files, COMPILED for Lisp compiled files, otherwise the PRINTFILETYPE (TEXT, TEDIT...) or NIL. EOL is CR, LF, CRLF, or NIL.

When both files exist, the date relation is one of <, =, or >. Otherwise, the date relation is \* if only one file exists. EQUIV is EQUIVALENT for files with different dates but exactly the same bytes, otherwise NIL.

If OUTPUTFILE is not NIL, then it is a filename or open stream on which selected entries will be printed (T for the terminal) by CDPRINT.

COMPAREDIRECTORIES sets the variable LASTCDENTRIES is set to the selected entries. This is used by the functions below if their CDENTRIES is NIL.

(CDFILES DIR FILEPATTERNS EXTENSIONSTOAVOID ALLVERSIONS DEPTH) [Function]

Returns a list of full filenames for files in directory DIR (NIL=T=the connected directory) that match the other file-name filtering criteria. Files are excluded if:

Their name does not match a pattern in FILEPATTERNS (NIL = \*). Dotted files are excluded unless FILEPATTERNS includes .\* and files in subdirectories are excluded if the number of subdirectories exceeds DEPTH (below).

Their extension is in the list EXTENSIONSTOAVOID (\* excludes all extensions).

They are not the highest version unless ALLVERSIONS=T.

DEPTH controls the depth of subdirectory exploration. T means all levels, NIL means no subdirectories. Otherwise the maximum number of ">" characters below the starting DIR in the fullname of files.

(CDFILES) produces all the newest, undotted files in the immediate connected directory.

(CDPRINT CDENTRIES FILE PRINTAUTHOR) [Function]

Prints CDENTRIES on FILE, with one line for each entry. The line for each entry is of the form

FILE1 (AUTHOR) SIZE DATE relation DATE FILE2 (AUTHOR) SIZE

For example

```
ACE.;1 (Joe) 4035 2-May-1985 18:03:54 < 30-Sep-1985 11:14:48 ACE.;3 (Sam) 5096.
```

The line for byte-equivalent files is prefixed with ==. If the files are equivalent except for a difference in end-of-line conventions, the equivalence prefix will indicate the convention for each file (C for CR, L, for LF, 2 for CRLF). Thus C2 indicates that the files are equivalent except that file1 marks line ends with CR and file2 with CRLF.

Note that because of the setting of LASTCDENTRIES, evaluating (CDPRINT) after COMPAREDIRECTORIES prints the results of the last comparison.

For conciseness, authors are included only if PRINTAUTHOR or if AUTHOR is included in the SELECT parameter of CDENTRIES. Also, the redundant file-name hosts/directories are not printed.

(CDMAP CDENTRIES FN) [Function]

(CDSUBSET CDENTRIES FN) [Function]

CDMAP applies FN to each CDENTRY in CDENTRIES. CDSUBSET applies FN and also returns the subset of CDENTRIES for which FN is non-NIL and preserves in the value the parameters of CDENTRIES. For convenience, at each invocation the variables MATCHNAME INFO1 DATEREL INFO2 and EQUIV are bound to the corresponding fields and can be used freely by FN.

## USEFUL UTILITIES

(FIX-DIRECTORY-DATES FILES) [Function]

For every file included in or specified by FILES, if it is a Lisp source or compiled whose directory creation date is more than 30 seconds later than its internal filecreated date (presumably because of copying), then its directory date is reset to match the internal date. FILES can be a list of file names or a pattern interpretable by FILDIR. Returns a list of files whose dates have been changed.

(FIX-EQUIV-DATES CDENTRIES) [Function]

If there is an entry in CDENTRIES whose files are EQUIVALENT but with different directory creation dates, the directory date of the file with the later date (presumably a copy) is reset to match the date of the earlier file. In the end all equivalent files will have the same (earliest) date. Returns a list of files whose dates have been changed.

(COPY-MISSING-FILES CDENTRIES TARGET MATCHNAMES) [Function]

Target is 1 or 2, indicating the direction of potential copies. If an entry with a source file but no target file has a matchname in MATCHNAMES, the source file is copied to the target directory. All target-absent files are copied if MATCHNAMES is NIL. Source properties (including version number) are preserved in the target.

(COPY-COMPARED-FILES CDENTRIES TARGET MATCHNAMES) [Function]

Target is 1 or 2, indicating the direction of potential copies. If an entry with both source and target files has a matchname in MATCHNAMES, the source file is copied to a new version of the target file. All files are copied if MATCHNAMES is NIL.

(COMPARE-ENTRY-SOURCE-FILES CDENTRY LISTSTREAM EXAMINE DW?)

[Function]

This is a simple wrapper for calling COMPARE-SOURCES if the CDENTRY files are Lisp source files. The function (CDENTRY MATCHNAME CDENTRIES is useful for extracting a particular entry, with CDENTRIES defaulting to LASTCDENTRIES.

(COMPILED-ON-SAME-SOURCE CDENTRIES) [Function]

Returns the subset of entries with Lisp compiled files (dfasl or lcom) that are compiled on the same source, according to SOURCE-FOR-COMPILED-P below. Presumably one should be removed to avoid confusion.

(FIND-SOURCE-FILES CFILES SDIRS DFASLMARGIN)

[Function]

Returns (CFILE . SFILES) pairs where CFILE is a Lisp compiled file in CFILES and SFILES is list of files in SDIRS that CFILE was compiled on according to SOURCE-FOR-COMPILED-P. This suggests that at least one of SFILES should be copied to CFILE's location (or vice versa).

(FIND-COMPILED-FILES SFILES CDIRS DFASLMARGIN)

[Function]

Returns (SFILE . CFILES) pairs where SFILE is a Lisp source file in SFILES and CFILES are files in CDIRS that are compiled on SFILE according to SOURCE-FOR-COMPILED-P. This suggests that at least one of CFILES should be copied to SFILE's location.

(FIND-UNCOMPILED-FILES FILES DFASLMARGIN COMPILEXTS)

[Function]

Returns a list of elements each of which corresponds to a source file in FILES for which no appropriate compiled file can be found. An appropriate compiled file is a file in the same location with extension in COMPILEEXTS (defaulting to \*COMPILED-EXTENSIONS\*) that satisfies SOURCE-FOR-COMPILED-P. Each element is a list of the form

(sourcefile . cfiles)

cfiles contains compiled files that were compiled on a different version of sourcefile, NIL if no such files exist. Each cfile item is a pair (cfile timediff) where timediff is the time difference (in minutes) between the creation date of the compiled-file's source and the creation date of sourcefile (positive if the cfile was compiled later, as should be the case). FILES can be an explicit list of files, or a file specification interpretable by FILDIR; in that case only the newest source-file versions are processed.

(FIND-UNSOURCED-FILES CFILES DFASLMARGIN COMPILEXTS)

[Function]

Returns the subset of the compiled files specified by CFILES for which a corresponding source file according to SOURCE-FOR-COMPILED-P cannot be found in the same directory. CFILES can be a list of files or a pattern that FILDIR can interpret. COMPILEEXTS can be one or more explicit compile-file extensions, defaulting to \*COMPILED-EXTENSIONS\*.

(SOURCE-FOR-COMPILED-P SOURCE COMPILED DFASLMARGIN)

[Function]

Returns T if it can confirm that Lisp COMPILED file was compiled on Lisp SOURCE file. SOURCE and COMPILED can be provided as CREATED-AS values, to avoid repetitive computation. This compares the information in the filecreated expressions, original file names and original dates, and not the current directory names and dates.

It appears that the times in DFASL files may differ from the filecreated source dates by a few minutes. The DFASLMARGIN can be provided to loosen up the date matching criterion. DFASLMARGIN is a pair (max min) and a DFASL COMPILED is deemed to be compiled on SOURCE if the compiled's source date is no more than max and no less than min minutes after the source date. A negative min allows for the possibility that the compiled-source date is earlier than the candidate source date. DFASLMARGIN defaults to (20 0). A single positive number x is coerced to (x 0). A single negative

number is coerced to `(-x x)` (compiled file is no more than `x` minutes later or earlier). `T` is infinity in either direction. Examples:

`(T 0)`: COMPILED compiled on source later than SOURCE  
`(0 T)`: COMPILED compiled on source earlier than SOURCE (odd)  
`12`: COMPILED compiled on source later than SOURCE by no more than 12 minutes  
`-12`: COMPILED compiled on source 12 minutes before or after SOURCE

`(FIND-MULTICOMPILED-FILES FILES SHOWINFO)` [Function]

Returns a list of files in `FILES` that have more than one type of compiled file (e.g. `LCOM` and `DFASL`). `FILES` is interpretable by `FILDIR`. If `SHOWINFO`, then the value contains a list for each file of the form

`(rootname loaded-version . CREATED-AS information for each compile-type)`

Otherwise just the rootname of the source is returns.

`(CREATED-AS FILE)` [Function]

If `FILE` is a Lisp source or compiled file, returns a record of its original filename and filecreated dates, and for compiled files, also the original compiled-on name and date. The return for a source file is a pair

`(sfullname sfilecreateddate)`

The return for a compiled file is a quadruple

`(cfullname cfilecreated sfullname sfilecreateddate)`

where `sfullname` and `sourcefilecreated` are extracted from the file's compiled-on information. The return is `(fullname NIL)` for a non-Lisp file.

`(EOLTYPE FILE SHOWCONTEXT)` [Function]

Returns the `EOLTYPE` of `FILE` (`CR`, `LF`, `CRLF`) if the type is unmistakable: contains at least one instance of one type and no instances of any others. Returns `NIL` if there is evidence of inconsistent types. If `SHOWCONTEXT` is an integer, it is the number of bytes for `EOLTYPE` to display before and after an instance of an inconsistent type. At each instance, the user is asked whether to continue scanning for other instances. `SHOWCONTEXT = T` is interpreted as 100.

`(BINCOMP FILE1 FILE2 EOLDIFFOK)` [Function]

Returns `T` if `FILE1` and `FILE2` are byte-identical. If `EOLDIFFOK` and `FILE1` and `FILE2` differ only in their eol conventions, the value is a list of the form `(EOL1 EOL2)`, e.g. `(CR CRLF)`. Otherwise the value is `NIL`.



---



---

## COMPARESOURCES

---



---

By: Bill van Melle (vanMelle.pa@Xerox.com)

Browser added by Ron Kaplan (12/2021)

### INTRODUCTION

COMPARESOURCES is a program for comparing two versions of a Lisp source file for differences. The comparison is completely brute-force: COMPARESOURCEs reads the complete contents of both files, and compares all the expressions for differences. The files need not be ones produced by MAKEFILE, as COMPARESOURCEs reads the contents with READFILE; however, the program is tuned for files of the type produced by MAKEFILE.

### HOW TO USE IT

The interface consists of a two functions, COMPARESOURCEs and CSBROWSER.

(COMPARESOURCEs *FILEX FILEY EXAMINE DW? LISTSTREAM*) [Function]

Compares the files named *FILEX* and *FILEY* for differences. For each type of file object (function, variable, record, etc), COMPARESOURCEs identifies which objects of that type differ, and for each such object prints on *LISTSTREAM* a comparison using the function COMPARELISTS. If an object exists on only one of the two files, this fact is noted instead by the message "*name is not on file*".

If *DW?* is true, COMPARESOURCEs calls DWIMIFY on each function body before performing the comparison. This is useful for comparing a file made with CLISP prettyprinted with one made without.

If *EXAMINE* is true, COMPARESOURCEs calls the editor to allow you to more closely examine expressions that differ. Its value is either T, meaning call the editor in all cases, or an atom or a list of atoms chosen from among the following:

|          |                                                                     |
|----------|---------------------------------------------------------------------|
| OLD      | Call the editor for changed objects that are on both files.         |
| NEW      | Call the editor for objects that are on only one file.              |
| MISC     | Call the editor for changed but otherwise unclassified expressions. |
| 2WINDOWS | Call the editor separately for each pair of changed objects.        |

In the OLD and MISC cases, the editor is called on a list of two elements, the two expressions. In the NEW case, the editor is called on just the single new expression.

The value returned by COMPARESOURCEs is a list whose elements are of the form (type . names), listing the names by type of all objects found to be different. Expressions of no particular type are identified collectively as "(Other --)".

(CSBROWSER *FILEX FILEY DW? LABEL1 LABEL2*) [Function]

This directs the output of a call to COMPARESOURCEs to a scrollable window. Clicking on the output for each identified difference brings up for further examination side-by-side SEDIT windows on the

different objects. LABEL1 and LABEL2 are optional strings that override the default way of constructing the title for the browser window.

## FORM OF THE OUTPUT

The output of COMPAREsources is in several sections. First, all functions are compared. Then expressions of other types (variables, macros, etc) are compared. When a difference is found, COMPAREsources prints the name of the object and calls COMPARELISTS, the same Interlisp function called by COMPARE and COMPAREDEFS. Finally, expressions inside of DECLARE: forms are recursively analyzed in a separate section in the same fashion. All DECLARE: forms of the same applicability (e.g., EVAL@COMPILE DONTCOPY) are handled in the same subsection.

The output of COMPARELISTS takes one of three forms. The usual form is an abbreviated printing of the two expressions with equal elements in the two structures denoted by "&" or "-n-" for a subsequence of *n* identical expressions. Identical elements are printed only for purposes of establishing the context of differences. For example,

**COMPAREsources :**

```
(LAMBDA -3- (PROG -16- (for -10- (COND (& (printout & T -4-) -2-)))
(TERPRI --) &))
(LAMBDA -3- (PROG -16- (for -10- (COND (& (printout & -4-) -2-)))
&))
```

indicates that in the function COMPAREsources, an extra argument was added to a `printout` form, and a `(TERPRI --)` expression was added before the final element of the `PROG`. The first 17 elements of the `PROG` form were unchanged, as were the first 11 and last 2 of the `for`.

A more abbreviated form of output occurs when the expressions differ only in a global substitution. In this case, COMPARELISTS prints "*(x -> y)*" to denote that all occurrences of *x* in the first expression were replaced by *y* in the second expression, and there were no other changes.

Finally, COMPARELISTS prints "SAME" if the expressions are "the same". Since COMPAREsources only calls COMPARELISTS when the two expressions are not EQUAL, the output SAME specifically means that the expressions differ only in the bodies of comments (which COMPARELISTS ignores).

## USER EXTENSIONS

COMPAREsources already "knows" about several kinds of file package objects, including FNS, VARS, MACROS, RECORDS, and PROPS. Any expression not identifiable as some particular type is compared as a vanilla expression. You can extend the set of types it knows about by adding to the following list:

COMPARESOURCETYPES [Variable]

The elements of this list are lists of the form

```
(TYPE PREDICATEFN COMPAREFN IDFN DESCRIPTION)
```

as follows:

|      |                                                                                                                    |
|------|--------------------------------------------------------------------------------------------------------------------|
| TYPE | The file package type of the object (or whatever name you wish to give it in the case of fictitious object types). |
|------|--------------------------------------------------------------------------------------------------------------------|

- PREDICATEFN A function of one argument, a single top-level expression as read from the file, that returns true if the expression is of the desired type.
- COMPAREFN A function of three arguments, one expression from each file (both guaranteed to have satisfied the PREDICATEFN), and the listing stream. COMPAREFN should compare the two expressions in some appropriate way, printing its results to the listing stream. A typical COMPAREFN calls the function COMPARELISTS on some subform of the expressions. If COMPAREFN is NIL, COMPARELISTS is used.
- IDFN A function of one argument, an expression, that returns the "name" of the object described by the expression. Two expressions are assumed to define the same object if their names are EQUAL. The name corresponds roughly to a file package name. For example, for type VARS it is the variable name; for type PROPS it is a pair (atom propname). If IDFN is NIL, CADDR is used.
- DESCRIPTION A string identifying the kind of object, for use in the comparison printout. If DESCRIPTION is NIL, (L-CASE TYPE T) is used.



---



---

## COMPARETEXT

---



---

By Mike Sannella. Tested in Medley by Larry Masinter, updated by Ron Kaplan 12/2021

Uses TEDIT, GRAPHER, REGIONMANAGER

### INTRODUCTION

COMPARETEXT is a rather non-standard text file comparison program which tries to address two problems: (1) the problem of detecting certain types of changes, such as detecting when a paragraph is moved to a different part of a document; and (2) the problem of showing the user what changes have been made in a document.

The text comparison algorithm is an adaptation of the one described in the article "A Technique for Isolating Differences Between Files" by Paul Heckel, in CACM, V21, #4, April 1978. The main idea is to break each of the two text files into "chunks" (words, lines, paragraphs, ...), hash each chunk into a hash value, and match up chunks with the same hash value in the two files. This method detects switching two chunks, or moving a chunk anywhere else in the document.

### COMPARING TEXT FILES

Two text files can be compared with the following function:

```
(COMPARETEXT FILE1 FILE2 HASH.TYPE CHUNKREGION FILELABELS TITLE TEXTWIDTH
TEXTHEIGHT) [Function]
```

*FILE1* and *FILE2* are the names of the two files to compare. The order is not important, except that in the resulting graph the *FILE1* information will appear on the left, and the *FILE2* info on the right. The files may also be provide as input streams.

*HASH.TYPE* determines how "chunks" of text are defined; how fine-grained the comparison will be. This can be PARA to hash by paragraphs (delimited by two consecutive CRs), LINE to hash by lines (delimited by one CR), or WORD to hash words (delimited by any white space). *HASH.TYPE*=NIL defaults to PARA.

*CHUNKREGION* is the region on the display screen used for the file comparison graph, the chunk window. If *CHUNKREGION*=NIL, the system asks the user to specify a region, prompting with a region that is just wide enough for the graph. If *CHUNKREGION*=T, a region in the lower left corner is used. IF *CHUNKREGION* is a position, the chunkwindow will be located relative to that position, with its horizontal midpoint at the specified XCOORD and its top at the YCOORD.

*FILELABELS* is an optional pair of labels that will appear over the columns of the difference graph instead of the (often overly long) full names of the files.

*TITLE* is an optional title to be used for the comparison window.

*TEXTWIDTH* and *TEXTHEIGHT* are optional parameters that control the size of each of the two text-display windows.

COMPARETEXT creates a graph with two columns. Each column contains the label for one of the files, and lists the chunks from that file. Each chunk is represented by an atom NNN:MMM, where NNN is the file pointer of the beginning of the chunk within the file, and MMM is the length of the chunk. Lines are drawn from one column to the other to show which chunks in one file are the same as those in the other file. Chunks with no lines going to them do not exist in the other file. [Note: a series of chunks in one file which are the same as a series of chunks in the other file are merged into one big chunk. A series of unconnected chunks is also merged.]

Pressing the LEFT mouse button over one of the chunk nodes causes the node and its counterpart in the other column to be inverted, and read-only Tedit windows are open on the files with the appropriate text selected. If a Tedit window to a file is already active, the selection is simply moved. If COMPARETEXT.AUTOTEDIT is true (initially), then regions are selected automatically for the Tedit windows, otherwise the mouse must be used to specify ghost regions.

Pressing the MIDDLE mouse button over a chunk node raises a pop-up menu with the items: PARA, LINE, and WORD. If one of these is selected, COMPARETEXT is called to compare the selected chunk with the last selected chunks (the ones that are boxed), using the hash type selected, and create a new graph window.

White space (space, tab, CR, LF) is used to delimit chunks, but is ignored when computing the hash value of a chunk. Therefore, if two paragraphs are identical except that one has a few extra CRs after it, they will be considered identical by COMPARETEXT.

If the variable COMPARETEXT.ALLCHUNKS is NIL (initially T), then the graph is abbreviated so that nodes for identical chunks in the same position are not shown.

---

---

**COMPILEBANG**

---

---

By: Nick Briggs (Briggs.pa@Xerox.com)

Required by TRILLIUM

This provides an interface to the compiler that avoids the interview for the common cases of in-core compilation. It contains a single function `COMPILE!`, and the Lisp macros `C` and `EDITMACRO C`:

`(COMPILE! X NOSAVE NOREDEFINE PRINTLAP)` [Function]

Calls the compiler to compile `X`. If `X` is a litatom, its definition is compiled and stored in the function cell unless `NOREDEFINE`, and the old definition if any is saved on the property list unless `NOSAVE`. No printing of lap or machine code is done unless `PRINTLAP`.

Thus, to simply compile the function `BAR`, do `COMPILE!(BAR)`.

`X` may also be a list form. In this case, `COMPILE!` assumes that the user is interested just in seeing how that form would compile. The form is embedded in a Lambda expression and compiled. Of course, there is no function-cell to be stored into or saved.

`C` [Lisp Macro]

The `LISPXMACRO C` calls `COMPILE!`, with `PRINTLAP` on, on the next element of the input line. Thus, `C BAR` will compile, redefine, and save the old definition for `BAR`.

`C (CONS)` will show how a call to `CONS` would compile.

The editmacro `C` calls `COMPILE!` on the current expression if it is a list, or on the form of which the current expression is an element.

---

---

**COURIERDEFS**

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

COURIERDEFS contains a procedure-less Courier program, called INTERLISP, which defines several Envos Lisp types as Courier constructed types or as new Courier primitive types (via a COURIERDEF property) for use with Courier server and client programs. The defined Envos Lisp types include:

|                  |                                                                                                                                                                     |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>ATOM</i>      | Converts to a string on writing and converts to an atom on reading.                                                                                                 |
| <i>FONT</i>      | Converts a FONTDESCRIPTOR to a record describing the font on writing and convert the record back to a FONTDESCRIPTOR on reading.                                    |
| <i>REGION</i>    | A sequence of INTEGER.                                                                                                                                              |
| <i>POSITION</i>  | Converts a POSITION record to two integers on writing and converts back to a POSITION record on reading.                                                            |
| <i>NUMBER</i>    | Like INTEGER but can also be NIL.                                                                                                                                   |
| <i>BRUSH</i>     | Converts the various possibilities for a brush (NIL, INTEGER, BRUSH RECORD etc.) to a CHOICE record on writing, converts back to original specification on reading. |
| <i>OPERATION</i> | An ENUMERATION of NIL, REPLACE, PAINT, INVERT or ERASE.                                                                                                             |
| <i>TEXTURE</i>   | Converts a TEXTURE, NIL or T to a CARDINAL on writing, returns a CARDINAL on reading.                                                                               |

This file is loaded by several other modules that define Courier servers and clients. A Courier program can use the types defined in the INTERLISP program by using the INHERITS slot in the Courier program definition.

---

---

**COURIEREVALSERVE**

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: COURIERSERVE

COURIEREVALSERVE implements both the client and server routines for the simple remote evaluation server described in the COURIERSERVE documentation.

The module defines two user functions:

(REMOTEEVAL *FORM* *COURIERSTREAM* [*NOERRORFLG*]) [Function]

(REMOTEAPPLY *FN* *ARGS* *COURIERSTREAM* [*NOERRORFLG*]) [Function]

*COURIERSTREAM* is obtained by calling *COURIER.OPEN* to connect with a host that is running the Courier server and has *COURIEREVALSERVE* loaded. If the *NOERRORFLG* is non-NIL, it is returned if an error is signaled by the remote host, otherwise the functions generate an error.

Due to the removal of *ERRORN* as of the Lyric release, the error handling is not as informative as in earlier versions. If you are connected to a pre-Lyric host, errors will work as before, otherwise instead of signaling the actual remote error (eg. “*Undefined car of form*”) the generic “*Remote evaluation error!*” error is raised. This is to maintain backward compatibility in the EVAL Courier program. Hopefully, this will be replaced by a new version of the EVAL program designed to correctly remote the new condition-based error handler.

---

---

## COURIERIMAGESTREAM

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: COURIERSERVE, COURIERDEFS and BITMAPFNS

COURIERIMAGESTREAM implements a Courier client and server program which allows remote hosts to do *image stream* manipulations on other workstations via the network. To do this, it defines the COURIER virtual image stream type which allows the user to manipulate remote image streams through local image streams.

### THE IMAGESTREAM COURIER PROGRAM

The module defines a Courier program called IMAGESTREAM (which inherits from the INTERLISP Courier program defined in COURIERDEFS). For each IMAGEOP in the IMAGEOPs definition, there is an equivalent Courier procedure in the IMAGESTREAM program. The module contains the code for both the Courier client and server.

### OPENING AND CLOSING REMOTE IMAGE STREAMS

Remote image streams can be opened using either the COURIER image stream type or using direct Courier calls.

#### The COURIER Image Stream Interface

Remote Courier image streams can be opened using:

```
(SETQ COURIERSTREAM (COURIER.OPEN HOST)
 (OPENIMAGESTREAM COURIERSTREAM 'COURIER OPTIONS))
```

which returns an image stream. The OPTIONS can include FILE and IMAGETYPE which are passed to OPENIMAGESTREAM on the remote host and if not supplied, a nameless DISPLAY image stream is opened. All other options are passed to the remote image stream. The image stream can be closed using CLOSEF.

#### The Courier Procedure Call Interface

Courier image streams can also be opened using Courier procedure calls from any Courier client with the Courier procedure:

```
(OPEN 0 (FILE IMAGETYPE) RETURNS (HANDLE) REPORTS NIL)
```

which is invoked from Lisp by doing:

```
(COURIER.CALL COURIERSTREAM 'IMAGESTREAM 'OPEN FILE IMAGETYPE OPTIONS)
```

where FILE, IMAGETYPE and OPTIONS are similar to the arguments to OPENIMAGESTREAM.

This call will return a handle to be used with the remainder of the IMAGESTREAM procedures. To close an image stream from a Courier client use the Courier procedure:

```
(CLOSE 1 (HANDLE) RETURNS NIL REPORTS NIL)
```

which is invoked from Lisp by doing:

```
(COURIER.CALL COURIERSTREAM 'IMAGESTREAM 'CLOSE HANDLE)
```

### **DIFFERENCES BETWEEN IMAGEOPS AND IMAGESTREAM COURIER PROCEDURES**

All of the IMAGEOPs are implemented in the COURIER image stream type as it merely passes the call to the IMAGEOPs of another image stream type on the remote host. No error checking is done, so invoking an illegal IMAGEOP will cause a Courier rejection of the call.

The arguments to the IMAGESTREAM Courier procedures are generally in the same order as the arguments to the various IM\* functions which implement an image stream (stream argument first). An exception is BITBLT (and SCALED BITBLT) which is defined as follows:

```
(BITBLT 32 (HANDLE BULK.DATA.SOURCE LEFT BOTTOM WIDTH HEIGHT
 SOURCETYPE OPERATION TEXTURE CLIPPINGREGION))
```

The BULK.DATA.SOURCE argument is used to transfer the bitmap using WRITEBINARYBITMAP. This is only relevant to direct Courier calls, the COURIER image stream BITBLT operation hides the differences.

When using the COURIER image stream type, the STRINGWIDTH, CHARWIDTH etc. IMAGEOPs are handled locally, not via Courier calls, to improve efficiency.

### **IMAGESTREAM PROGRAM VERSIONS**

The current implementation of the IMAGESTREAM Courier program is version **1**. This module also has the previous version of the program (**0**) defined as OLDIMAGESTREAM (just the procedure definitions that differ, it inherits from IMAGESTREAM). This allows the current version of the program to accept calls from older versions, but not vice-versa. However, the new version of the IMAGESTREAM Courier program can be loaded and used with the old (pre-Lyric) functions.

---



---

## COURIERSERVE

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

COURIERSERVE implements a Courier server process for Envos Lisp allowing other hosts to make Courier calls into the workstation. The server supports both multiple Courier stream connections as well as *expedited* (single packet) and *broadcast* calls.

### STARTING A COURIER SERVER

The Courier server can be started by evaluating:

(COURIER.START.SERVER [RESTART]) [Function]

Once the server is running, it can be invoked by a remote host client using COURIER.OPEN for a Courier stream connection or by using COURIER.EXPEDITED.CALL or COURIER.BROADCAST.CALL for expedited calls. The functions for making Courier client calls from Lisp are documented in the *Interlisp-D Reference Manual* (pages 31.15–31.26).

(COURIER.RESET.SOCKET) [Function]

(Re)Opens and closes the Courier socket. Not normally a user routine, this function is called by COURIER.START.SERVER but it can be called directly if “*socket already open!*” errors persist on the Courier socket (5).

### DEFINING A COURIER SERVER FUNCTION

Defining a Courier server program is identical to defining a client program except for the additional field IMPLEMENTEDBY in each procedure in the PROCEDURES section of the Courier program definition:

```
PROCEDURES
((LAYOUT 0 (GRAPHNODES ROOTIDS FORMAT FONT MOTHERD PERSONALD FAMILYD)
 RETURNS (GRAPH)
 REPORTS (LAYOUT.ERROR)
 IMPLEMENTEDBY GRAPH.REMOTELAYOUT))
```

The order of the RETURNS, REPORTS and IMPLEMENTEDBY fields is significant and should be maintained.

The *server function*, named in the IMPLEMENTEDBY field, is invoked when a Courier call to the procedure is made. The server function is applied to the Courier stream, the Courier program and the Courier procedure followed by the arguments named in the Courier definition. The arguments for GRAPH.REMOTELAYOUT would be (COURIERSTREAM PROGRAM PROCEDURE GRAPHNODES ROOTIDS FORMAT ...).

Note that the COURIERSTREAM, PROGRAM and PROCEDURE arguments are not necessarily used, they are made available for implementing special servers.



## RETURNING VALUES FROM A COURIER PROGRAM

Results or errors can be returned by a Courier server function by one of two different methods. In the usual, simple case, the server function can return as its result a list starting with one of RETURN, ABORT or REJECT followed by the appropriate values.

For the RETURN result, the tail of the list should be the results as defined in the Courier procedure definition, eg. (RETURN 23 "John").

For the ABORT result, the tail of the list should contain the reason for the abnormal termination (as defined in the Courier program), followed by any error arguments, eg. (ABORT NAME.NOT.FOUND "John").

For the REJECT result, the tail of the list should contain the rejection error as defined in the Courier standard. The only rejection that should occur inside a server function should be UNSPECIFIED if the program needs to reject for any reason. The other rejection types are handled by the Courier server.

Alternatively, the server function can return results directly to the Courier stream and return NIL as its result. To return results directly to the Courier stream use:

(COURIER.RETURN *COURIERSTREAM PROGRAM PROCEDURE RESULTLIST*) [Function]

(COURIER.ABORT *COURIERSTREAM PROGRAM ERROR RESULTLIST*) [Function]

(COURIER.REJECT *COURIERSTREAM ERROR RESULTLIST*) [Function]

## EXPEDITED AND BROADCAST COURIER CALLS

The Courier server allows expedited and broadcast Courier calls. The only difference the server function would see if invoked due to an expedited call is that the Courier stream it is handed is actually a record containing an XIP packet and a socket. If the server function does not use the Courier stream directly, then this difference is invisible.

If the server function actually needs a Courier stream to operate (eg. an NS CHAT server), then it should probably include an USE.COURIER abort error in its definition. If the server function needs a Courier stream due to *bulk data* arguments, this will be trapped in the Courier server itself, which will reject appropriately and not invoke the server function.

## USING BULK DATA IN A SERVER FUNCTION

If a server function takes a bulk data argument (either BULK.DATA.SINK or BULK.DATA.SOURCE), it is handed an open bulk data stream for that argument when invoked. If the server function returns a result by returning one of the RETURN or ABORT forms as its result, the bulk data stream will be closed automatically. If the server function returns results directly to the Courier stream using COURIER.RETURN or COURIER.ABORT, then the server function must first close the bulk data stream using:

(CLOSE.BULK.DATA *STREAM [ABORTFLG]*) [Function]

The CLOSEF function does not work on the bulk data stream argument and using it will hang the Courier connection. Only the *immediate* bulk data transfer type is handled. NULL, ACTIVE or PASSIVE bulk data transfer types will cause a Courier rejection of type UNSPECIFIED.

## SIMPLE SERVER DEFINITION

Below is the Courier definition for a simple evaluation server. The two functions EVAL.REMOTE and APPLY.REMOTE are all that would need to be defined to make the server run:

```

((1105 0)
 TYPES ((SEXPR STRING)
 (FN STRING)
 (ARGS (SEQUENCE SEXPR))
 (ERRORN (RECORD (ERROR.NUMBER CARDINAL)
 (ERROR.MESSAGE SEXPR))))
 PROCEDURES ((EVAL 0 (SEXPR)
 RETURNS (SEXPR)
 REPORTS (REMOTE.EVAL.ERROR REMOTE.READ.ERROR)
 IMPLEMENTEDBY EVAL.REMOTE)
 (APPLY 1 (FN ARGS)
 RETURNS (SEXPR)
 REPORTS (REMOTE.APPLY.ERROR REMOTE.READ.ERROR)
 IMPLEMENTEDBY APPLY.REMOTE))
 ERRORS ((REMOTE.EVAL.ERROR 0 (ERRORN))
 (REMOTE.APPLY.ERROR 1 (ERRORN))
 (REMOTE.READ.ERROR 2 (ERRORN)))
)
```

**RELATED FILES**

The modules CHATSERVER-NS, COURIERDEFS, COURIEREVALSERVE, COURIERIMAGESTREAM, MONITOR, REMOTEPSW and NSTALK all define Courier servers and/or Courier type definitions.

---



---

## CROCK

---



---

By: Kelly Roach

New Owner: Herb Jellinek (Jellinek.pa@Xerox.com)

CROCK sets up an analog face clock in the user's environment. To use, LOAD CROCK.LCOM and call (CROCK). CROCK requires that PROCESSWORLD be running (automatic in Fugue or later).

### CROCK

Function CROCK has the form

(CROCK *REGION*) [Function]

The first invocation creates a clock window, CROCKWINDOW, occupying *REGION* with style CROCK.DEFAULT.STYLE. If *REGION* is left NIL, a region will be prompted for. Subsequent invocations use CROCKWINDOW. Only one clock window may exist at any given time. The clock is updated once a minute.

### STYLE

The clock's style is maintained as a property list and can be found by (WINDOWPROP CROCKWINDOW 'STYLE). There are four independent boolean properties which the user may control: HANDS (the hands of the clock), TIMES (time digits printed where the hands end), RINGS (rings on the clock face), and NUMBERS (12 numbers around the outside of the clock face). The style first used will be CROCK.DEFAULT.STYLE (bound to '(HANDS T TIMES NIL RINGS NIL NUMBERS T) when CROCK is first loaded).

### CROCK.DATEFORMAT

The user can control how the date will be printed in CROCKWINDOW. CROCK.DATEFORMAT should have the form (DATEFORMAT . <tokens>) where each <token> is one of NO.DATE, NO.TIME, NUMBER.OF.MONTH, YEAR.LONG, SLASHES, SPACES, NO.LEADING.SPACES, TIME.ZONE, or NO.SECONDS. These are all listed on pp23.57-58 of the IRM. Unfortunately, some other possibilities, such as DAY.OF.WEEK have not been implemented by Interlisp-D yet and are therefore not available to CROCK yet. The default value for CROCK.DATEFORMAT is (DATEFORMAT NO.SECONDS). For example,

```
(SETQ CROCK.DATEFORMAT
 '(DATEFORMAT SLASHES NUMBER.OF.MONTH NO.SECONDS))
```

would make CROCK print a date string like

28/09/84 14:53

instead of a date string like

28-Sep-84 14:53

Since CROCK updates itself only once a minute, it is probably a good idea to always include NO.SECONDS in your CROCK.DATEFORMAT.

### **CROCK.ALARM AND CROCK.TUNE**

The user can set CROCK's alarm via

(CROCK.ALARM *DATESTRING*) [Function]

where DATESTRING is any arg acceptable to Interlisp's IDATE (such as the date CROCK prints in CROCKWINDOW). CROCK will act appropriately when time reaches DATESTRING. Dandelion users can set global CROCK.TUNE to a tune to be played by Interlisp's PLAYTUNE when CROCK's alarm acts.

### **RECOMMENDED USAGE**

The simplest way to call CROCK from your init file or other function is to set your CROCK globals, then call CROCK:

(SETQ CROCK.DEFAULT.STYLE *STYLE*) [Variable]

(SETQ CROCK.DATEFORMAT *DATEFORMAT*) [Variable]

(SETQ CROCK.TUNE *TUNE*) [Variable]

(CROCK *REGION*) [Function]

You supply <style>, <dateformat>, <tune>, and <region>. You only need the SETQs if you want non-default values. If no <region> is supplied, CROCK will prompt for one.

### **LEFT MOUSE BUTTON**

Buttoning CROCKWINDOW with the left mouse button requests immediate update of the clock. (Of course, it may take a while for the process scheduler to get to it.)

### **MIDDLE MOUSE BUTTON**

Buttoning CROCKWINDOW with the middle mouse button presents a menu of commands for modifying the clock's style. Menu item SHOW.STYLE prints the clock's style.

### **RIGHT MOUSE BUTTON**

Buttoning CROCKWINDOW with the left mouse button presents the usual window menu. RESHAPEing the CROCKWINDOW causes the clock to change its size to fit the new window region. CLOSEing the CROCKWINDOW deletes the clock process.

---



---

## DATEFORMAT-EDITOR

---



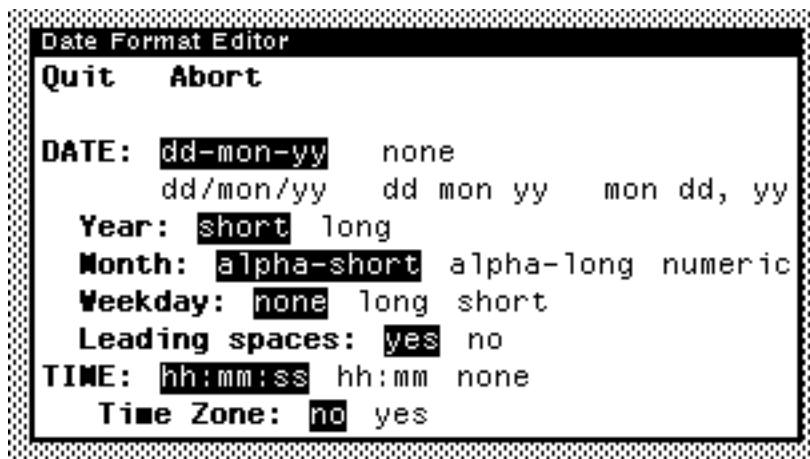
---

By: Johannes A. G. M. Koomen  
 (Koomen.wbst@Xerox.com or Koomen@CS.Rochester.edu)

This document last edited on May 19, 1989 by Bill van Melle.

### DESCRIPTION

DATEFORMAT-EDITOR provides a menu-based interface for creating and editing date formatting lists (see IRM, Section 12.5). The menu is a Free Menu (see FREEMENU in Medley Release Notes), and looks like:



### INTERFACE

(EDIT-DATEFORMAT *DATEFORMAT*)

[Function]

DATEFORMAT is either NIL or the value returned from a call to the function DATEFORMAT (see IRM, Section 12.5). EDIT-DATEFORMAT starts by pre-selecting date formatting keys according to DATEFORMAT, or default ones if DATEFORMAT is NIL. It then enters a busy-wait loop, blocking until the DateFormat Editor window is closed, or **Quit** or **Abort** is selected. EDIT-DATEFORMAT returns a new value obtained from the function DATEFORMAT given the selected date formatting keys if **Quit** was selected, otherwise NIL.

DATEFORMAT-EDITOR-ITEMS

[Variable]

A list of items acceptable to the function FM.FORMATMENU (see FREEMENU in the Release Notes). Unfortunately, some of the date format details are embedded in the DateFormat Editor, rather than in these items, so leave ID's and LABEL's alone, otherwise mung around to your heart's content if you desire a different layout for the DateFormat Editor. Initial value is reflected by the screen snap above.

(GET-DATEFORMAT-EDITOR *RECOMPUTE?*)

[Document Object]

Returns the FreeMenu window of the DateFormat Editor. If *RECOMPUTE?* is non-NIL, recomputes the FreeMenu. Use this function with argument T if you change the variable DATEFORMAT-EDITOR-ITEMS.

### EXTENDED DATEFORMAT OPTIONS

DATEFORMAT-EDITOR supports the following additional DATEFORMAT options by virtue of loading the module DATEPATCH:

MONTH.LONG

[DateFormat Option]

Provides for full names of months rather than the first three characters. For instance, "20 February 1987" is produced by (GDATE NIL (DATEFORMAT MONTH.LONG YEAR.LONG SPACES NO.TIME)).

MONTH.LEADING

[DateFormat Option]

Causes the month to appear before the day. For instance, "February 20, 1987" is produced by (GDATE NIL (DATEFORMAT MONTH.LEADING MONTH.LONG YEAR.LONG NO.TIME)). MONTH.LEADING implies SPACES and disables NUMBER.OF.MONTH.

DATESORT

[Internal Lafite Utility]

Datesort places an item on the background menu. When selected you will be prompted for the name of a Lafite Mail folder to be sorted, and the name of a new resulting sorted folder. The folder messages will then be sorted in ascending order on their "Date:" fields (this takes some time).

---

---

**DEBUGGER-CONTEXT**

---

---

By: Herb Jellinek (Jellinek.pa@xerox.com)

This document last edited on August 13, 1987.

## Introduction

When debugging Common Lisp programs, have you ever wished that the Xerox Lisp debugger let you do things in the right lexical context? Ever wish you could access locally-defined functions, return from blocks, or evaluate variables in the debugger without resorting to the inspector? Ever wish you could tell the "big boys" in Washington a thing or two? Well, two out of three ain't bad. DEBUGGER-CONTEXT gives you the ability to access and modify the lexical context of code you are debugging in a straightforward way.

## User Interface

DEBUGGER-CONTEXT defines two debugger commands that affect the debugger's lexical context.

`lex` [Debugger Command]

Sets the lexical context of the debugger window to that of the stack frame selected in the backtrace window (`il:lastpos`). Henceforth, all evaluation done in this debugger window will be with respect to that lexical environment. The lexical environment of a frame is taken to be the first one encountered as an argument in that frame. Once the `lex` command is given, selecting other stack frames in the backtrace window will not change the lexical environment; you must issue another `lex` command to do so.

The lexical environment of a frame is taken to be the first one encountered as an argument in that frame.

`unlex` [Debugger Command]

Removes the current lexical context; all evaluation will subsequently be done with respect to the current dynamic environment.



---

---

**DECL**

---

---

UNSUPPORTED

INTERNAL

Uses: SIMPLIFY, LABEL and LAMBDATRAN

**NOTE TO LYRIC/MEDLEY USERS**

The DECL module is not supported in Lyric/Medley since it uses the DWIM facilities heavily, and DWIM is not supported in Lyric/Medley. It is being released as a LispUsers module only for backward compatibility. The DECL module only runs under the OLD-INTERLISP-T executive, and all the code that uses it will also have to run under this executive. Therefore, you may wish to convert code that uses DECL to something else.

**INTRODUCTION**

The Decl LispUsers package is contained on the file DECL.LCOM. The Decl package requires the LambdaTran package. LAMBDATRAN.LCOM will automatically be loaded with Decl if it is not already present.

The Decl package extends Interlisp to allow the user to declare the types of variables and expressions appearing in functions. It provides a convenient way of constraining the behavior of programs when the generality and flexibility of ordinary Interlisp is either unnecessary, confusing, or inefficient.

Decl provides a simple language for declarations, and augments the interpreter and the compiler to guarantee that these declarations are always satisfied. The declarations make programs more readable by indicating the type, and therefore something about the intended usage, of variables and expressions in the code. They facilitate debugging by localizing errors that manifest themselves as type incompatibilities. Finally, the declaration information is available for other purposes: compiler macros can consult the declarations to produce more efficient code; coercions for arguments at user interfaces can be automatically generated; and the declarations will be noticed by the Masterscope function analyzer.

The declarations interpreted by the Decl package are in terms of a set of declaration types called *decltypes*, each of which specifies a set of acceptable values and also (optionally) other type-specific behavior. The Decl package provides a set of facilities for defining decltypes and their relations to each other, including type-valued expressions and a comprehensive treatment of union types.

The following description of the Decl package is divided into three parts. First, the syntactic extensions that permit the concise attachment of declarations to program elements are discussed. Second, the mechanisms by which new decltypes can be defined and manipulated are covered. Finally, some additional capabilities based on the availability of declarations are outlined.

**USING DECLARATIONS IN PROGRAMS**

Declarations may be attached to the values of arbitrary expressions and to LAMBDA and PROG variables throughout (or for part of) their lexical scope. The declarations are attached using constructs that resemble the ordinary Interlisp LAMBDA, PROG, and PROGN, but which also permit the expression of declarations. The following examples illustrate the use of declarations in programs.

Consider the following definition for the factorial function (FACT *N*):

```
[LAMBDA (N)
 (COND
 ((EQ N 0) 1)
 (T (ITIMES N (FACT (SUB1 N))
```

Obviously, this function presupposes that *N* is a number, and the run-time checks in ITIMES and SUB1 will cause an error if this is not so. For instance, (FACT T) will cause an error and print the message NON-NUMERIC ARG T. By defining FACT as a DLAMBDA, the Decl package analog of LAMBDA, this presupposition can be stated directly in the code:

```
[DLAMBDA ((N NUMBERP))
 (COND
 ((EQ N 0) 1)
 (T (ITIMES N (FACT (SUB1 N))
```

With this definition, (FACT T) will result in a NON-NUMERIC ARG T error when the body of the code is executed. Instead, the NUMBERP declaration will be checked when the function is first entered, and a *declaration fault* will occur. Thus, the message that the user will see will not dwell on the offending value T, but instead give a symbolic indication of what variable and declaration were violated, as follows:

```
DECLARATION NOT SATISFIED
((N NUMBERP) BROKEN):
```

The user is left in a break from which the values of variables, e.g., *N*, can be examined to determine what the problem is.

The function FACT also makes other presuppositions concerning its argument, *N*. For example, FACT will go into an infinite recursive loop if *N* is a number less than zero. Although the user could program an explicit check for this unexpected situation, such coding is tedious and tends to obscure the underlying algorithm. Instead, the requirement that *N* not be negative can be succinctly stated by declaring it to be a subtype of NUMBERP that is restricted to non-negative numbers. This can be done by adding a SATISFIES clause to *N*'s type specification:

```
[DLAMBDA ([N NUMBERP (SATISFIES (NOT (MINUSP N))
 (COND
 ((EQ N 0) 1)
 (T (ITIMES N (FACT (SUB1 N))
```

The predicate in the SATISFIES clause will be evaluated after *N* is bound and found to satisfy NUMBERP, but before the function body is executed. In the event of a declaration fault, the SATISFIES condition will be included in the error message. For example, (FACT -1) would result in:

```
DECLARATION NOT SATISFIED
```

```
((N NUMBERP (SATISFIES (NOT (MINUSP N))) BROKEN) :
```

The DLAMBDA construct also permits the type of the value that is returned by the function to be declared by means of the pseudo-variable RETURNS. For example, the following definition specifies that FACT is to return a positive integer:

```
[DLAMBDA ([N NUMBERP (SATISFIES (NOT (MINUSP N)
 [RETURNS FIXP (SATISFIES (IGREATERP VALUE 0))
 (COND
 ((EQ N 0) 1)
 (T (ITIMES N (FACT (SUB1 N))
```

After the function body is evaluated, its value is bound to the variable VALUE and the RETURNS declaration is checked. A declaration fault will occur if the value is not satisfactory. This prevents a bad value from propagating to the caller of FACT, perhaps causing an error far away from the source of the difficulty.

Declaring a variable causes its value to be checked not only when it is first bound, but also whenever that variable is reset by SETQ within the DLAMBDA. In other words, the type-checking machinery will not allow a declared variable to take on an improper value. An iterative version of the factorial function illustrates this feature in the context of a DPROG, the analog of PROG:

```
[DLAMBDA ([N NUMBERP (SATISFIES (NOT (MINUSP N)
 [RETURNS FIXP (SATISFIES (IGREATERP VALUE 0))
[DPROG ([TEMP 1 FIXP (SATISFIES (IGREATERP TEMP 0)
 [RETURNS FIXP (SATISFIES (IGREATERP VALUE 0))
 LP (COND ((EQ N 0) (RETURN TEMP)))
 (SETQ TEMP (ITIMES N TEMP))
 (SETQ N (SUB1 N))
 (GO LP]
```

DPROG declarations are much like DLAMBDA declarations, except that they also allow an initial value for the variable to be specified. In the above example, TEMP is declared to be a positive integer throughout the computation and N is declared to be non-negative. Thus, a bug which caused an incorrect value to be assigned by one of the SETQ expressions would cause a declaration failure. Note that the RETURNS declaration for a DPROG is also useful in detecting the common bug of omitting an explicit RETURN.

### DLAMBDA's

The Decl package version of a LAMBDA expression is an expression beginning with the atom DLAMBDA. Such an expression is a function object that may be used in any context where a LAMBDA expression may be used. It resembles a LAMBDA expression except that it permits declaration expressions in its argument list, as illustrated in the examples given earlier. Each element of the argument list of a DLAMBDA may be a literal atom (as in a conventional LAMBDA) or a list of the form (NAME TYPE .EXTRAS). Strictly, this would require a declaration with a SATISFIES clause to take the form (N (NUMBERP (SATISFIES --)) --). However, due to the frequency with which this construction is used, it may be written without the inner set of parentheses, e.g., (N NUMBERP (SATISFIES --) --).

*NAME* fulfills the standard function of a parameter, i.e., providing a name to which the value of the corresponding argument will be bound.

*TYPE* is either a Decl package type name or type expression. When the DLAMBDA is entered, its arguments will be evaluated and bound to the corresponding argument names, and then, after all the argument names have been bound, the declarations will be checked. The type checking is delayed so that SATISFIES predicates can include references to other variables bound by the same DLAMBDA. For example, one might wish to define a function whose two arguments are not only both required to be of some given type, but are also required to satisfy some relationship (e.g., that one is less than the other).

*EXTRAS* allows some additional properties to be attached to a variable. One such property is the accessibility of *NAME* outside the current lexical scope. Accessibility specifications include the atoms LOCAL or SPECIAL, which indicate that this variable is to be compiled so that it is either a LOCALVAR or a SPECVAR, respectively. This is illustrated by the following example:

```
[DLAMBDA ((A LISTP SPECIAL)
 (B FIXP LOCAL))
...]
```

A more informative equivalent to the SPECIAL key word is the USEDIN form, the tail of which can be a list of the other functions that are expected to have access to the variable.<sup>1</sup>

```
[DLAMBDA ((A LISTP (USEDIN FOO FIE))
 (B FIXP LOCAL))
...]
```

*EXTRAS* may also include a comment in standard format, so that descriptive information may be given where a variable is bound:

```
[DLAMBDA ((A LISTP (USEDIN FOO FIE)) (* This is an important variable))
 (B FIXP LOCAL))
...]
```

As mentioned earlier, the value returned by a DLAMBDA can also be declared, by means of the pseudo-variable RETURNS. The RETURNS declaration is just like other DLAMBDA declarations, except (1) in any SATISFIES predicate, the value of the function is referred to by the distinguished name VALUE; and (2) it makes no sense to declare the return value to be LOCAL or SPECIAL.

## DPROG

Just as DLAMBDA resembles LAMBDA, DPROG is analogous to PROG. As for an ordinary PROG, a variable binding may be specified as an atom or a list including an initial value form. However, a DPROG binding also allows *TYPE* and *EXTRAS* information to appear following the initial value form. The format for these augmented variable bindings is (*NAME INITIALVALUE TYPE .EXTRAS*).

The only difference between a DPROG binding and a DLAMBDA binding is that the second position is interpreted as the initial value for the variable. Note that if the user wishes to supply a type declaration for a variable, an initial value *must* be specified. The same rules apply for the interpretation of the type information for DPROGs as for DLAMBDA, and the same set of optional *EXTRAS* can be used. DPROGs may also declare the type of the value they return, by specifying the pseudo-variable RETURNS.

Just as for a DLAMBDA, type tests in a DPROG are not asserted until after all the variables have been bound, thus permitting predicates to refer to other variables being bound by this DPROG. If NIL

appears as the initial value for a binding (i.e., the atom NIL actually appears in the code, not simply an expression that evaluates to NIL) the initial type test will be suppressed, but subsequent type tests, e.g., following a SETQ, will still be performed.

A common construct in Lisp is to bind and initialize a PROG variable to the value of a complicated expression in order to avoid recomputing it, and then to use this value in initializing other PROG variables, e.g.

```
[PROG ((A EXPRESSION))
 (RETURN (PROG ((B... (A...))
 (C... (A...))))
 ...]
```

The ugliness of such constructions in conventional Lisp often tempts the programmer to loosen the scoping relationships of the variables by binding them all at a single level and using SETQ's in the body of the PROG to establish the initial values for variables that depend on the initial values of other variables, e.g.,

```
[PROG ((A EXPRESSION) B C)
 (SETQ B (...A...))
 (SETQ C (...A...))
 ...]
```

In the Decl package environment, this procedure undermines the protection offered by the type mechanism by encouraging the use of uninitialized variables. Therefore, the DPROG offers a syntactic form to encourage more virtuous initialization of its variables. A DPROG variable list may be segmented by occurrences of the special atom THEN, which causes the binding of its variables in stages, so that the bindings made in earlier stages can be used in later ones, e.g.,

```
[DPROG ((A (LENGTH FOO) FIXP LOCAL)
 THEN (B (SQRT A) FLOATP)
 THEN (C (CONS A B) LISTP))
 ...]
```

Each stage is carried out as a conventional set of DPROG bindings (i.e., simultaneously, followed by the appropriate type testing). This layering of the bindings permits one to gradually descend into a inner scope, binding the local names in a very structured and clean fashion, with initial values type-checked as soon as possible.

## DECLARATIONS IN ITERATIVE STATEMENTS

The CLISP iterative statement provides a very useful facility for specifying a variety of PROGS that follow certain widely used formats. The Decl package allows declarations to be made for the scope of an iterative statement via the DECLARE CLISP (I.S. operator). DECLARE can appear as an operator anywhere in an iterative statement, followed by a list of declarations, for example:

(for J from 1 to 10 declare (J FIXP) do. . .

Note that DECLARE declarations do not create bindings, but merely provide declarations for existing bindings. For this reason, an initial value cannot be specified and the form of the declaration is the same as that of DLAMBDA's, namely create (*NAME TYPE . EXTRAS*).

Note that variables bound *outside* of the scope of the iterative statement, i.e., a variable used freely in the I.S., can also be declared using this construction. Such a declaration will only be in effect for the scope of the iterative statement.

### DECLARING A VARIABLE FOR A RESTRICTED LEXICAL SCOPE

The Decl package also permits declaring the type of a variable over some restricted portion of its existence. For example, suppose the variable *X* is either a fixed or floating number, and a program branches to treat the two cases separately. On one path *X* is known to be fixed, whereas on the other it is known to be floating. The Decl package DPROGN construct can be used in such cases to state the type of the variable along each path. DPROGN is exactly like PROGN, except that the second element of the form is interpreted as a list of DLAMBDA format declarations. These declarations are added to any existing declarations in the containing scope, and the composite declaration (created using the ALLOF type expression), is considered to hold throughout the lexical scope created by the DPROGN. Thus, our example becomes:

```
(if (FIXP X)
 then (DPROGN ((X FIXP))...else (DPROGN ((X FLOATP)) ...))
```

Like DPROG and DLAMBDA, the value of a DPROGN may also be declared, using the pseudo-variable RETURNS.

DPROGN may be used not only to restrict the declarations of local variables, but also to declare variables that are being used freely. For example, if the variable *A* is used freely inside a function but is known to be FIXP, this fact could be noted by enclosing the body of the function in (DPROGN ((A FIXP FREE)) *BODY*). Instead of FREE, the more specific construction (BOUNDIN *FUNCTION1 FUNCTION 2. . .*) can be used. This not only states that the variable is used freely but also gives the names of the functions that might have provided this binding.<sup>2</sup>

Since the DPROGN form introduces another level of parenthesization, which results in the enclosed forms being prettyprinted indented, the Decl package also permits such declarations to be attached to their enclosing DLAMBDA or DPROG scopes by placing a DECL expression, e.g., (DECL (A FIXP (BOUNDIN FUM))), before the first executable form in that scope. Like DPROGN's, DECL declarations use DLAMBDA format.

### DECLARING THE VALUES OF EXPRESSIONS

The Decl package allows the value of an arbitrary form to be declared with the Decl construct THE. A THE expression is of the form (THE *TYPE . FORMS*), e.g., (THE FIXP (FOO X)). FORMS are evaluated in order, and the value of the *last* one is checked to see if it satisfies *TYPE*, a type name or type expression. If so, its value is returned, otherwise a declaration fault occurs.

### ASSERTIONS

The Decl package also allows for checking that an arbitrary predicate holds at a particular point in a program's execution, e.g., a condition that must hold at function entry but not throughout its execution. Such predicates can be checked using an expression of the form (ASSERT *FORM1 FORM2*), in which each *FORM1* is either a list (which will be evaluated) or a variable (whose declaration will be checked). Unless all elements of the ASSERT form are satisfied, a declaration fault will take place.

ASSERTing a variable provides a convenient way of verifying that the value of the variable has not been improperly changed by a lower function. Although a similar effect could be achieved for predicates by explicit checks of the form (OR *PREDICATE* (SHOULDNT)), ASSERT also provides the ability both to check that a variable's declaration is currently satisfied and to remove its checks at compile time without source code modification (see COMPILEIGNOREDECL).

### USING TYPE EXPRESSIONS AS PREDICATES

The Decl package extends the Record package TYPE? construct so that it accepts decltypes, as well as record names, e.g., (TYPE? (FIXP (SATISFIES (ILESSP VALUE 0))) EXPR). Thus, a TYPE? expression is exactly the same as a THE expression except that, rather than causing a declaration fault, TYPE? is a predicate that determines whether or not the value satisfies the given type.

## ENFORCEMENT

The Decl package is a “soft” typing system—that is, the data objects themselves are not inherently typed. Consequently, declarations can only be enforced within the lexical scope in which the declaration takes place, and then only in certain contexts. In general, changes to a variable’s value such as those resulting from side effects to embedded structure (e.g., RPLACA, SETN, etc.) or free variable references from outside the scope of the declaration cannot be, and therefore are not, enforced.

Declarations *are* enforced, i.e., checked, in three different situations: when a declared variable is bound to some value or rebound with SETQ or SETQQ, when a declared expression is evaluated, and when an ASSERT expression is evaluated. In a binding context, the type check takes place *after* the binding, including any user-defined behavior specified by the type’s binding function. Any failure of the declarations causes a break to occur and an informative message to be printed. In that break, the name to which the declaration is attached (or *VALUE* if no name is available) will be bound to the offending value. Thus, in the FACT T example above, N would be bound to T. The problem can be repaired either by returning an acceptable value from the break via the RETURN command, or by assigning an acceptable value to the offending name and returning from the break via an OK or GO command. The unsatisfied declaration will be reasserted when the computation is continued, so an unacceptable value will be detected.<sup>3</sup>

The automatic enforcement of type declarations is a very flexible and powerful aid to program development. It does, however, exact a considerable run-time cost because of all the checking involved. Factors of two to ten in running speed are not uncommon, especially where low-level, frequently used functions employ type declarations. As a result, it is usually desirable to remove the declaration enforcement code when the system is believed to be bug-free and performance becomes more central. This can be done with the variable COMPILEIGNOREDECL.

COMPILEIGNOREDECL [Variable]

Setting the value of the variable COMPILEIGNOREDECL to T (initially NIL) instructs the compiler not to insert declaration enforcement tests in the compiled code. More selective removal can be achieved by setting COMPILEIGNOREDECL to a list of function names. Any function whose name is found on this list is compiled without declaration enforcement.

IGNOREDECL. VAL [File Com]

Declaration enforcement may be suppressed selectively by a file using the IGNOREDECL file package command. If this appears in a file’s file commands, it redefines the value of COMPILEIGNOREDECL to VAL for the compilation of this file only.

Note: The period in the IGNOREDECL file package command is significant. To set COMPILEIGNOREDECL to T, use (IGNOREDECL . T), not (IGNOREDECL T).

## DECLTYPES

A Decl package type, or decltype, specifies a subset of data values to which values of this type are restricted. For example, a “positive number” type might be defined to include only those values that are numbers and greater than zero. A type may also specify how certain operations, such as assignment or binding (see BINDFN), are to be performed on variables declared to be of this type.

The inclusion relations among the sets of values that satisfy the different types define a natural partial ordering on types, bound by the universal type ANY (which all values satisfy) and the empty type

NONE (which no value satisfies). Each type has one or more *supertypes* (each type has at least ANY as a supertype) and one or more *subtypes* (each type has at least NONE as a subtype). This structure is important to the user of Decl as it provides the framework in which new types are defined. Typically, much of the definition of a new type is defaulted, rather than specified explicitly. The definition will be completed by inheriting attributes which are shared by all its immediate supertypes.

An initial set of decltypes that defines the Interlisp built-in data types and a few other commonly used types is provided. Thereafter, new decltypes are created in terms of existing ones using the type expressions described below. For conciseness, such new types can be associated with literal atoms using the function DECLTYPE.

## PREDEFINED TYPES

Some commonly used types, such as the Interlisp built-in data types, are already defined when the Decl package is loaded. These types, indented to show subtype-supertype relations, are:<sup>4</sup>

ANY

|         |         |            |        |
|---------|---------|------------|--------|
| ATOM    | LST     |            |        |
| ARRAYP  | STRINGP | FUNCTION   | STACKP |
| LITATOM | ALIST   | HARRAYP    |        |
| NIL     | LISTP   | READTABLEP |        |
| NUMBERP |         |            |        |
| FIXP    |         |            |        |
| LARGE P |         |            |        |
| SMALLP  |         |            |        |
| FLOATP  |         |            |        |

NONE

Note that the definition of LST causes NIL to have multiple supertypes, i.e., LITATOM and LST, reflecting the duality of NIL as an atom and a (degenerate) list.

In addition, declarations made using the Record package also define types that are attached as subtypes to an appropriate existing type (e.g., a TYPERECORD declaration defines a subtype of LISTP, a DATATYPE declaration a subtype of ANY, etc.) and may be used directly in declaration contexts.

## TYPE EXPRESSIONS

Type expressions provide convenient ways for defining new types in terms of modifications to, or compositions of one or more existing types.

(MEMQ VALUE1. . VALUE N)

[Type Expression]

Specifies a type whose values can be any one of the fixed set of elements VALUE 1. . VALUE N. For example, the status of a device might be represented by a datum restricted to the values BUSY and FREE. Such a "device status" type could be defined via (MEMQ BUSY FREE). The new type will be a subtype of the narrowest type that all of the alternatives satisfy (e.g., the "device status" type would be a subtype of LITATOM). The membership test uses EQ if this supertype is a LITATOM; EQUAL otherwise. Thus, lists, floating point numbers, etc., can be included in the set of alternatives.



(ONEOF TYPE 1 . . TYPE N)

[Type Expression]

Specifies a type that is the union of two or more other types. For example, the notion of a possibly degenerate list is something that is either LISTP or NIL. Such a type can be (and the built-in type LST in fact is) defined simply as (ONEOF NIL LISTP). A union data type becomes a supertype of all of the alternative types specified in the ONEOF expression, and a subtype of their lowest common supertype. The type properties of a union type are taken from its alternative types if they all agree, otherwise from the supertype.

(ALLOF TYPE 1 . . TYPE N)

[Type Expression]

Specifies a type that is the intersection of two or more other types. For example, a variable may be required to satisfy both FIXP and also some type that is defined as (NUMBERP (SATISFIES PREDICATE)). The latter type will admit numbers that are not FIXP, i.e., floating point numbers; the former does not include PREDICATE. Both restrictions can be obtained by using the type (ALLOF (NUMBERP (SATISFIES PREDICATE)) FIXP).<sup>5</sup>

(OF AGGREGATE OF ELEMENT)

[Type Expression]

Specifies *DECLaggregate*, a type that is an aggregate of values of some other type (e.g., list of numbers, array of strings, etc.). AGGREGATE must be a type that provides an EVERYFN property. The EVERYFN is used to apply an arbitrary function to each of the elements of a datum of the aggregate type, and check whether the result is non-NIL for each element. ELEMENT may be any type expression. For example, the type "list of either strings or atoms" can be defined as (LISTP OF (ONEOF STRINGP ATOM)). The type test for the new type will consist of applying the type test for ELEMENT to each element of the aggregate type using the EVERYFN property. The new type will be a subtype of its aggregate type.<sup>6</sup>

(SATISFIES TYPE (SATISFIES FORM 1 . . FORM N))

[Type Expression]

Specifies a type whose values are a subset of the values of an existing type. The type test for the new type will first check that the base type is satisfied, i.e., that the object is a member of TYPE, and then evaluate FORM 1 . . FORM N. If each form returns a non-NIL value, the type is satisfied.

The value that is being tested may be referred to in FORM 1 . . FORM N by either (a) the variable name if the type expression appears in a binding context such as DLAMBDA or DPROG, (b) the distinguished atom ELT for a SATISFIES clause on the elements of an aggregate type, or (c) the distinguished atom VALUE, when the type expression is used in a context where no name is available (e.g., a RETURNS declaration). For example, one might declare the program variable A to be a negative integer via (FIXP (SATISFIES (MINUSP A))) or declare the value of a DLAMBDA to be of type ((ONEOF FIXP FLOATP) (SATISFIES (GREATERP VALUE 25))).

Note that more than one SATISFIES clause may appear in a single type expression attached to different alternatives in a ONEOF type expression, or attached to both the elements and the overall structure of an aggregate. For example,

```
[LISTP OF [FIXP (SATISFIES (ILEQ ELT (CAR VALUE)
(SATISFIES (ILESSP (LENGTH VALUE) 7]
```

specifies a list of less than seven integers each of which is no greater than the first element of the list.

(SHARED TYPE)

[Type Expression]

Specifies DECLshared, a subtype of TYPE, with default binding behavior, i.e., the binding function (see BINDFN), if any, will be suppressed.<sup>7</sup> For example, if the type FLOATP were redefined so that DLAMBDA and DPROG bindings of variables that were declared to be FLOATP copied their initial values (e.g., to allow SETNs to be free of side effects), then variables declared (SHARED FLOATP) would be initialized in the normal fashion, without copying their initial values.

**NAMED TYPES**

Although type expressions can be used in any declaration context, it is often desirable to save the definition of a new type if it is to be used frequently, or if a more complex specification of its behavior is to be given than is convenient in an expression. The ability to define a named type is provided by the function DECLTYPE.

(DECLTYPE *TYPENAME TYPE PROP1 VAL1*  
*PROPN VALN*)

[Function]

NLambda, nospread function. *TYPENAME* is a literal atom, *TYPE* is either the name of an existing type or a type expression, and *PROP 1, VAL 1...PROP N, VAL N* is a specification (in property list format) of other attributes of the type. DECLTYPE derives a type from *TYPE*, associates it with *TYPENAME*, and then defines any properties specified with the values given.

The following properties are interpreted by the Decl package.<sup>8</sup> Each of these properties can have as its value either a function name or a LAMBDA expression.

TESTFN

[Property]

will be used by the Decl package to test whether a given value satisfies this type. The type is considered satisfied if *FN* applied to the item is non-NIL. For example, one might define the type INTEGER with TESTFN FIXP.<sup>9</sup>

EVERYFN

[Property]

EVERYFN specifies a mapping function that can apply a functional argument to each “element” of an instance of this type, and which will return NIL unless the result of every such application was non-NIL. *FN* must be a function of two arguments: the aggregate and the function to be applied. For example, the EVERYFN for the built-in type LISTP is EVERY. The Decl package uses the EVERYFN property of the aggregate type to construct a type test for aggregate type expressions. In fact, it is the presence of an EVERYFN property that allows a type to be used as an aggregate type.<sup>10</sup>

BINDFN

[Property]

BINDFN is used to compute from the initial value supplied for a DLAMBDA or DPROG variable of this type, the value to which the variable will actually be initialized. *FN* must be a function of one argument that will be applied to the initial value, and which should produce another value which is to be used to make the binding.<sup>11</sup> For example, a BINDFN could be used to bind variables of some type so that new bindings are copies of the initial value. Thus, if FLOATP were given the BINDFN FPLUS, any variable declared FLOATP would be initialized with a new floating box, rather than sharing with that of the original initial value.<sup>12</sup>

SETFN

[Property]

is used for performing a SETQ or SETQQ of variables of this type. *FN* is a function of two arguments, the name of the variable and its new value. A SETFN is typically used to avoid the allocation of storage for intermediate results. Note that the SETFN is not the mechanism for the enforcement of type compatibility, which is checked *after* the assignment has taken place. Also note that not all functions that can change values are affected: in particular, SET and SETN are not.

**MANIPULATING NAMED TYPES**

DECLTYPES is a file package type. Thus all of the operations relating to file package types, e.g., GETDEF, PUTDEF, EDITDEF, DELDEF, SHOWDEF, etc., can be performed on decltypes.<sup>13</sup>

The file package command, DECLTYPES, is provided to dump named decltypes symbolically. They will be written as a series of DECLTYPE forms that will specify only those fields that differ from the corresponding field of their supertype(s). If the type depends on any unnamed types, those types will

be dumped (as a compound type expression), continuing up the supertype chain until a named type is found. Care should be exercised to ensure that enough of the named type context is dumped to allow the type definition to remain meaningful.

The functions GETDECLTYPEPROP and SETDECLTYPEPROP, defined analogously to the property list functions for atoms, allow the manipulation of the properties of named types. Setting a property to NIL with SETDECLTYPEPROP removes it from the type.

## RELATIONS BETWEEN TYPES

The notion of equivalence of two types is not well defined. However, type equivalence is rarely of interest. What is of interest is type *inclusion*, i.e., whether one type is a supertype or subtype of another. The predicate COVERS can be used to determine whether the values of one type include those of another.

(COVERS *HI LO*) [Function]

COVERS is T if *HI* can be found on some (possibly empty) supertype chain of *LO*; else NIL. Thus, (COVERS 'FIXP (DECLOF 4)) = T, even though the DECLTYPE of four is SMALLP, not FIXP. The extremal cases are the obvious identities:

(COVERS 'ANY ANYTYPE) = (COVERS ANYTYPE 'NONE) = (COVERS X ) for any type X = T.

COVERS allows declaration-based transformations of a form that depend on elements of the form being of a certain type to express their applicability conditions in terms of the weakest type to which they apply, without explicit concern for other types that may be subtypes of it. For example, if a particular transformation is to be applied whenever an element is of type NUMBERP, the program that applies that transformation does not have to check whether the element is of type SMALLP, LARGE P, FIXP, FLOATP, etc., but can simply ask whether NUMBERP COVERS the type of that element.

The elementary relations among the types, out of which arbitrary traversals of the type space can be constructed, are made available via:

(SUBTYPE TYPE) [Function]

Returns the list of types that are *immediate* subtypes of TYPE.

(SUPERTYPES TYPE) [Function]

Returns the list of types that are *immediate* supertypes of TYPE.

## THE DECLARATION DATA BASE

One of the primary uses of type declarations is to provide information that other systems can use to interpret or optimize code. For example, one might choose to write all arithmetic operations in terms of general functions like PLUS and TIMES and then use variable declarations to substitute more efficient, special-purpose code at compile time based on the types of the operands. To this end, a data base of declarations is made available by the Decl package to support these operations.

(DECLOF FORM) [Function]

Returns the type of FORM in the current declaration context. If FORM is an atom, DECLOF will look up that atom directly in its data base of current declarations. Otherwise, DECLOF will look on the property list of (CAR FORM) for a DECLOF property, as described below. If there is no DECLOF property, DECLOF will check if (CAR FORM) is one of a large set of functions of known result type (e.g., the arithmetic functions). Failing that, if (CAR FORM) has a MACRO property, DECLOF will apply itself to the result of expanding (with EXPANDMACRO), the macro definition. Finally, if FORM is a Lisp program element that DECLOF "understands" (e.g., a COND, PROG, SELECTQ, etc.), DECLOF applies itself recursively to the part(s) of the contained form which will be returned as value.<sup>14</sup>

## DECLOF

[Property]

Allows the specification of the type of the values returned by a particular function. The value of the DECLOF property can be either a type, i.e., a type name or a type expression, or a list of the form (FUNCTION *FN*), where *FN* is a function object. *FN* will be applied (by DECLOF) to the form whose CAR has this DECLOF property on its property list. The value of this function application will then be considered to be the type of the form.

As an example of how declarations can be used to automatically generate more efficient code, consider an arithmetic package. Declarations of numeric variables could be used to guide code generation to avoid the inefficiencies of Interlisp's handling of arithmetic values. Not only could the generic arithmetic functions be automatically specialized, as suggested above, but by redefining the BINDFN and the SETFN properties for the types FLOATP and LARGE to reuse storage in the appropriate contexts (i.e., when the new value can be determined to be of the appropriate type), tremendous economies could be realized by not allocating storage to intermediate results that must later be reclaimed by the garbage collector. The Decl package has been used as the basis for several such code optimizing systems.

**DECLARATIONS AND MASTERSCOPE**

The Decl package notifies MASTERSCOPE about type declarations and defines a new MASTERSCOPE relation, TYPE, which depends on declarations. Thus, the user can ask questions such as "WHO USES MUMBLE AS A TYPE?," "DOES FOO USE FIXP AS A TYPE?," and so on.

**END NOTES**

1. USEDIN is mainly for documentation purposes, since there is no way for such a restriction to be enforced.
2. Like USEDIN declarations, FREE and BOUNDIN declarations cannot be checked, and are for documentation purposes only.
3. With this exception, assignments to variables from within the break are not considered to be in the scope of the declarations that were in effect when the break took place and so are not checked.
4. LST is defined as either LISTP or NIL. i.e., a list or NIL. The name LST is used because the name LIST is treated specially by CLISP. A LIST is defined as either NIL or a list of elements each of which is of type LISTP.
5. When a value is tested, the component type tests are applied from left to right.
6. The built-in aggregate types are ARRAP, LISTP, LST, and STRINGP (and their subtypes).
7. As no predefined type has a binding function, this is of no concern until the user defines or redefines a type to have a binding function.
8. Actually, any property can be attached to a type, and will be available for use by user functions via the function GETDECLTYPEPROP.
9. Typically, the TESTFN for a type is derived from its type expression, rather than specified explicitly. The ability to specify the TESTFN is provided for those cases where a predicate is available that is much more efficient than that which would be derived from the type expression. For example, the type SMALLP is defined to have the function SMALLP as its TESTFN, rather than (LAMBDA(DATUM) (AND(NUMBERP DATUM)(FIXP DATUM) (SMALLP DATUM))) as would be derived from the subtype structure.
10. Note that a type's EVERYFN is not used in type tests for that type, but only in type tests for types defined by OF expressions that used this type as the aggregate type. For example, EVERY is not

used in defining whether some value satisfies the type LISTP. The Decl package never applies the EVERYFN of a type to a value without first verifying that the value satisfies that type.

11. For a PPROG binding, FN will be applied to no arguments if the initial value is lexically NIL.
12. The BINDFN, if any, associated with a type may be suppressed in a declaration context by creating a subtype with the type-expressing operator SHARED.
13. Deleting a named type could possibly invalidate other type definitions that have the named type as a subtype or supertype. Consequently, the deleted type is simply unnamed and left in the type space as long as it is needed.
14. "The current declaration context" is defined by the environment at the time that DECLOF is called. Code-reading systems, such as the compiler and the interpreter, keep track of the lexical scope within which they are currently operating, in particular, which declarations are in effect. Note that (currently) DECLOF does *not* have access to any global data base of declarations. For example, DECLOF does not have information available about the types of arguments of, or the value returned by, a particular function, unless it is currently "inside" that function. However, the DECLOF property can be used to inform DECLOF of the type of the value returned by a particular function.

---

---

## DEFAULTSUBITEMFN

---

---

By: Nick Briggs (Briggs.pa@Xerox.com)

The DEFAULTSUBITEMFN module redefines the DEFAULTSUBITEMFN to permit an extended specification of menu subitems. If the CAR of the 4th element of a menu item is the keyword EVAL, the CADR of that 4th element is evaluated and the results used as the subitem specifications. During the evaluation the variables MENU and ITEM are bound respectively to the menu and item of which the EVAL subitem spec is a part. This module is only a stopgap measure until it is possible to easily redefine the BackgroundMenu subitem function, but it will provide this facility on all menus that do not explicitly specify a subitem function.

example menu item entries:

```
(foo foo.selected "No help for you!" (EVAL dynamic.foo.subitems))
```

using a variable containing subitems, or

```
(foo foo.selected "No help for you!" (EVAL (compute.foo.subitems)))
```

using a function to recompute the subitems.

It is prudent to make the expressions used in the EVAL subitems quite efficient, since they will be called many times.

DEMO -- utilities for running demos / tutorials in Medley

includes

```
 OPEN-URL (URL)
 (rename of ShellBrowse)
 MEDLEY-CONTRIB (REPO)
 shows GitHub contributors to given repo
 uses ShellBrowse
 BKSYSOBJ(string)
```

DEMO-\*.TEDIT

```
 contains scripts / TEDIT file talks
 add your own
```

BKSYSOBJ is the start of a facility

```
(TEDIT.INSERT.OBJ (BKSYSOBJ "(CONS NIL)" (TEXTSTREAM(WHICHW)]
```

You should see (CONS NIL) in the TEDIT stream, clicking should shove (CONS NIL into an Interlisp exec, waiting for ) or ]. (probably the image objectg should be shaded, may also have to set the RDTBL flag on BKSYSBUF for strings, but this is a start).

# **Medley Interlisp**

**March 2023**

**Interlisp.org**

- DEMO-OVERVIEW (this file)
- DEMO-FEATURES (maybe other DEMO-FEATURE-XXX )
- DEMO-PROJECT (what have we been doing / demos)



## What is Medley Interlisp?

- The software for the Xerox Lisp machines, developed from 1960's (BBN), thru 1970's and 80's (Xerox), and 90's (Envos, Venue)
- ACM Software System Award 1992

*The features of **structure editing**, **source code management**, **code analysis** and **cross-referencing** combined to support **rapid incremental development**. The 1992 ACM Software System Award was awarded to the Interlisp system for pioneering work in programming environments.*

- Then: expensive, slow, unwieldy, and ... unavailable
- Now: on modern hardware of all sizes, 1000 times faster

---

## Medley Interlisp is an IDE

*In Interlisp, you could refer to a function you hadn't written yet, run your code until it broke, and from the break have it pop open an editor with the signature of your function. You could also inspect the values of the arguments passed. You could then write the function and continue the computation.*

*Similarly if your code broke with an error, you could edit the function on the stack to correct the error, and continue the computation. ...*

*When demonstrating this capability in Interlisp, the response was along the lines of "why would I ever want that?" ... If you've never used an environment focused on programmer productivity you have no idea how to even think along those lines. -- Simon Brooke*

**Medley Interlisp**

**Features Demos**

---

## A Residential Programming Environment supports Incremental Development

*A residential programming environment is one in which you define, edit, debug in the live environment; write out to files to save for future session. The system tracks what changed. Multiple features work together, to act as a "programmer's assistant":*

*DWIM - Do What I Mean, Spelling Correction: DWIM is more than just spelling correction; it's more an attitude -- not to signal errors if what the user meant is obvious and correctable.*

```
(DEFINEQ (FOO1 (X) (PLUSS X X] <-- PLUS misspelled.
 < Note []

superparenthesis.
PP FOO1
(FOO1 6]
Y
PP FOO1
```

*History and UNDO: Interlisp was the first known system. DWIM changes are undoable.*

```
??
UNDO <dwim event>
PP* FOO1
```

*- The file manager tracks what's been defined or changed. The file "coms" are like a manifest -determines what goes where.*

```
FILES?)
```

Y FILEB  
 (CLEANUP)  
 SEE FILEB

- *Helpsys and DInfo (also a first) interactive documentation*

MAN INFILE  
 MAN CL:WITH-OPEN-FILE  
 MAN CLHS.OPENER  
 <right click DInfo>

- *Masterscope (interactive cross reference) has an extensible english-like language for querying*

LOAD(HELPSYS PROP)  
 . ANALYZE ON HELPSYS  
 . SHOW WHERE ANY CALLS CLHS.OPENER  
 USE EDIT FOR SHOW  
 . SHOW PATHS FROM HELPSYS

---

# Common Lisp and Interlisp Integration

an Inter-medley of Uncommon Lisps

- freely intermix CL and IL function, macro, variable definitions
- All datatypes are common: lists, NIL, symbols, arrays, strings, structures, numbers
- many functions and special forms are the same: CAR, COND, ED
- Some are slightly different: CL:EVAL vs. IL:EVAL, CL:LAMBDA vs. IL:LAMBDA, ...
- Extend CL to include IL and IL to include CL
- Common Lisp definitions and declarations managed by Interlisp environment

*Demo:*

```
. SHOW WHERE ANY ON HELPSYS CALLS CL:WHEN
USE EDIT FOR SHOW
```

---

## Structure editors

`EDITMODE(TELETYPE)` original structure editor: powerful, programmable, short commands (BO 3) P (SW 1 2). Useful for editing huge list structures or Lisp functions.

`EDITMODE(DEDIT)` (Load it to try). Like the TTY editor with menu and showing the results in a window.

`EDITMODE(SEDIT:SEDIT)` Default editor. SEdit maintains illusion that you are editing structure. Parentheses are balanced at all times, but you can just type and backspace. Keyboard controls and attached menu.

Common to all: user never counts parentheses, modifies whitespace, or needs to fiddle with line breaks or indentation

---

## The Virtual Machine and OS

- D-machines had microcode to interpret a bytecoded instruction set. Subsequently, the microcode was reimplemented in C (named "maiko").
- "Sysouts" (memory images) can be moved from machine to machine; only maiko needs to have been compiled for each OS and chip architecture.
- **Medley is small** (relatively speaking). Bytecodes are compact. Medley online uses 16MB/user (64MB max). Installed, 256mb max. Time to make new image 15 seconds; restart a saved image "in the blink of an eye".
- Interlisp-D was the **whole operating system**: scheduler, window manager, network, drivers. Now can rely on host OS for device drivers



---

# **GIT: A Repository for Medley Definitions**

## **Conventionally, GIT tracks and compares files**

- Change detection and presentation based on line-editing semantics: Mismatching character sequence => significant difference

## **Medley's source files: external archives for structured definitions with metadata**

- Saved and loaded, maybe printed, but never edited
- A given definition can be represented in different (but semantically equivalent) line and character sequences

## **Medley interface to GIT: Definition-based change tracking**

**P(ull)R(equest)C(ompare)** command (demo)

- Retrieve changed files from GIT, find / parse PR vs master alternative definitions, compare as Lisp-structure differences

## **Idea: load and manage definitions @ commit level granularity**

- If function FOO is included in commits C and D, definitions FOO;C and FOO;D are co-resident.
-

# **LispUsers, Library, Internal**

(Favorite LispUsers)

Some other development tools: SPY, File Browser

---

## **Revive Applications using Interlisp**

Notecards - early HyperText

Rooms - Screen / Desktop management

LOOPS - Lisp Object Oriented Programming System (not CLOS)

Truckin' - LOOPS game for teaching "Knowledge Representation"

LFG - Lexical Functional Grammar

.... and others

**Medley Interlisp**

**2023 Project 18 March 2023**

**Interlisp.org**

## **What have we been doing?**

- + Adapting the system to current environments and standards

  - hardware: keyboard, mouse, display, CPU (64bit, little endian)

  - software: standard C, Unicode, Posix, UI Guidelines

- + Lower barriers to entry

- + Demonstrate unique and original features

- + Help with revival of other applications built using system

- + Improve maintainability for future users

- + Gather and update documentation

---

## **Who might use Medley Interlisp?**

- Retrocomputing enthusiasts
- Researchers, IDE tool creators, looking into ideas
- Software and AI historians (present and future)
- Software archivists (SPN, technical infrastructure)
- Developers of new applications

---

## Who is involved?

- + Some of original developers from Xerox PARC (in 70's )
- + Developers of s using Interlisp (recursive revivers)
- Open Source contributors
- Students of computer science and history
- + Friends, enthusiasts, and ... you?

```
(OPEN-URL "https://github.com/orgs/interlisp/people")
(MEDLEY-CONTRIB "medley")
(MEDLEY-CONTRIB "maiko")
(MEDLEY-CONTRIB "online")
(MEDLEY-CONTRIB "Interlisp.github.io")
```

---

## More on Modernizing Medley

### Goals

- Adapt Medley UI to current hardware
- Make Medley work more like current applications for comfort

**Mouse:** Wheel Scroll, Window move on title, Window resize on corner. Warning: Exec windows. Consolation: you can do it all with menus. Reduction: some menus are invoked by "middle" mouse button which on some mice is hard to press.

**Keyboard:** Desired state: compatible keystrokes. Subtasks: decide what keystrokes do what; slash through umpteen layers of keyboard re-interpretation.

**Display:** Color, high-resolution displays, modern fonts.



---

## **Reduce Barriers to Entry**

### **- Running Online**

```
OPEN-URL("https://www.youtube.com/watch?v=mI3Ga5LyI1I")
```

```
OPEN-URL("https://online.interlisp.org")
```

### **- Installation Instructions**

```
OPEN-URL("https://interlisp.org/running/")
```

---

## More notes on Running on Modern Systems

- about K&R C and C standards; type declarations, big-endian vs. little endian, 32 bit vs 64 eliminating errors, file-sytem changes, process handles, bugs left over from 24- to 28-bit address space
- docker, installers, CO / CI(?), two levels of virtualization
- [online.interlisp.org](http://online.interlisp.org): connect in seconds.
- Mysteries in the code we inherited, "software archeology"
- Robust "loadup process", all day vs seconds
- bytecode virtual machine aids portability.

```
fb {li}<maiko>bin/makefile-*. *-x
```

Linux, Mac, Windows (WSL1, WSL2, Cygwin)

Docker, FreeBSD, OpenBSD, SunOS5  
x86\_64, i386, arm7l, arm64, and older

---

## Getting Involved

- *Try things out, report new problems, or new reproducible cases*
- *Clean up old issues*

(OPEN-URL "https://github.com/interlisp/medley/issues")

(FILESLOAD PICK)

PICK ISSUE

- *Network emulation*
- *Donate? See our GitHub sponsor page*

---

## DIALPHONE

---

By: Michel Denber (Denber.WBST@Xerox.COM)

### INTRODUCTION

DIALPHONE is a simple computer-controlled telephone dialer. It requires a modem connected to the RS-232 port. It should work with a wide variety of modems since it makes no assumption about any return codes from the modem. It lets you dial your phone by typing (or selecting) a number from the computer's keyboard. A history list of recently dialed numbers is maintained. It knows how to deal with differences between dialing local extensions, outside calls, and long distance. It also translates phone numbers containing letters back into numbers.

### OPERATION

Load DIALPHONE.LCOM from your local LispUsers directory and call (DIALPHONE). The program will ask you for your own phone number. This is used within Xerox for billing long distance calls. If you just enter a CR, the program will not append any number at the "quick-quick-quick-slow" tone when dialing long distance. Next, it will try to attach a menu with a telephone icon to an AddressBook window (if you have the LispUsers package AddressBook loaded), since it makes a useful complement to AddressBook. If you do not have AddressBook loaded, it will prompt you to click in some window where you would like the Dial icon attached. If you click in an area of background, a stand-alone menu will be created. Before dialing a number, check the various parameters described below.

To dial a number, click on the phone icon with the left mouse button. You will see the prompt "Number please:" in the prompt window. You may now either type in a number or shift-select it out of another window. To abort the operation, type a CR without a number. If you select the icon with the middle button, you will get a menu of the last 10 numbers dialed. Listen to the number being dialed over the modem's speaker, then pick up the receiver. The program will make the modem hang up automatically shortly after the number is dialed.

The program converts any letters in the number you give it into numbers. The program is designed for use on a PBX-style system (although it is easy to modify if desired). Numbers less than six digits long are assumed to be internal extensions and are dialed exactly as given. Numbers of 7 or 8 digits are assumed to be local outside calls; the program prefixes a "9" to them. Numbers longer than 8 digits are assumed to be long distance; the program appends your extension to them for billing. Xerox Intelnet numbers should be typed in their usual form, starting with "8\*". Area codes and exchanges should be separated with dashes, e.g. "716-555-1212".

You can also call the program directly, e.g. (DIALPHONE "555-1212"). It will return the number dialed. If you try to dial a number while the modem port is already in use, the program will print an error message in the Prompt Window and return without dialing.

DIALHISTSIZE

[Variable]

This controls how many numbers will appear in the menu you get when you select the phone icon with the middle button. Default = 10.

DIALPREFIX [Variable]

The modem command your modem needs to initiate dialing. Default = "ATDT". Change this to "ATDP" if you do not have touch-tone service.

DIALSUFFIX [Variable]

The modem command your modem needs to terminate a command. Default = <CR> (ASCII 13).

LASTNUMBERDIALED [Variable]

The last number you dialed. initially NIL.

PHONEBILLNUMBER [Variable]

The number the program should use to append to long-distance numbers. Initially "" (null string).

---

---

## DICTTOOL

---

---

By: Maxwell (Maxwell.pa@Xerox.com)

Uses DICTCLIENT, ANALYZER

INTERNAL

### INTRODUCTION

DICTTOOL is the user's interface to the Dictionary Server. The Dictionary Server is a prototype of a shared network resource for providing a suite of dictionary-based capabilities to programs running on client workstations. It has on it the American Heritage dictionary, the Word Finder synonym package by Microlytics, a Proofreader, and the WordNerd, a package for searching for words based on their meaning.

(Note: The American Heritage dictionary has been licensed to us by Houghton-Mifflin for research purposes only, and so we have not made the Dictionary Server generally available. The Dictionary Server should only be used by people within PARC.)

### HOW TO USE DICTTOOL

When you load the DICTTOOL, it automatically adds a new menu item named "Dictionary" to the TEdit menu and the Background menu. The "Dictionary" menu item has three sub-items: "Get Definition", "Get Synonyms", and "Search For Word". Here is how each one works:

#### Get Definition

If you make a selection in a TEdit document, and then invoke the "Get Definition" command in that document, then DictTool will ask for a confirmation and then fetch the definition for that word from the Dictionary Server, printing it in a separate window. If there is more than one entry in the American Heritage Dictionary for that word, then it will print the definitions one after another. The Dictionary Server knows how to find the root forms of words, and so "breathing" "breathes" and "breathe" will all give you the same entry.

If there is no selection in the TEdit document, or if you deny the confirmation, or if instead of using the TEdit menu you use the Background menu, then DictTool will first prompt you for a word to look up and then fetch its definition. (Since it is very hard to make a null selection with TEdit, DictTool treats a one character selection as meaning "no selection". If you really want to look up a single letter in the dictionary, you can type it in when prompted for a word.)

If you want to look up several definitions at once, separate the entries with semi-colons followed by spaces. (i.e. "camera; photography; motion picture"). Semi-colons are used as delimiters because some of the entries in the American Heritage Dictionary have spaces and commas in them (as in "motion picture"). It also makes it easier to look up words in the output of the WordNerd (see "Search For Word" below).

### Get Synonyms

The interface for getting synonyms is exactly the same as the interface for getting definitions. If you make a selection, then DictTool will first confirm the selection and then print out the synonyms in the same window that the definitions are printed in. If you don't make a selection, then DictTool will first prompt you for a word. The format of the information printed out is a series of synonym classes separated by carriage returns. Each synonym class begins with the part of speech that its elements belong to. The elements themselves are separated by commas.

### Search For Word

The interface to the WordNerd is a little different from the other interfaces. Instead of typing just one word in, you want to type a list of keywords separated by spaces. For instance, if you were looking for the word for a mechanical model of the solar system, you might type:

*Type keywords to search on:* mechanical model solar system

The WordNerd then searches for words that have at least two of these keywords in their definitions. The results would be sorted according to the number of keywords found, with the words having the most keywords printed first:

**mechanical model solar system:** orrery

**mechanical solar system:** mechanism

**model solar system:** planetarium

**mechanical system:** automation; bar1; component; degree of freedom; energy level; hookup; ignition; instrument; key1; linkwork; load; machine; neutral; perpetual motion; quantize; resonance; schematic; servomechanism; shafting; stress; suspension; unit

**solar system:** Copernican; cosmic; Earth; Ganymede3; Jupiter2; Mars2; mercury; Milky Way; nebular hypothesis; Neptune3; Pallas; planet; planetesimal hypothesis; Pluto2; Saturn2; solar battery; space; sun; Uranus2; Venus2; Vesta2

(The numbers after some of the words mean that this is the nth entry of this word in the American Heritage Dictionary.) If there is a word in the list that you want to see the definition for, you can merely select it and get its definition with "Get Definition". In this case you would probably want to know what "orrery" means:

**or|re|ry** *n., pl. -ries*. A mechanical model of the solar system. [After Charles Boyle (1676–1731), fourth Earl of Orrery, for whom one was made.]

There is also a mechanism for indicating that two words are synonyms of one another, and hence should not be counted as separate keywords for the purpose of deciding whether a word has the minimum two keywords. All you need to do is put parentheses around the words in question. For instance if you were looking for the word for the little plastic thing on the end of a shoe lace, you might try:

*Type keywords to search on: (shoe lace shoelace) (end tip)*

And get in return:

**shoe+ end+:** aglet; fall; heel1; lift; point; quarter; spike1; toe

(A plus at the end of a word indicates a synonym class.)

If you only give the DictTool one word, then it will print out all of the words in the dictionary that have that word in its definition.

### Max Words

There are two sub-items in the "Search For Word" sub-menu: "Max Words" and "Min Keywords". The first sub-item, "Max Words", allows the user to specify the maximum number of words that should be returned on each search. DictTool is set up to only return 100 words at a time. If WordNerd finds more than a hundred words, then it truncates the list and indicates how many words it eliminated. If you want to see the words that were eliminated, just make the same request with the same keywords in the same order and the WordNerd will return the next 100 words. (If there is no selection in the document, then DictTool will prompt you with the last set of keywords so that this is easier.) However, if 100 words is too many or too few, you can change it with this menu item or by setting the global variable DictTool.MaxWords.

### Min Keywords

DictTool is set up to only return a word if it has at least two of the user's keywords in its definition. If the user wants, he can raise or lower the minimum as he sees fit. The minimum only comes into play whenever the user gives more keywords than the minimum, otherwise the WordNerd looks for words that have at least one of the keywords in their definition. A minimum of 1 means that only one word has to match. A negative minimum means that the WordNerd will set the minimum relative to the number of keywords given. For instance, a minimum of -1 says that all but one of the keywords have to match for the word to be returned. A minimum of zero means that *all* of the words have to match.

### Search For Phrase



The Search For Phrase command returns all of the entries in the American Heritage Dictionary that have a particular phrase in them. It does this with the help of the Search For Word command, which is why it is a sub-command of that command. Whenever you search for a phrase, the dictionary server first uses the Search For Word command to get the list of words in the dictionary that have all of the words of the phrase in it. It then looks up the definition of each of these words, and returns the words that have the phrase in their definition. This can be a very time-consuming operation, so you should use this command sparingly. But if you are concerned about locality and word order, then this command can save you a lot of time.

## **PROOFREADING**

The Dictionary Server also provides proofreading facilities similar to the PROOFREADER package. The interface is exactly the same: there is a "Proofread" menu item on the TEdit menu which produces a special fixed menu for proofreading. The only difference is that all of the proofreading is done remotely on the server. You should only use the Dictionary Server for proofreading small documents; if you are going to do a lot of proofreading, it is better to use the PROOFREADER. (For more documentation on how to proofread, see PROOFREADER.)

---

---

## DIGI-CLOCK

---

---

By: Keith Mountford (Mountford.AISNorth@Xerox.Com)

### INTRODUCTION

DIGI-CLOCK is a digital clock which allows you to keep track of the time in multiple time zones.

### STARTING DIGI-CLOCK

Loading DIGI-CLOCK will kill any existing DIGI-CLOCK process and restart the clock. Once the clock is loaded it can be restarted by typing (DIGI-CLOCK) or (DIGI-CLOCK T). The second of these restarts the clock from scratch, rebuilding everything; the first, simply restarts the process and does not undo any changes made to the clock. Left buttoning in the window causes the clock to update itself. The clock updates itself approximately once a minute.

### CHANGING DIGI-CLOCK

The clock font, the time, the local time zone, the alarm, the alarm mode (loud or quite), the clock mode (12 or 24 hour) are all settable from the middle button menu. This menu also allows you to add clocks for other time zones. The auxilliary clocks also have middle button menus which allow you to set the time zone for that window and edit the time zone heading. The default is for all of the auxilliary clocks have the same font and changing the font in one changes the font in all of them unless the submenu item "Set Aux Clock Font In Just This Window" is slected. Selecting "Shape to Fit" will reshape the clock windows to their minimum size.

If the menu font options are not sufficient you can set the global variables \*DC-FONT\* and \*DC-AUXW-FONT\*. The date format is bound to the variable \*DC-DATEFORMAT\* and can be changed by editing or setting this variable. The clock does not deal with seconds gracefully in 12-hour mode and it will not allow NUMBER.OF.MONTH in 12-hour mode. The regional time zones are stored on the global list \*DC-TIME-ZONE-LIST\*.

### SETTING DIGI-CLOCK

Choosing "Set Time" from the middle button menu, brings up a menu which allows you to set the time.

### SETTING THE DIGI-CLOCK ALARM

DIGI-CLOCK includes an alarm clock which can be set to any number of dates in any order. The alarm stores a brief message to remind you why the alarm was set. To set the alarm choose the "Set Alarm" middle button option. Once you have set the time, the clock will prompt you for a message. This message can be longer than the window, but only one line long. When the alarm rings, the window will shape to fit the message.

The alarm calls the function RINGBELLS once a minute until the alarm is turned off, which can be annoying. To run the alarm in quiet mode, select Quiet Alarm from the middle button menu. Selecting Quiet Alarm changes this menu option to Loud Alarm and sets the alarm to run in quiet mode. Selecting Loud Alarm will toggle the alarm back to its original noisy setting.

To unset the alarm, select "Delete Alarm Setting" from the middle-button menu and then select the time you want deleted from the pop-up menu.

To turn the alarm off, select "Turn Alarm Off" from the middle-button menu.

---



---

## DINFO

---



---

By: Doug Cutting (Cutting.PA@Xerox.COM)

Uses: GRAPHER, TEDIT

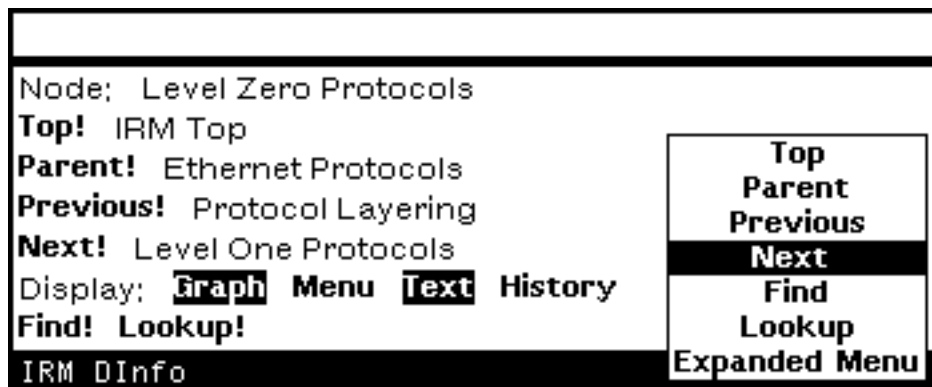
This document last edited on October 7, 1987.

### INTRODUCTION

DInfo is a system for browsing graph structured documentation. Graphs for the *Interlisp-D Reference Manual* and the *Xerox Quintus Prolog Manual* are available, as are tools for creating and editing new graphs.

### USER INTERFACE

Selecting DInfo from the background menu will pop up a menu listing the available graphs (see DINFO.GRAPHs below). Selecting one of these items will bring up a browser on that graph. Most interaction with DInfo is done through menus at the top of each graph browser which look like this:



The Expanded Menu may be closed independently of the graph browser, and re-opened by selecting **Expanded Menu** from the title bar menu. Except for this command, the commands in the title bar menu are identical to their counterparts in the Expanded Menu, so only the latter commands are documented.

Next to **Node:**, **Top!**, **Parent!**, **Previous!**, and **Next!** are printed the names of the node currently being visited, the name of the top node in the graph, the name of the node previous to the current node, and the name of the node next to the previous node, respectively. **Top!**, **Parent!**, **Previous!**, and **Next!** are also commands as follows:

**Top!** [DInfo Menu Command]

Visits the top node in the graph.

**Parent!** [DInfo Menu Command]

Visits the parent of the current node.

**Previous!**

[DInfo Menu Command]

Visits the node previous to the current node.

**Next!**

[DInfo Menu Command]

Visits the node following the current node.

**Previous!** and **Next!** thus provide sequential access to the graph.

The Display: toggles control what will be displayed when a node is visited:

**Graph**

[DInfo Menu Toggle]

toggles display of a Grapher display of the graph local to the current node. Selecting a node in this display will visit the corresponding node in the graph.

**Menu**

[DInfo Menu Toggle]

Toggles display of a menu of subnodes of the current node. If the current node has no subnodes no menu will be displayed. Selecting an item in the subnode Menu will visit that node in the graph.

**Text**

[DInfo Menu Toggle]

Toggles display of the text of the current node. Turning this off will speed up the visiting of nodes considerably, useful when searching for a particular node.

**History**

[DInfo Menu Toggle]

Toggles display of a menu containing the history of nodes visited. Selecting an item from this menu will revisit that node.

**Lookup!**

[DInfo Menu Command]

If selected with the left mouse button prompts for a term and then calls the LOOKUPFN of the graph with it. Using the middle button will re-call the LOOKUPFN with whatever was used previously. This is intended for the lookup of terms in a graph-dependent index. In the case of the *Interlisp-D Reference Manual* DInfo Graph, Lookup! will lookup a term in the index of the IRM, and then visit the node containing the reference to this term.

(DINFO GRAPH.OR.FILE WINDOW.OR.REGION — —)

[Function]

Starts a DInfo browser on GRAPH.OR.FILE in WINDOW.OR.REGION. If GRAPH.OR.FILE is NIL, an empty graph will be created. If WINDOW.OR.REGION is NIL, the user will be prompted for a window or region.

**GLOBAL VARIABLES**

DINFOMODES

[Variable]

Determines which of the toggles will be selected when DInfo is initially started; it should be a list with recognized members being GRAPH, MENU, TEXT, and HISTORY. Default is (GRAPH TEXT).

DINFO.HISTORY.LENGTH

[Variable]

Determines the maximum length of DInfo's history. Default is 20.

DINFO.GRAPHS

[Variable]

Determines the contents of the menu raised by selecting DInfo from the background menu. Should be a menu-items style list where when the CADR of an item is evaluated it returns either a Dinfo Graph (a DINFOGRAPH record) or the name of a file containing a Dinfo Graph (i.e, something suitable for passing to DInfo as the GRAPH.OR.FILE argument).

**INTERNALS**

The following information is included for the programmer interested in adding alternate graphs to DInfo.

**DINFONODE****[Record]**

Contains the following fields:

**ID** Unique identifier for node in graph, ala GRAPHNODE field NODEID. Note that EQ is used for checking identity of nodes.

**LABEL** The print name of a node. Analagous to the GRAPHNODE field NODELABEL.

**FILE** The file containing the documentation for this node. Should not generally include HOST and DIRECTORY fields as DInfo will default these (assuming all documentation files are on one directory, see below).

**FROMBYTE** Byte number in FILE where the documentation for this node begins.

**TOBYTE** Byte number in FILE where the documentation for this node ends.

DInfo uses OPENTEXTSTREAM to display its files, and thus any TEdit file can be included. Note that if a file has any formatting (image objects in particular), the byte number of a character in a file is not necessarily the same as the TEdit character number of that character.

**PARENT** The ID field of the node parent to this node.

**CHILDREN** A list of the ID's of the subnodes of this node.

**NEXTNODE** The ID of the next node in the graph.

**PREVIOUSNODE** The ID of the node previous to this node in the graph.

**USERDATA** Unused. Note that there is no special access function for this field as, for example, WINDOWPROP is for the USERDATA field of a WINDOW. This field is left open for use by implementors for whatever they see fit.

**DINFOGRAPH****[Data Type]**

Contains the following fields of interest to the implementor:

**NAME** The name of the graph. Note that when DInfo reads a graph from a file (with DINFO.READ.GRAPH) this field is set to the NAME field of the file name the graph is read from.

**NODELST** The list of nodes in the graph. Each node should be a DINFONODE record.

**TOPNODEID** The ID field of the root, or top node of the graph.

**WINDOW** The main window of the graph browser.

**CURRENTNODE** Used by DInfo to keep track of where in the graph it is.

**DEFAULTHOST** Used if no host is specified in the FILE field of a node.

**DEFAULTDEVICE** Used if no device is specified in the FILE field of a node.

**DEFAULTDIR** Used if no directory is specified in the FILE field of a node.

Note that DEFAULTHOST, DEFAULTDEVICE and DEFAULTDIR are set when a DINFOGRAPH is read from a file (by DINFO.READ.GRAPH) to the host, device and directory of that file.

**TEXTPROPS** Will be passed as the PROPS argument to OPENTEXTSTREAM when the file for a node in the graph is displayed. This feature can be used to fake some formatting .

LOOKUPFN Will be called when the user selects Lookup! from DInfo's Expanded Menu with two arguments: The string to look up, and the current DInfo graph.

MENUFN Called when the middle mouse is depressed in the title bar of a graph's window. If not specified, DINFO.DEFAULT.MENU will be used. Passed one argument of the current DInfo graph. DINFO.EDIT.MENU is a MENUFN that allows editing of DInfo graphs. Selecting >>Empty Graph<< from the menu raised by selecting DInfo from the Background Menu will start DInfo on a an empty graph with this MENUFN.

FREEMENUITEMS DInfo's Expanded Menu is implemented as a FreeMenu. This property holds a list of FreeMenu item descriptions suitable for passing to FREEMENU (see FreeMenu documentation). This list will be appended onto the bottom row of buttons (**Find!** and **Lookup!**) whenever a FreeMenu is created for this graph.

USERDATA Accessed by the macro DINFOGRAPHPROP. See below.

(DINFOGRAPHPROP GRAPH PROP VALUE) [Macro]

If VALUE is not specified, will return the PROP of GRAPH. PROP can be either a real DINFOGRAPH field or something in the USERDATA field. If VALUE is specified it will be put in GRAPH at PROP. Note that in this case it will return the *new value* stored in, not the previous value, as many other Interlisp-D access functions do.

(DINFO.UPDATE GRAPH NODE SEL) [Function]

Will visit NODE in GRAPH. NODE, if specified, should be a DINFONODE record which is in the NODELIST of GRAPH. SEL is used by DInfo's *Interlisp-D Reference Manual* lookup facility, and should be useful in implementing other lookup facilities. SEL determines what in the TEXT of this node will be selected. SEL should be a string or a list of the format (NAME X) where NAME is the name of the selection, X is the character number in the text of NODE to be scrolled to. If SEL is a string, the string will be searched for in the text of NODE and selected. This is useful for the lookup of terms.

(DINFOGRAPH WINDOW) [Function]

Return the DINFOGRAPH associated with window. Note that the pointer from the window to the graph is destroyed when the window is closed to remove circularity. For this reason it is better to keep a handle on the DINFOGRAPH and use (DINFOGRAPHPROP <DInfoGraph> 'WINDOW) when you need the WINDOW.

(DINFO.READ.GRAPH FILE QUIETFLG) [Function]

Reads a file written by DINFO.WRITE.GRAPH, and returns the DINFOGRAPH contained therein. If QUIETFLG is non-NIL, nothing will be printed out while reading.

(DINFO.WRITE.GRAPH GRAPH FILE) [Function]

Writes GRAPH to FILE such that it can be read by DINFO.READ.GRAPH.

(DINFO.READ.KOTO.GRAPH GRAPH FILE) [Function]

Reads a file written by Koto DINFO.WRITE.GRAPH and returns a Lyric DINFOGRAPH. Thus:

```
(DINFO.WRITE.GRAPH
 (DINFO.READ.KOTO.GRAPH <file1> T)
 <file2>)
```

will convert the Koto format DINFOGRAPH in <file1> to a Lyric format DINFOGRAPH in <file2>.

## DIR-TREE

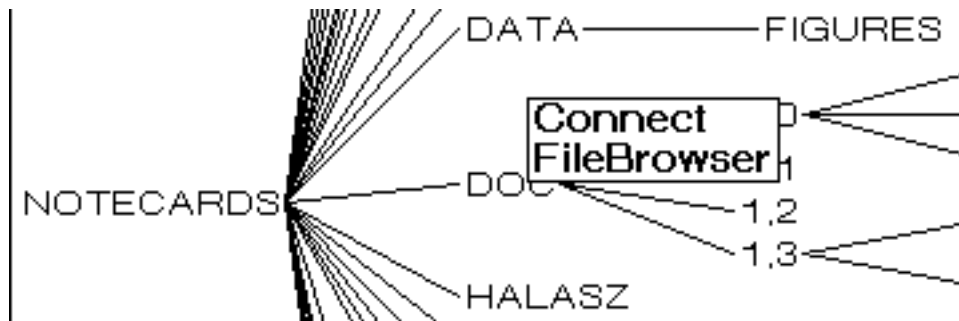
Dir-Tree builds directory trees using the grapher package.

It works on IFS, XNS, and NFS devices from Medley.

Loading Dir-Tree puts a menu option on your background menu.

Dir-Tree can take awhile to compute a large directory. Do not give it a /users directory to graph.

From the directory window you can left button on a node and bring up a menu to Connect to the directory or call FileBrowser on the directory.



At the moment shift-select from the graph does not work but you can achieve the same effect by clicking the middle mouse button on a node.

Holding the left mouse button down in the title bar brings up the recompute option. On NS servers it also allows you to set the Enumeration Depth.



I make no pretenses that Dir-Tree is anything like a finished product. However, it has been satisfactorily working for me in its present state for over a month now (Jan '89).

I have been asked when it will be available. Since I will not have any time to work on it until the NoteCards doc is done, those who are interested are welcome to use it in its present state.

### KNOWN PROBLEMS:

When using it from a Sun DO NOT use the {UNIX} device. This hopelessly confuses Dir-Tree.

When specifying the root directory on a Sun, you must get the case right. Dir-Tree will still generate a graph but it will only go down one level, and it will not correctly truncate the root directory name in the graph window.

Keith Mountford.



---

---

## DOC-OBJECTS

---

---

Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox.com or Koomen@CS.Rochester.edu)

Uses: TEDIT, IMAGEOBJ, DATEFORMAT-EDITOR

This document last edited on October 27, 1987.

### DESCRIPTION

DOC-OBJECTS is a generic, extensible interface for including image objects in TEdit documents. It hooks into TEdit by an extra entry on TEdit's middle button menu, as well as by redefining what happens on typing CTRL-O. Clicking the menu entry or typing CTRL-O brings up an *Objects* menu. Selecting an object causes an instance of the designated object to be inserted in the document at the position of the caret. Clicking outside the Objects menu has no effect. DOC-OBJECTS comes with a set of predefined Document Objects, which are described below. Additional Objects can easily be added to the Objects menu.

### Predefined Objects

#### Time Stamp

[Document Object]

A TimeStamp reflects the date the document containing it was last PUT into a file. Each PUT causes a TimeStamp to be updated. Clicking the middle button over a TimeStamp brings up a DateFormat editor. The TimeStamp can be given any appearance consistent with the function DATEFORMAT (see IRM, Section 12.5). The object following the next colon is a TimeStamp object for this file: 15 Sep 1988 18:11 PDT (Thursday). Individual characters of a TimeStamp cannot be altered by TEdit, but a TimeStamp can be given arbitrary TEdit Looks. The DATEFORMAT-EDITOR package is automatically loaded by the DOC-OBJECTS package.

#### File Stamp

[Document Object]

A FileStamp reflects the name of the file into which the document containing it was last PUT. Each PUT causes a FileStamp to be updated. It cannot be edited. A FileStamp is initially displayed as "-- not yet filed --".

#### Include

[Document Object]

This document object is a dynamic version of the static TEdit Include command, and is intended to facilitate the unbundling of document chapters and sections, while maintaining the ability to print the entire document or any portion of it. When an Include object is created, the user is prompted for a file name. An Include object can be enabled or disabled. If it is enabled, the object shows in the TEdit window as '@Include[MySubFile.TEdit]', and the indicated file will be included during a hardcopy operation. If it is disabled, then the object shows (both in the TEdit window and on hardcopy) as '@DoNotInclude[MySubFile.TEdit]'.

Middle-clicking on an Include object pops up a menu with the following fields: "New File" (prompt for a new file name), "Edit File" (TEdit the Include file, or bring it to the top if it is already being edited), "Enable" (include the file during hardcopy), and "Disable" (do not include the file during hardcopy).

Two caveats:

- 1) For best results, make an Include object the last thing in a paragraph, or put it in a paragraph of its own, and set the line and paragraph leadings to 0. The Include object forces a paragraph break right after the Include object during hardcopy, to prevent the looks of the paragraph containing the Include object to mask the looks of the first paragraph in the file being included.
- 2) A document containing Include objects is best hardcopied from a FileBrowser window, rather than through the hardcopy command on the TEdit window menu. It will work properly either way, but it's a bit unnerving to watch TEdit trying to reflect on the display the inclusion of one or more files before hardcopy and the removal of the included files after hardcopy.

Horizontal Rule

[Document Object]

This provides a more user-friendly interface to the HRULE package (which is automatically loaded by the DOC-OBJECTS package). Upon selecting it a numberpad is brought up repeatedly, with which the user can indicate the thickness of alternating black and white lines. The resulting HRule object is inserted in the document whenever the numberpad is aborted or returns 0. The DOC-OBJECTS package also modifies the HRule object such that it can be edited: clicking the middle button over an HRule object brings up a structure editor (such as SEdit) on a list containing the thicknesses of the lines composing the HRule. This list can be altered in any way, as long as the editor returns another list of numbers (presumably of odd length).

Eval'd Form

[Document Object]

Selecting this object causes a type-in window to pop up. The value of the form typed in is assumed to be an image object. This is what TEdit used to do on typing CTRL-O. For TEdit's purpose, an image object is a Lisp value of type IMAGEOBJ, BITMAP, STRINGP, LITATOM, or REGION. The latter is assumed to refer to a region of the screen.

Screen Snap

[Document Object]

Selecting this object prompts for a region of the screen. A bitmap containing a copy of the given region of the screen is inserted in the document. This is equivalent to clicking the right button in the display's background while holding the SHIFT key down.

### Extending the Document Objects interface

DocObjectsMenuCommands

[Variable]

This variable contains a list of menu items which are displayed in the Document Objects pop-up menu. It is analogous to the variable BackgroundMenuCommands (*cf.* IRM, Section 28.8). The Lisp form in each item is assumed to evaluate to an image object (as defined under Eval'd Form described above).

DocObjectsMenu

[Variable]

This variable caches the Document Objects menu. Set it to NIL whenever you alter the variable DocObjectsMenuCommands.

DocObjectsMenuFont [Variable]

This variable contains a font descriptor which is used for displaying the items in the Document Objects menu. The initial value is (FONTCREATE '(MODERN 12 BOLD)). Set the variable DocObjectsMenuFont to NIL whenever you alter DocObjectsMenuFont.

(DOCOBJ-STRING-IMAGEBOX STRING IMAGESTREAM) [Function]

A useful function for Document Objects that wish to display as a string of characters (such as a TimeStamp). The Document Object's IMAGEBOXFN can call this function to obtain an image box with the TEdit Looks that apply to the Document Object taken into account.

(DOCOBJ-WAIT-MOUSE WINDOWSTREAM) [Function]

A useful function for Document Objects that wish to assure that their buttoneventfn takes action only if the mouse buttons were let up within the Object's region (i.e., the clipping region of the Object's window stream). It returns T when the mouse buttons go up within the region, or NIL when the mouse moves out of the region while a button is still down.

### **Future predefined Document Objects**

Watch this space for objects such as Index & Index Entry, Citation & Bibliography, ...

---

---

**DONZ**

---

---

**The *Excruciatingly* User Friendly Environment**

By: Jeff Shrager et al.

Checked out for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

Files: DONZ. DONZ.dfasl DONZ.TEDIT

**Description:**

Loading DONZ starts a background process (DONZ.RUN) which causes your ICONS to become "user friendly". Telling you what this means would spoil all the fun of discovery.

**Customizing DONZ:**

The delay between activations of DONZ is done by (DISMISS DONZ.DELAY). Its value defaults to 5000 (5 seconds). When DONZ wakes up, it tries out one window, selected at random, from all the windows in (OPENWINDOWS). If the selected window is not an ICON, nothing happens and DONZ wakes up again in 5 more seconds. If that window is an ICON, then DONZ tries to find a message for it as described below. This method results in DONZ's "friendliness rate" running approximately in proportion to the ratio of icons to opened windows on your screen. Thus, if you are doing real work, DONZ won't bug you, but if you just have a screen full of icons, DONZ will be *exceedingly friendly*.

The list DONZ.TEST.MESSAGE.ALIST has the form:

(frob1 frob2 frob3...)

where each frob is of the form:

(testfn msg1 msg2 msg3...)

TESTFN will be called with one argument: the \*MAIN\* window with which this icon is associated. That is, for instance, the TEDIT window that the icon will expand to...NOT the icon window. TESTFN should decide whether or not this window is of a type that it will handle, and return T or NIL as appropriate. When one of the TESTFNs returns T, one of the messages in the tail of the associated frob list will be selected at random and displayed...in the way that they are displayed...you'll see! If none of the frobs accepts responsibility for the present icon, there are a few default messages built into DONZ.

The msgs should be lists containing individual words (appropriately capitalized) so that .PARA in the PRINTOUT can do the appropriate word division.

**Notes:**

DONZ has to be killed via the PSW. Be gentle.

DONZ continues to run during screen idle. This results in fairly funny theatrics on the part of your icons.

The display is done with PRINOUT which does some bogusness to indent, erasing some of the good parts of the window. This is slightly messy, but otherwise innocuous.

Anyone who can guess the origin of the name of this package belongs on the East Coast.

**Acknowledgements:**

Thanks to Ros Chast for originating the idea, Mike Kazar and Dave Nichols for their original implementation of DONZ at CMU.

---

---

## **DORADOCOLOR**

---

---

Maintained By: Frank Shih (Shih.envos@Xerox.com)

INTERNAL

Uses: COLOR

This document last edited on 8-Nov-88.

### **INTRODUCTION**

This package is the Xerox Lisp software driver for the Dorado (Xerox 1132) color display.

### **NECESSARY HARDWARE**

You need a Xerox 1132 with color card and a third party color display. Please contact your Xerox representative for details concerning acquiring and setting up all the required hardware. Some notes on configuring the Conrac color monitor are given at the end of this document.

Assuming you have all the hardware you need, turn it all on. This means

- (1) Your 1132 is running Xerox LISP.
- (2) Your 1132 has an 1132 color card installed.
- (3) Your color display is plugged in and powered on.
- (4) Three cables for red, green, and blue signal connect the 1132 color card to the color display.

Any reconnections should be made while your 1132 is off. Until you issue some software commands, a black color display is normal.

### **DORADOCOLOR SOFTWARE**

The DORADOCOLOR package provides the machine dependent portion of software that is needed to drive your color display assuming you are using an 1132 with 1132 color card. Other than LOADING the DORADOCOLOR package and turning the DORADOCOLOR package on using the function `COLORDISPLAY`, all additional functionality is provided by and documented with the `COLOR` package. There are no DORADOCOLOR functions that the user needs to call directly. The user calls functions described in the `COLOR` documentation. A single exception to these comments is a global variable `\DORADOCOLOR.LEFTMARGIN` which controls the period of time the display controller should wait before turning on the color guns. `\DORADOCOLOR.LEFTMARGIN` is normally set to 56; if this value causes odd results with your monitor, try setting `\DORADOCOLOR.LEFTMARGIN` a little higher or

lower and reinitializing the display. (You can reinitialize the display by calling COLORDISPLAY twice in succession).

Once your hardware is on, you can proceed to issue COLOR commands to your hardware. You should have the DORADOCOLOR package already LOADED from your LIBRARY directory. That is, you've already done something like (FILESLOAD DORADOCOLOR). At this point it may be convenient to follow this documentation along with the documentation for COLOR in the Lisp Library Packages Manual. If you now type

```
(COLORDISPLAY 'ON 'DORADOCOLOR)
```

your display will now change from total black to a color test pattern with horizontal and vertical stripes. The sequence of events is that there should be a noticable flicker on your color display, followed by a white wall covering the color display, followed by the painting of this white wall with horizontal and vertical stripes of color woven together. There are now some simple tests you can do to satisfy yourself that your hardware is working. Here is a small list of things to try:

```
(SETQ CSBM (COLORSCREENBITMAP))
```

```
(BLTSHADE 'WHITE CSBM)
```

```
(BLTSHADE 'RED CSBM)
```

```
(BLTSHADE 'GREEN CSBM)
```

```
(BLTSHADE 'BLUE CSBM)
```

```
(SETQ DS (DSPCREATE CSBM))
```

```
(DRAWLINE 0 0 500 500 10 'REPLACE DS 'YELLOW)
```

```
(DRAWLINE 500 0 0 500 10 'REPLACE DS 'CYAN)
```

Assuming all has gone well to this point, you should now be able to try all the functions described in the COLOR package documentation. The COLORDemo package is a good source of test programs to try — (IL:LOAD 'COLORDemo.LCOM) to get this package. Both COLOR and COLORDemo documentation are in your LispUsers' Manual.

## KNOWN BUGS

As of 11/88, there are several known bugs with the color code. Dragging the mouse off the right hand edge of the display appears to hang the Dorado. Also, color fonts do not seem to work (probably because in Medley AC fonts are unrotated using pilotbtt) .

## APPENDIX - THE CONRAC COLOR MONITOR

This section describes configuring the Conrac color monitor (model 7211C19) for use with the Dorado.

**Back Panel** - Connections need to be made as follows: each color cable (red, green, blue) should be connected to the color-corresponding IN terminal. The black cable should be connected to the SYNC/HDRIVE IN terminal. The rocker switches next to the connectors should be set to 75 ohms.

**Front Panel** - The BRIGHT and CONTRast knobs push in to configure to their preset settings, pull out for adjustment. The SCREEN buttons should be toggled out, the CHANNEL button should be toggled out.

**Internal Switches** - If after connecting the monitor and starting up the color software, the display does not appear to be in sync, you may need to check the internal settings inside the monitor. To do so, you will need to remove the cover and several internal metal panels (which may be screwed to the bottom of the monitor as well). Viewed from the front of the monitor, on the left hand side is the Scan Board, on the right hand side is the Video Board. Adjustments should be made with the power off, exercise caution!

On the Scan Board, there is a Scan Rate Jumper (pins 52-61) which controls the scan rate (different resolutions). This jumper should be in the bottom position (pins 52-59), specifying low resolution (525 lines).

Below it to the left are three potentiometers, the lower one (R126) controls the horizontal hold for low resolution. Other "pots" of interest control pincushion (R65), width (R78), height (R4), v-hold (R2), v-center (R24), h-center (R90).

On the Video Board, there is a Sync Jumper (pins 35-37) which controls the sync rate. This jumper should be in the upper position (pins 35-36).

Further details can be found in the Conrac 7211 Manual.



---



---

## DSPSCALE

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

DSPSCALE allows a program to output to different types of streams (display, Interpress, etc.) without corrections for scaling. This module provides self-scaling graphics through two different methods: a virtual self-scaling *image stream* that overlays a regular image stream and/or new versions of the various image stream graphic manipulation functions (DRAWLINE, DSPTOPMARGIN, etc.). The goal of both methods is to make it possible to modify the normal scaling factor of an image stream without modification to the program generating the output.

### VIRTUAL SELF-SCALING IMAGE STREAM

This module implements a virtual image stream type, called SCALED, which is used to overlay any other image stream and provide automatic scaling to the natural scale of the image stream or any user selected scaling factor. The function OPENIMAGESTREAM is used to overlay a scaled image stream over a regular one. For example, the following will open a scaled image stream on top of an Interpress image stream:

```
(OPENIMAGESTREAM (OPENIMAGESTREAM 'TEST.IP 'INTERPRESS) 'SCALED)
```

The only difference between the virtual stream and a normal image stream is that the SCALE argument to the DSPSCALE function is active and can be used to change the scale of the stream (multiplying the scale specified by the standard scaling factor of the stream).

### SELF-SCALING GRAPHICS FUNCTIONS

As an alternative to the self-scaling image stream, self scaling versions of the various graphics functions are provided. For most of the graphic functions, self-scaling versions have been defined which have an ! (exclamation point) at the end of their name (eg. DRAWLINE vs. DRAWLINE!):

|                      |                  |                 |
|----------------------|------------------|-----------------|
| CENTERPRINTINREGION! | DRAWELLIPSE!     | DSPSCALE!       |
| CHARWIDTH!           | DRAWLINE!        | RELDRAWTO!      |
| CHARWIDTHTHY!        | DRAWPOINT!       | RELMOVETO!      |
| CURSORPOSITION!      | DRAWPOLYGON!     | SCALEDBITBLT!   |
| BITBLT!              | DRAWTO!          | STRINGREGION!   |
| BITMAPBIT!           | FILLCIRCLE!      | STRINGWIDTH!    |
| BLTSHADE!            | FILLPOLYGON!     | DSPSPACEFACTOR! |
| DSPBACKUP!           | FONTPROP!        | DSPTOPMARGIN!   |
| DSPBOTTOMMARGIN!     | GETPOSITION!     | DSPXOFFSET!     |
| DSPCLIPPINGREGION!   | DSPLEFTMARGIN!   | DSPXPOSITION!   |
| DRAWARC!             | DSPLINEFEED!     | DSPYOFFSET!     |
| DRAWBETWEEN!         | MOVETO!          | DSPYPOSITION!   |
| DRAWCIRCLE!          | MOVETOUPPERLEFT! |                 |
| DRAWCURVE!           | DSPRIGHTMARGIN!  |                 |

The set includes both output *and* input functions since it is necessary when getting, for example, a mouse position, to unscale the position to put it back into the program's virtual coordinate system. By default, these functions can be used directly in place of their non-! counterparts and they will automatically scale their arguments to the DSPSCALE of the output stream. Some of the above functions are not identical with their non-! counterparts, as explained below:

(DSPSCALE! *SCALE STREAM*) [Function]

In this version of DSPSCALE, the *SCALE* argument is active and will multiply *STREAM*'s normal scaling factor. If you have a program that draws a circle, for example, to a window using the appropriate ! functions, you can cause it to draw a different size circle (larger or smaller) by using DSPSCALE! to change the scaling factor of the window without touching the source program.

(CHARWIDTH! *CHARCODE FONT STREAM*) [Function]

(CHARWIDTHY! *CHARCODE FONT STREAM*) [Function]

(FONTPROP! *FONT PROP STREAM*) [Function]

(STRINGWIDTH! *STR FONT FLG RDTBL STREAM*) [Function]

All of the above functions have one extra argument (as compared to their non-! equivalents) which is the *STREAM* in question. This is necessary to do the scaling calculations.

The module also defines a couple of new stream manipulation functions:

(DSPTRANSLATE! *Tx Ty STREAM*) or (DSPTRANSLATE! *POSITION STREAM*) [Function]

Defines the amount of X and Y translation that should be added to graphic operations to *STREAM*. Similar to DSPTRANSLATE but works even if the image stream does not have an IMTRANSLATE method. The second form of the arguments is for backward (Koto) compatibility.

(DSPUNITS! *UNITS STREAM*) [Function]

Essentially the inverse of DSPSCALE!, this function lets you set how many *UNITS* (pixels or whatever) the source program generates for each unit pixel on the output stream (multiplied by the output stream's default scaling).

It is possible to use both the virtual image stream and the self-scaling graphics functions together as long as the self-scaling graphics functions are applied to the real stream, not the virtual one.

### Lisp Data Type Scaling Functions

The routines below are used by the ! functions and the virtual image stream for scaling numbers, positions, regions and other data types and are useful for defining other self-scaling functions:

(DSPSCALE.BRUSH *BRUSH STREAM*) [Function]

(DSPSCALE.DASHING *DASHING STREAM*) [Function]

(DSPSCALE.POINTS *KNOTS STREAM*) [Function]

(DSPSCALE.REGION *REGION STREAM [SmashRegion]*) [Function]

(DSPSCALE.NUMBER *NUMBER STREAM*) [Function]

(DSPSCALE.POSITION *POSITION STREAM [SmashPosition]*) [Function]

(DSPSCALE.XPOSITION *NUMBER STREAM*) [Function]

(DSPSCALE.YPOSITION *NUMBER STREAM*) [Function]

(DSPSCALE.WIDTH *WIDTH STREAM*) [Function]

(DSPUNSCALE.REGION *REGION STREAM [SmashRegion]*) [Function]

(DSPUNSCALE.POSITION *POSITION STREAM [SmashPosition]*) [Function]

(DSPUNSCALE.NUMBER *NUMBER STREAM [OFFSET]*) [Function]

(DSPUNSCALE.XPOSITION *NUMBER STREAM*) [Macro]

(DSPUNSCALE.YPOSITION *NUMBER STREAM*) [Macro]

---



---

## EDITBG

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

EDITBG is a tool for editing both the background and background border shades. The functions CHANGEBACKGROUND and CHANGEBACKGROUNDBORDER both take a shade argument but the shade is interpreted differently. A normal black & white shade consists of 16 pixels (see EDITSHADE in the Interlisp Reference Manual) as does the border shade, which covers twice the area. The normal shade has 4 x 4 pixels but the border shade has 2 x 8 pixels where the pixels are twice as tall. WHITESHADE and BLACKSHADE appear the same for both, as does the standard background shade (shown below) but arbitrary shades do not appear the same.

**Background**

|   |    |    |    |
|---|----|----|----|
|   | 14 | 13 | 12 |
|   | 10 | 9  | 8  |
| 7 | 6  |    | 4  |
| 3 | 2  |    | 0  |

**Background Border**

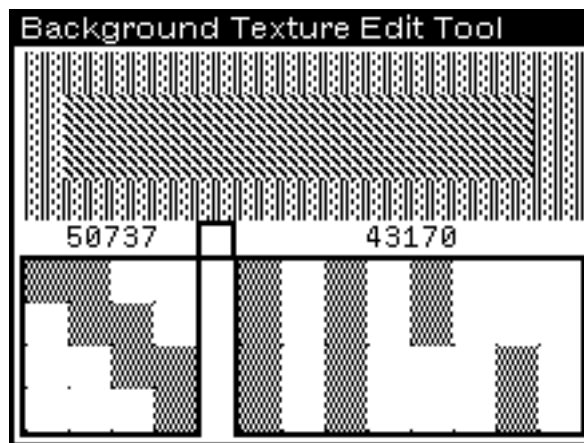
|   |    |    |    |   |    |   |   |
|---|----|----|----|---|----|---|---|
|   | 14 | 13 | 12 |   | 10 | 9 | 8 |
|   |    |    |    |   |    |   |   |
| 7 | 6  |    | 4  | 3 | 2  |   | 0 |
|   |    |    |    |   |    |   |   |

$$34850 = 2^{15} + 2^{11} + 2^5 + 2^1$$

(EDITBACKGROUND)

[Function]

Brings up an edit tool (also available from the background menu) which lets you edit both a normal shade and a border shade and see how they combine:



The bottom half of the window has a background texture editor on the left and a border texture editor on the right. The top half of the window shows the background texture within the border texture as it would appear on the screen. Buttoning the small box in the center of the window will change the background and border textures on the screen to those displayed.

---

---

**EDITKEYS**

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

EDITKEYS provides a set of "logical" keys corresponding to the Dandelion (1108) function keys. This allows 1132 users to take advantage of interfaces designed for the 1108 keyboard. Calling (BUILDFNKEYS) builds a window with 8 keys like the one below:



These keys can be "pressed" by bugging the mouse (left or middle mouse button) inside the key's image. The effect of pressing one of these keys is to generate the same character code as the key generates on the 1108. The state of the shift keys (at the time the mouse is let up) are taken into consideration, but interfaces that use KEYDOWNP are not affected.

---

---

## EQUATIONEDITOR

---

---

### PROGRAMMER'S GUIDE

By: Tad Hogg (Hogg.pa@Xerox.com)

#### DESCRIPTION

This document describes how to define new kinds of equations to be used with the equation editor in TEdit, and how to construct equation image objects by function calls.

#### User Switches:

The global variable *EquationDefaultSelectionFn* specifies the default function to be called when an equation is selected with the middle mouse button. The initial value, EQIO.DefaultSelectFn, allows the user to select a piece of the equation by selecting from a menu.

The global variable *UnknownEquationData* is a formatted string to use for displaying equations whose types are not defined.

The global variable *EquationInfo* is used to record all currently defined equation types. The specification information can be obtained by calling

(EQIO.GetInfo *type info*) [Function]

which returns the specified info for equations of the given type. Any of the PROPS mentioned below for EQIO.AddType, or *formFn* or *numPieces*, can be used as a value for *info* to get the corresponding data for this type of equation. Example: (EQIO.GetInfo 'fraction 'numPieces) returns the number of pieces in a fraction.

Individual specification items of an existing equation type can be modified using

(EQIO.SetInfo *type info newValue*) [Function]

although the caller must be sure that any new information is consistent with the remaining properties. For example, (EQIO.SetInfo 'fraction 'menuLabel myLabel) will make the value of myLabel be used for fractions in the equation type menu.

#### Defining new kinds of equations:

This module allows new types of equations to be defined. An equation consists of some number of pieces of text and, perhaps, some extra symbols or lines. A method for computing the relative position of the various pieces must be specified when new equation types are used. The pieces of equations consist of "formatted strings" which allow font information to be associated with strings and can also include image objects (which thus allows equations to contain other equations). Formatted strings are described below. Additional properties can be specified to determine the particular behavior of the new equation.

Additionally, a function can be provided to allow equations of the new type to be created under program control. Typically these are named EQ.Make.xxx where xxx = atom specifying the equation type.

Specifically, a new equation type (or a new definition for an existing type) is created by calling

(EQIO.AddType *type formFn numPieces PROPS*) [Function]

where

- *type* is an atom identifying the equation type (e.g. fraction)
- *formFn* is the name of a function, described in detail below, which specifies the relative location of the equation pieces and, if requested, draws any extra lines or symbols required by the equation that are not included in any of its pieces. The *formFn* can also specify the selection region to be used for each piece of the equation, i.e. that region of the equation within which the left mouse button can be used to select the piece.
- *numPieces* is the number of parts the equation has (e.g. a fraction has two parts: numerator and denominator). If the equation has a variable number of pieces, then *numPieces* is the default initial value.
- *PROPS* is a prop list of optional properties for the equation which are described below.

*The equation form function:*

The form function is called with arguments (eqnObj imageStream draw?) and specifies the size of the entire equation and the location of each piece with respect to the lower left corner of the box. Furthermore, if *draw?* is non-NIL, it draws any extra lines or symbols required by the equation that are not included in any of its pieces. (If *draw?* is NIL, nothing should be drawn -- the function is being called only to determine how big the equation is and the relative location of its parts.) The *formFn* can also specify the selection region to be used for each part of the equation.

For example, a fraction has two parts (numerator and denominator) and a single extra line between them. In this case the *formFn* draws the line and specifies the location of the numerator and denominator.

Specifically, the *formFn* should return an equation specification created by a call to

(EQIO.MakeSpec *box dataSpecList*) [Function]

where *box* is an IMAGEBOX which specifies the size of the equation; and *dataSpecList* is a list containing a piece specification for each piece of the equation. A piece specification is created by a call to

(EQIO.MakeDataSpec *pos selectRegion*) [Function]

where *pos* is the position, relative to the lower left corner of the box, where this piece is to be displayed and *selectRegion* gives the selection region to use for this piece (relative to the l.l. corner of the equation box). If *selectRegion* is NIL, then the region within which the piece is displayed will be used as the selection region. Normally the *selectRegion* should include the display region inside it, and is intended to allow selection regions to be larger than just the display region. Note that *pos* specifies where the stream should be positioned before displaying the formatted string defining the contents of the piece.

The ATTACHEDBOX routines, described below, may be useful for constructing the form functions of new equations since it provides functions to position various regions so that they do not overlap.

*Equation type properties:*

The following properties can be specified for any equation type when it is defined with EQIO.AddType. Note that all equations of a given type have the same values for these properties.

- *changeFn* A function with argument (eqnObj) called when the number of pieces in a variable piece equation is changed. It is meant to allow any specific properties such as saved menus to be adjusted to reflect the change.
- *initialData* A way to specify the text to be used when equations of this type are created. It can be either the name of a function or a list. If it is a list, then it should contain a single integer for each piece of the equation. This number specifies how the font size for that piece should be changed relative to the initial font size set in TEdit. For example, 0 means use the default font, +1 means use the next bigger font, -2 means use a font two sizes smaller, etc. Missing values default to zero, i.e. the pieces are initially set in the normal size font. The actual fonts used are specified by the array EquationFontSpecs and any request for a font larger (smaller) than the largest (smallest) available in the array defaults to using the largest (smallest). In this case, the initial text will be a single blank.

If initialData is a function, it is called with arguments (initialFontSpec type numPieces dataList) when a new equation of this type is created. It should return a list of formatted strings, with each item in the list to be used for the corresponding piece of the equation. The argument initialFontSpec will specify the current font when the equation is added; and numPieces will be the number of pieces in the equation. dataList, if non-NIL, is a list of items to use for each of the pieces of the equation. The items can be either format strings, or just a string in which case the initialData function should attach an appropriate font.

- *initialPropFn* A function with argument (type) called when a new equation of this type is created. It returns a prop list to be used as properties for this equation. These values are added to (and override) any props given in objectProps. For example, this allows the user to specify the number of rows and columns in a new matrix. If the number of pieces is to be specified, it should be returned as the value of the numPieces prop, and the equation type should allow a variable number of pieces.
- *makeFn* A function which constructs an image obj for this type of equation from its arguments. Generally this will just provide a convenient way of calling EQN.Make.
- *menuLabel* A label to use for this type in the equation type menu displayed after selecting Equation in the main TEdit menu. If not given, the type atom is used as the label.
- *objectProps* A prop list of properties and their initial values that are needed for this equation. These properties can be used by the formFn to lay out the equation and will typically be modified by the wholeEditFn.
- *pieceNames* A list of names of the pieces of the equation. This is used by the default middle button selection function to provide a menu of choices. E.g. ("numerator" "denominator") for a fraction. If this property is not specified, then the default menu will just contain the numbers of the pieces. This is generally meant for equations with a fixed number of pieces.
- *specialSelectFn* A function with argument (eqnObj) to be called when the equation is selected with the middle button instead of the default action. It should return the number of the piece selected, or NIL if no piece of the equation is selected. The selected piece, if any, will then be edited.
- *wholeEditFn* A function with arguments (eqnObj window button) called when the entire equation, rather than a single piece, is selected. This can be used to change global properties of

the equation and should return non-NIL if the object is modified, NIL otherwise. *window* is the window which contains the equation and *button* is the mouse button used to select it.

- *variable?* Non-NIL to indicate this equation type allows a variable number of pieces.

*Equation data functions:*

The functions used with new equation types should make use of the routines provided for formatted strings as well as the following functions when using the various equation data:

- **(EQIO.EqnType eqnObj)** returns the atom specifying the kind of equation *eqnObj* is.
- **(EQIO.EqnDataList eqnObj)** returns the list of formatted strings which specify the pieces of the equation.
- **(EQIO.SetDataList eqnObj newDataList)** replaces the data list (i.e. the list of formatted strings corresponding to each piece of the equation) with *newDataList*. The caller must update the number of pieces in the equation appropriately. This is useful, for instance, when the wholeEditFn has made major changes in the equation.
- **(EQIO.EqnData eqnObj piece#)** returns the formatted string corresponding to the piece of *eqnObj* specified by *piece#*.
- **(EQIO.EqnProperty eqnObj prop {newValue})** returns the current value of the specified property of *eqnObj* if *newValue* is not present, otherwise sets the specified property to the value of *newValue* even if it is NIL. This can be used to associate arbitrary properties with individual equations. Currently, the following properties are used by the equation editor and should not be used for other purposes:
  - *fontSpec* a specification of the current font at the time the equation was created
  - *numPieces* the current number of pieces in a variable-piece equation
  - *selectionMenu* the current default middle-button selection menu for a variable-piece equation

When equations are copied, any data items that are not atoms, strings or lists are set to NIL. These items are also PRIN2'ed on files. Thus other data types should only be used to cache values (e.g. menus) that can be recomputed if necessary from the other properties.

- **(EQIO.NumPieces eqnObj {newValue})** returns the current number of pieces in *eqnObj* if *newValue* is not present. Otherwise if *eqnObj* is a variable-piece equation, it sets the number of pieces to *newValue* and adjusts any necessary properties by calling the equation's changeFn.

Additionally, properties can be associated with the equation type itself by use of

- **(EQIO.TypeProp type prop {newValue})** which gets or sets the property *prop* for equation *type*. This can be used to save properties that are the same for all equations of a given type (e.g. selection menus for equations with a fixed number of pieces).

Equation image objects can be created by a program (and then, for example, inserted into TEdit) by use of

- **(EQN.Make type dataList fontSpec PROPS)** which makes a *type* equation whose arguments are format strings contained in *dataList*. *fontSpec* is an initial font specification and *PROPS* is a prop list of equation properties which should include *numPieces* for equations with a variable number of pieces.



**FORMATSTRINGS**

The basic components of equations are represented as formatstrings which allow fonts and super/subscripting to be associated with strings and can also include image objects. A format string is a list of items, each of which is either an imageobject or a list giving a font specification, a string and an optional shift specifying the number of points to move up when displaying the string. The following functions can be used to create and display format strings as well as insert and extract them from TEXTSTREAMS. All formatstrings must be on a single line.

**Functions:**

For creating and accessing format strings:

(FS.MakeItem *fontSpec string shift*) [Function]

creates a formatstring item from *fontSpec*, a font specification such as (Gacha 10), *string* and *shift*. The string can be null.

(FS.ItemFont *item*) [Function]

returns the font associated with *item*, or NIL if *item* is an imageobject.

(FS.ItemValue *item*) [Function]

returns *item* if it is an imageobject, otherwise its associated string.

(FS.ItemShift *item*) [Function]

returns the shift associated with *item*.

For inserting and extracting from TEXTSTREAMS:

(FS.Extract *stream*) [Function]

returns a format string created from the text in the TEXTSTREAM stream. Any unallowed characters (as determined by FS.AllowedChar) in the stream are ignored. Note that any format information other than character fonts, super/subscripting and image objects is discarded. The file pointer associated with the stream is modified.

(FS.Insert *data stream*) [Function]

inserts the format string *data* at the current location in TEXTSTREAM stream.

For displaying and manipulating the format strings:

(FS.Box *data imageStream*) [Function]

returns an IMAGEBOX specifying the size of the format string *data* on *imageStream*.

(FS.Copy *data*) [Function]

returns a copy of the format string *data*.

(FS.Display *data imageStream invert?*) [Function]

displays the format string *data* on *imageStream*. If *invert?* is non-NIL, the display is inverted.

(FS.Get *fileStream*) [Function]

reads a formatstring, or list of formatstrings, from the current location on *fileStream*.

(FS.Put *data* *fileStream*)

[Function]

prints the formatstring (or list of formatstrings) *data* to *fileStream*.

Additional functions:

(FS.AllowedChar *charcode*)

[Function]

returns non-NIL if *charcode* is allowed in formatstrings.

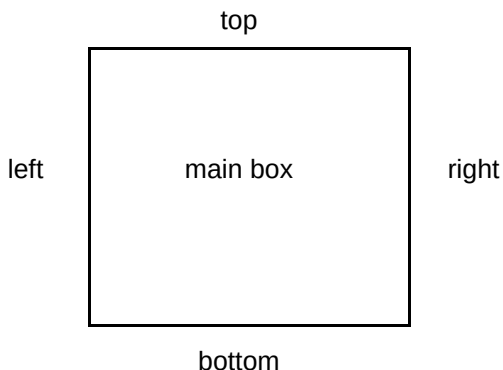
(FS.RealStringP *item* *nullOK*)

[Function]

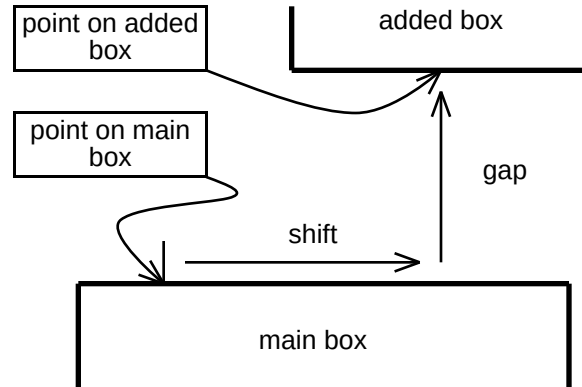
returns non-NIL if *item*'s value is a string (rather than an imageobject) and either *nullOK* is non-NIL or the string is not the nul string.

## ATTACHED BOXES

The following functions place image boxes in specific locations with respect to a main box so that the added boxes won't overlap. The desired position of a new box is specified by the side of the main box to place it next to and the position of a point on the side of the new box with respect to a point on the side of the main box. The placed regions are specified with respect to the lower left corner of the main box. Sides are specified by one of the atoms *top*, *bottom*, *left* or *right* and are with respect to the main box.



The position of the added box is specified relative to some side of the main box. Specifically, the location of a reference point on the near side of the added box, the *addPt*, is given with respect to a reference point on the side of the main box, the *mainPt*. The possible points along the side are specified by one of the atoms *low*, *high*, *center* or *display* corresponding to the corner nearest the lower left corner of the box, the corner farthest from the l.l. corner of the box, the center of the side, and the display point of the image box respectively. The location of the *addPt* with respect to the *mainPt* is specified by a distance along the side, the *shift*, and a distance perpendicular to the side, the *gap*. These distances can be positive or negative. A negative value for the *gap* will cause the added box to overlap the main box. All boxes are assumed to have no kerning (i.e. *XKERN* field is zero).



### Functions:

For creating and accessing format strings:

(AB.PositionRegion *mainBox addedRegions side mainPt addBox addPt gap shift clear*) [Function]

Positions *addBox* with respect to *mainBox* avoiding overlap with previously added regions. The parameters are: *mainBox* is an image box specifying the main box; *addedRegions* is a list of regions (measured with respect to the lower left corner of *mainBox*) that have already been placed next to the main box; *side* is the side (one of *top*, *bottom*, *left* or *right*) of the main box next to which the new box should be placed; *mainPt* is the reference point along the side of the main box (one of *low*, *high*, *center* or *display*); *addBox* is an image box specifying the box to be placed; *addPt* is the reference point along the side of the added box; *gap* and *shift* specify the relative positions of the reference points; and *clear* is the minimum (nonnegative) distance that the new box is allowed to be from any of the previously added regions (NIL defaults to zero which prevents any overlap of the added regions).

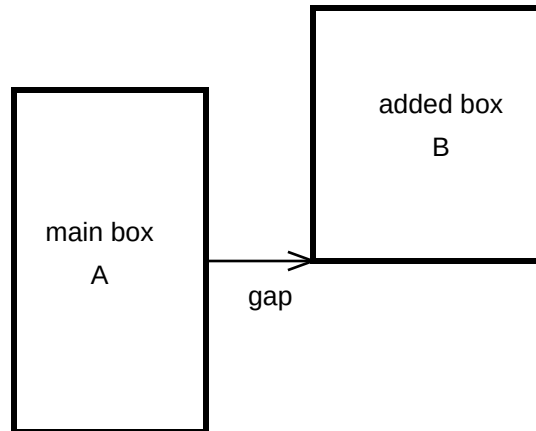
The function returns a list of the form (*region newAddedRegions*) where *region* is the region, w.r.t. the lower left corner of *mainBox*, where *addBox* was placed and *newAddedRegions* is the list of added regions updated to include this newly placed region. If the specified location of *addBox* causes it to be within a distance *clear* of any of the regions in *addedRegions*, the box is moved away from the main box in a direction perpendicular to the side (i.e. the gap is increased) until it is far enough from the previous regions.

(AB.Position2Regions *mainBox addedRegions side highBox highPt lowBox lowPt highGap lowGap highShift lowShift clear*) [Function]

This function places two boxes next to the same side of *mainBox*. If the two new boxes are within a distance *clear* of each other, they are moved apart in a direction parallel to the side next to which they are placed so that the distance each box moves is proportional to its size. Then these regions are individually checked for being too close to previously added regions and, if necessary, are moved away from the main box (i.e. perpendicular to the side). The function returns a list of the form (*highRegion lowRegion newAddedRegions*).

Example:

To place box B to the right of box A such that the low point of the near side of box B is a distance *gap* from the center of the right side of A, i.e.



use (AB.PositionRegion A *addedRegions* 'right' center B 'low gap 0) where *addedRegions* is a list of previously added regions which should not overlap B.

---

---

## EQUATIONS

---

---

By: Tad Hogg (Hogg.pa@Xerox.com)

### DESCRIPTION

This module provides interactive editing of mathematical equations within TEdit. An equation consists of a number of pieces of text and possibly one or more special symbols. For many purposes, such as deletion and copy selection, equations behave as a single (large) character in TEdit. Operations on their pieces are described below.

#### To load:

The equation editor and a standard set of equation types is obtained by loading EQUATIONS.LCOM.

#### To use:

This section describes the procedures by which equations can be inserted into documents and their pieces modified.

##### *Adding an equation to a TEdit document:*

To add an equation, first move the caret to the desired insertion point and then select "Equation" from the main TEdit menu (obtained by holding down the middle button in the window's title bar). This will display a menu of known equation types. Selecting one of these will insert the corresponding equation into the document at the current location of the caret. To abort the insertion, click outside the menu. Once the equation is inserted, a subeditor will be created for each of the equation pieces, one at a time. This can be used to fill in the various pieces of the equation and its use is described below. Some equation types will prompt for additional information before inserting the new equation (e.g. inserting a matrix will prompt for the desired number of rows and columns).

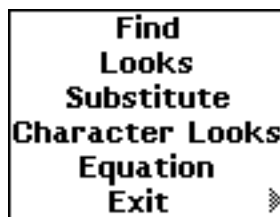
##### *Editing a currently existing equation:*

In order to modify a piece of an existing equation (e.g. the numerator of a fraction), the piece must be selected with the mouse in one of two ways. First, you can point at the piece in the displayed equation and press the left mouse button. Alternatively, pointing at the equation and pressing the middle button will display a menu from which the desired piece can be selected. This is useful for selecting pieces that are too small to conveniently point at with the mouse. In either case, if a piece is selected, a subeditor will start on that piece. In addition, some equation types may also allow changes to global properties when no specific piece is selected (e.g. changing the number of rows in a matrix).

##### *Using the equation piece subeditor:*

The subeditor is attached to the bottom of the main edit window and allows individual pieces of the equation to be modified with normal TEdit operations. While a subeditor is active, the corresponding piece of the equation in the main window is inverted. Since the text in equation pieces must be on a single line, the subeditor will not accept control characters such as carriage returns. Instead the edit window will flash when such characters are typed.

In the subeditor, the TEdit menu is modified to provide a limited set of TEdit commands as well as additional commands relevant to equations. The menu appears as



Selecting *Find*, *Looks*, *Substitute* or *Character Looks* invokes the corresponding TEdit action. Selecting *Equation* acts as described above and allows equations to be embedded inside other equations. *Exit* ends the subedit of the equation piece, updates the equation in the main editor and, if this is a newly inserted equation, automatically starts editing the next piece.

The *Exit* item also has three possible subitems which are used to exit from the equation editor and specify a desired follow up action. Specifically, *Next Piece* ends the edit of the current equation piece and creates a new editor on the next piece. In this context, the pieces of the equation are considered to form a circular list so that successive uses of the *Next Piece* option will edit each piece of the equation in turn. The second subitem, *Finish Eqn*, ends the current equation edit and does not continue with any other pieces of the equation. Finally, *Abort* ends the current edit without changing the equation.

When a subeditor is terminated, any TEdit looks or formatting other than character fonts and sub/superscripting are ignored.

The subeditor can also be terminated by the key normally used to advance to the next fill-in slot in TEdit (i.e. text of the form ">...<<"). Specifically, if there are no remaining slots in the subeditor, using this key is equivalent to selecting *Exit* from the command menu described above. By default, this key is the middle-blank key on Dolphins and Dorados and the OPEN key on DLions.

### User Switches:

The global variable *EquationFontSpecs* is an array of font specifications in order of increasing size which is used to determine initial fonts for the equation pieces. This can be modified if additional or different default fonts are desired.

The global variable *EQ.UseNSChars* determines the kind of characters to use when displaying equations that use special symbols (e.g. sum or product) on the screen. Specifically, if non-NIL then symbols from the NS character set are used, otherwise the Sigma 20 font is used. It is initially set to NIL.

When NS characters are used (on the screen when *EQ.UseNSChars* is non-NIL, or for Interpress), the global variable *EQ.NSChars* determines the particular NS characters to use. It is a property list of the form (TYPE1 ITEM1 ...) where TYPE is the kind of equation (e.g. SUM, PRODUCT, etc) and ITEM gives the font and character number to use, e.g. ((MODERN 30) 61301) for an INTEGRAL. This variable compensates for the lack of large symbols in various Interpress fonts.

The file EQUATIONPROGRAM.TEDIT describes how to define new kinds of equations, as well as how to create equations in a program with function calls.

### Examples:

Examples of equations are given in the file EQUATIONEXAMPLES.TEDIT. The equation module must be loaded before reading this file.

**Limitations:**

- Equations that are larger than the available room in the current TEdit window will not be displayed.
- The text of each piece of an equation must be on a single line.
- All image objects inserted into equations, as well as the equations themselves, must not have any kerning, i.e. the XKERN field of all imagebox records must be zero.

**Koto Incompatibility:**

Due to a change in image object I/O, files containing equations written in Lyric/Medley may not be readable in Koto.

---



---

## ETHERBOOT

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: Various microcode, germ and boot files.

ETHERBOOT is a Envos Lisp background network server process which allows Dandelions and/or Doves (other than the one the server is running on) to boot utility programs from the Ethernet (as an alternative to floppies). On a Dandelion, a 3, 4 or 6 boot from the maintenance panel initiates an Etherboot; on a Dove the boot icons are used (sometimes in combination with a number key):

| Dandelion | Dove | Boot Type                                     |
|-----------|------|-----------------------------------------------|
| 0003      | F3   | Ethernet non-diagnostic boot of the Installer |
| 0004      | F7   | Ethernet diagnostic boot of the Installer     |
| 0006      | F3-1 | Ethernet boot of experimental software        |

(ETHERBOOT [*LOGFILE*]) [Function]

To start the server, (ADD.PROCESS ' (ETHERBOOT) ). *LOGFILE* is an optional argument which should be an open stream to log transactions in.

BOOTFILEDIRECTORIES [Variable]

The boot files are searched for on the directories in this list which should point to the (possibly remote) directory where the boot files are kept, initially ' ( {CORE} {DSK} ). The server will not respond to requests for boot files that are not available.

(CACHE.BOOT.FILES [*TYPES*]) [Function]

Since Lisp can take longer to open a remote file than the timeout on some (simple) requests, this function can be used to copy some of the boot files listed in ETHERBOOTFILES to the BOOTFILECACHEDIRECTORY. *TYPES* defaults to those listed in BOOTFILECACHETYPES.

BOOTFILECACHEDIRECTORY [Variable]

The directory into which CACHE.BOOT.FILES copies boot files, initially {CORE}.

BOOTFILECACHETYPES [Variable]

The default types of files that CACHE.BOOT.FILES copies to the BOOTFILECACHEDIRECTORY, initially ' (DB GERM) .

BOOTFILEREQUESTTYPES [Variable]

An association list which contains the type numbers of the requests that the boot server handles along with a description of the request type and the function which handles it. Currently, the request types are *Simple* and *SPP*.



## ETHERBOOTFILES

[Variable]

The table of boot file numbers and names. Each entry consists of a description of the boot file, the name of the file and the file number (48 bit) by which the file is requested. Since the boot server is table driven, different boot files can be substituted. Initially, ETHERBOOTFILES contains:

|                                                |                            |              |
|------------------------------------------------|----------------------------|--------------|
| ("Standard DLion Ethernet Initial Microcode"   | EtherInitial.db            | 2852126720)  |
| ("Standard DLion Diagnostic Microcode"         | MoonBoot.db                | 2852126728)  |
| ("Standard DLion Mesa Microcode"               | Mesa.db                    | 2852126736)  |
| ("Standard DLion Germ"                         | DLion.germ                 | 2852126744)  |
| ("Standard DLion Boot File"                    | SimpleNetExecDLion.boot    | 2852126752)  |
| ("Standard DLion Diagnostics Boot File"        | EIDiskDLion.boot           | 2852127232)  |
| ("Standard DLion Installer Boot File"          | InstallerNSDLion.boot      | 2852127234)  |
|                                                |                            |              |
| ("Alternate DLion Ethernet Initial Microcode"  | EtherInitialAlt.db         | 2852126721)  |
| ("Alternate DLion Mesa Microcode"              | Mesa.db                    | 2852126738)  |
| ("Alternate DLion Germ"                        | DLion.germ                 | 2852126746)  |
| ("Alternate DLion Boot File"                   | InstallerNSDLion.boot      | 2852126754)  |
|                                                |                            |              |
| ("Standard TriDLion Diagnostic Microcode"      | Moonboot.db                | 2852126729)  |
| ("Standard TriDLion Mesa Microcode"            | TridentRavenMesa.db        | 2852126737)  |
| ("Standard TriDLion Germ"                      | TriDLion.germ              | 2852126745)  |
| ("Standard TriDLion Boot File"                 | SimpleNetExecTriDLion.boot | 2852126753)  |
|                                                |                            |              |
| ("Alternate TriDLion Mesa Microcode"           | TridentRavenMesa.db        | 2852126739)  |
| ("Alternate TriDLion Germ"                     | TriDLion.germ              | 2852126747)  |
| ("Alternate TriDLion Boot File"                | InstallerNSTriDLion.boot   | 2852126753)  |
|                                                |                            |              |
| ("Standard Dove Ethernet Initial Microcode"    | EtherInitialDove.db        | 2852128768)  |
| ("Standard Dove Diagnostic Microcode"          | MoonRise.db                | 2852128776)  |
| ("Standard Dove Mesa Microcode"                | MesaDove.db                | 2852128784)  |
| ("Standard Dove Germ"                          | Dove.germ                  | 2852128792)  |
| ("Standard Dove Boot File"                     | SimpleNetExecDove.boot     | 2852128800)  |
|                                                |                            |              |
| ("Alternate Dove Ethernet Initial Microcode"   | EtherInitialDove.db        | 2852128769)  |
| ("Alternate Dove Diagnostic Microcode"         | MoonRise.db                | 2852128777)  |
| ("Alternate Dove Mesa Microcode"               | MesaDove.db                | 2852128785)  |
| ("Alternate Dove Germ"                         | Dove.germ                  | 2852128793)  |
| ("Alternate Dove Boot File"                    | InstallerNSDove.boot       | 2852128801)  |
|                                                |                            |              |
| ("Dove Simple Net Exec"                        | SimpleNetExecDove.boot     | 2852128824)  |
| ("Dove Configuration Utility"                  | SysConfigOfflineDove.boot  | 2852128825)  |
| ("Dove Installer"                              | InstallerNSDove.boot       | 2852128826)  |
| ("Dove Diagnostics Utility"                    | DiagDiskUtilDove.boot      | 2852128828)  |
| ("Dove Rigid Disk Diagnostics Utility"         | DiagRDDove.boot            | 2852128829)  |
| ("Dove Ethernet Diagnostics Utility"           | DiagEtherDove.boot         | 2852128830)  |
| ("Dove Keyboard & Display Diagnostics Utility" | KDMDove.boot               | 2852128831)) |

The boot file numbers overlay the host number space so Dandelion/Dove boot file numbers begin at 25200000000 octal.

## KNOWN PROBLEMS

- The server can only handle one connection at a time.
- Due to as yet unknown reasons, a Dandelion running the server is not able to service simple Dove requests; all other combinations should work.

---

## EXAMINEDEFS

---

By Ron Kaplan

This document created in December 2021.

`EXAMINEDEFS` brings up side-by-side windows for comparison of the definitions of `NAME` as `TYPE` from sources `SOURCE1` and `SOURCE2`.

```
(EXAMINEDEFS NAME TYPE SOURCE1 SOURCE2 TITLE1 TITLE2 REGION) [Function]
```

The kind of comparison is determined by the value of the variable `EXAMINEWITH`. If `EXAMINEWITH` is `SEdit`, the different definitions are shown in side-by-side `SEdit` windows. This allows for examination--but not editing--of the definitions: The structures shown in `SEdit` are copies of the definitions that `GETDEF` obtains from the sources and so any changes in either `SEdit` will have no lasting effect. A separate `SEdit` session with a particular definition (maybe different from either of the sources) can be used in parallel for edits guided by the separate examination. Also, currently, the particular locations of differences in the two definitions are not highlighted by `SEdit` selections.

If `EXAMINEWITH` is `COMPARETEXT` (the initial value), then the definitions are printed in read-only `TEDIT` windows (as per `PF-TEDIT`) and compared with `COMPARETEXT`. The `COMPARETEXT` browser makes it easy to iterate from difference to difference.

If `SOURCE1` or `SOURCE2` is a list and not a `GETDEF` source specification, then a copy of that structure will be taken as the definition to be shown.

If `REGION` is a region, it is used as the initial suggestion for the constellation region that side-by-side `SEdit` windows will share. (`COMPARETEXT` windows are determined by the `COMPARETEXT` protocols.)

`TITLE1` and `TITLE2` if provided are used to override the default titles of the examination windows. The windows are attached, in that they move, reshape, and close together, but scroll independently.

A second function is provided to bring up side-by-side `TEDIT-SEE` windows as an aid in examining the differences between files. The optional `REGION` again specifies a constellation region that the `TEDIT`'s for the two files will share.

```
(EXAMINEFILES FILE1 FILE2 TITLE1 TITLE2 REGION) [Function]
```

---

---

**EYECON**

---

---

By: Bob Bane (Bane.envos@Xerox.com)

**DESCRIPTION**

EYECON creates a window with two eyes in it that follow your cursor around. The eyes also wink when you click your mouse buttons: left and right buttons close the corresponding eye, middle button closes both. Here's what it looks like:



The EYECON window can be shrunk into an icon that looks like this:



doing this kills the process which tracks the cursor. Opening the icon restarts the tracking process.

To start it, load EYECON and call (EYECON.OPEN *left bottom*) to create an EYECON window at (*left bottom*).

**INSPIRATION**

I hacked this up after hearing about something similar that runs under Suntools. The bitmap for the eyes was borrowed from Spy and slightly modified (thanks, Larry!). Julian Orr suggested that the eyes should wink.

---

---

## FILEWATCH

---

---

Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester)

This document last edited on October 19, 1987.

### INTRODUCTION

FILEWATCH is a facility for keeping an eye on open files. It periodically updates a display showing each open file stream, its current file pointer location, the total file size, a percentage bar, and a read/write/both indicator.

### DESCRIPTION

Invoking the function FILEWATCH (or selecting the "FileWatch" entry on the BackgroundMenu) starts up the FileWatch process if not already running, or brings up a FileWatch control menu allowing you to forget a currently displayed file (*i.e.*, stop displaying the file), recall a previously forgotten file, close an open file (after mouse confirmation), change some or all FileWatch display properties, or quit the FileWatch process. The Forget, Recall and Close entries on the FileWatch control menu have roll-outs to let you perform the operation on several files at once.

FileWatch can be customized by setting the FileWatch properties (see below) using the function FILEWATCHPROP. Right buttoning any FileWatch window brings up the FileWatch control menu, with the provision that the Forget and Close commands apply to the file displayed in that FileWatch window. Middle buttoning any FileWatch window allows you to move the entire FileWatch display, and left buttoning cause the window to be redisplayed.

### DETAILS

(FILEWATCH Command)

[Function]

If Command is 'ON and no FileWatch process is already running, starts a process to watch open files. If Status is 'OFF or 'QUIT and there is a FileWatch process running, kills the process. If Command is neither one of the above nor one of the FileWatch commands listed below, starts a process to watch open files if not already running, otherwise brings up the FileWatch control menu. Returns the process if running, otherwise NIL.

FORGET

[FileWatch command]

Brings up a menu of files currently being watched. Select the one you no longer want to have watched.

FORGET-MANY

[FileWatch command]

Repeatedly performs the FORGET command until no other files are being watched or you make a null selection.

RECALL

[FileWatch command]

Brings up a menu of forgotten files. Select the one you want to have watched again.

RECALL-MANY [FileWatch command]

Repeatedly performs the RECALL command until all forgotten files are being watched again or you make a null selection.

CLOSE [FileWatch command]

Brings up a menu of open files. Select the one you want to have closed.

CLOSE-MANY [FileWatch command]

Repeatedly performs the CLOSE command until all open files have been closed or you make a null selection.

MOVE [FileWatch command]

Performs the SET-ANCHOR, SET-POSITION, and SET-JUSTIFICATION commands.

SET-ANCHOR [FileWatch command]

Brings up a menu of four corner names. Select the one on you wish to anchor the FileWatch display. For instance, selecting Top-Right causes FileWatch windows to be stacked downwards with the top right corner of the first FileWatch window at the FileWatch display position.

SET-POSITION [FileWatch command]

Indicate where the FileWatch display should be positioned by moving the region of the combined FileWatch windows.

SET-JUSTIFICATION [FileWatch command]

Requests confirmation to turn FileWatch window justification on, *i.e.*, make all FileWatch windows the same width as the largest one.

(FILEWATCHPROP PropName [PropValue]) [Function]

If PropValue is given, sets the property value accordingly. Always returns the current (old) value of the property. This is a general facility which you can use for whatever purpose you deem appropriate. However, there are some properties that have a predefined meaning to FileWatch:

ALL-FILES? [FileWatch property]

If NIL, FileWatch displays only user visible open files; otherwise all open files (including, for example, dribble and file cacher files) are displayed. Initially set to NIL. Caveat: setting this property to T will give you access to things that might be dangerous to play with. In particular, closing certain system files on the Dorado may cause your machine to crash, and may leave the local file system in an unhealthy state.

ANCHOR [FileWatch property]

Each open file that is being watched gets its own FileWatch window. Multiple windows are stacked automatically. The total region occupied by this stack is anchored at the corner indicated by this property. The only legal values are TOP-LEFT, TOP-RIGHT, BOTTOM-LEFT, BOTTOM-RIGHT. Initially set to BOTTOM-RIGHT. If the anchor is at one of the bottom corners the stack grows upward, otherwise downward. If the anchor is at one of the left corners the stack is aligned by left edge, otherwise by right edge (see also the JUSTIFIED? property).

## FILTERS

[FileWatch property]

A list of file patterns, for example '("{CORE}\*.\*;\*)'. An open file that matches any of the patterns will not be watched. Initially set to NIL. Note that each pattern is expanded to include the HOST and DIRECTORY equal to that of (DIRECTORYNAME), EXTENSION and VERSION equal to "\*", unless already specified. For example, in my case, the filter "\*JUNK\*" expands to "{Ice}<Koomen>Lisp>\*JUNK\*.\*;\*". If you really wanted to filter all junk files, use the filter "{\*}\*JUNK\*".

## FONT

[FileWatch property]

The font used for the FileWatch displays, specified in a form suitable to give to the function FONTCREATE. Initially set to '(GACHA 8).

## INTERVAL

[FileWatch property]

The value given to the function BLOCK. This should be either NIL or an integer indicating the number of milliseconds to wait between FileWatch display updates. Initially set to 1000. Note that FileWatch generates several FIXP's for large files every time through the loop, so setting this to NIL may cause excessive storage allocation and reclamation.

## JUSTIFIED?

[FileWatch property]

If T all FileWatch windows are aligned along both left and right edges, and are grown or shrunk as needed to accomodate the maximum filename length currently in use. This is aesthetically more pleasing but incurs increased overhead due to frequent reshaping of the windows. Initially set to NIL.

## POSITION

[FileWatch property]

The location of the anchored corner of the FileWatch display. Initially set to the bottom right corner of the screen: (CONS SCREENWIDTH 0).

## SHADE

[FileWatch property]

The shade used for the FileWatch thermometers. Initially set to GRAYSHADE.

## SORTFN

[FileWatch property]

Either NIL or the name of a function taking two filenames as arguments (such as ALPHORDER), which is used to sort the list of open files being watched. Initially set to NIL (*i.e.*, no sorting).

---

---

## FILLREGION

---

---

Originally By: Mike Bird (Inference Corp., Los Angeles, CA)

Jim Wogulis (Wogulis@ICS.UCI.EDU)

Greg Wexler (Wexler.pasa@Xerox)

New Owner: James M. Turner (Turner.Lexington@Xerox.com)

### INTRODUCTION

The Fillregion package provides a function which will allow the user to "fill in" arbitrary regions of a bitmap or window with a shade or bitmap (or any valid shade argument to BITBLT).. The regions must be defined by a black or white outline. There are two functions provided to the user: FILL.REGION and AUTO.FILL.

(FILL.REGION *window.or.bm interior.pos shade*) [Function]

*window.or.bm* : Must be either a window or bitmap otherwise an error occurs.

*interior.pos* : Must be a position within window.or.bm that is within the interior of the region to be filled.

*shade* : Shade can be any valid shade argument that BITBLT will accept.

This will return the window.or.bm with the specified region filled in. The region to be filled is determined by the pixel specified at interior.pos. If the pixel is black, all the connected black regions will be shaded, otherwise, if the pixel is white, all the connected white region will be filled. If the user aborts the function before completion, the original window.or.bm will be restored.

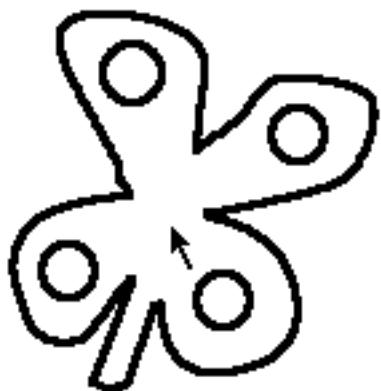
(AUTO.FILL *shade*) [Function]

*shade* : Shade can be any valid shade argument that BITBLT will accept.

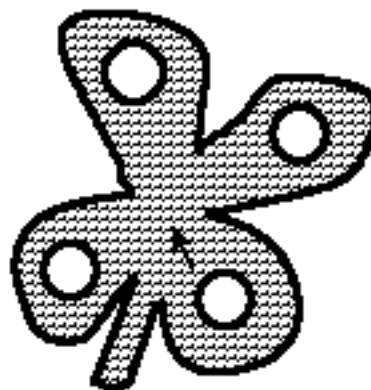
With your mouse pointing inside the appropriate region in a window, this function will fill in the region with the shade specified. This package only works for one bit per pixel bitmaps, color is not supported.

### Example:

(AUTO.FILL 1234)



results in:



Comments and suggestions are welcome.



---

---

**FINGER**

---

---

By: Greg Nuyens (mcvax!inria!nuyens@seismo.css.gov)

**INTRODUCTION**

Finger is a facility for determining and displaying information about other users running Xerox LISP. It displays the user's name, the Etherhostname (or the octal net address when no nameserver is available) and the user's idle time (time since last keystroke or mouseaction). Only other users who have the finger server loaded will be displayed. Users can specify the net radius to query, a list specifying only which users they want displayed, or similarly, only which hosts are to be displayed.

Loading Finger begins the finger server (which responds to queries). To display finger information, the following top-level function is provided:

(FINGER WHO HOST HOPS ICON?) [Function]

*WHO* is an optional list of usernames specifying which people are to be displayed if a response is received. *who* defaults to **FINGER.CROWD**, initially **NIL** meaning display all responses.

*HOST* is an optional list of etherhostnames analogous to *WHO*. Specifying both *WHO* and *HOST* denotes union. *HOST* defaults to **NIL**, denoting all hosts.

*HOPS* specifies the net radius to query. 0 specifies only nets to which you are directly connected. *hops* defaults to **FINGER.NET.HOPS**, initially 2.

*ICON?* specifies whether initial display should be the finger icon



or a display window. *ICON?* defaults to **NIL** meaning display. {typically, in an init file the call would be **(FINGER NIL NIL NIL T)**}.

The display window is updated each time the users bugs the display window with left or middle mouse button, and when most window operations are performed on the display window (shape, repaint, expand from icon, etc.). Right button retains the standard window menu.

Options:

the following are user specifiable variables affecting the operation of Finger.

**FINGER.ICON.POSITION** a position indicating the original position for the icon. Initially (900,500).

**FINGER.DISPLAY.POSITION** a position indicating the original position for the display window. Initially (650,325).

**FINGER.DISPLAY.HEIGHT** height of the display window. Initially 140. The display width is correct for the display format and need not be changed.

**FINGER.TIMEOUT** milliseconds to wait for the last response packet. Initially 1500.

**FINGER.NET.HOPS** net radius to be queried.

**FINGER.CROWD** list of potential users to be displayed (discussed above).

**FINGER.INFINITY.MINUTES** number of minutes to be considered infinite idle time. Initially 90.

Additional functions of interest to the user are:

**(END.FINGER)** which kills the finger server process, closes the sockets, closes the windows, etc.

**(FINGER.SERVER)** will start a finger server process.

---

---

## FREE MENU CREATOR

---

---

By : André Blavier (Rank Xerox France)

This document last edited on September 29, 1988

The Free Menu Creator application is designed to create interactively Free Menu description lists.

It is a full graphical tool with which you can move, shape, box, group the Free Menu Items and attach properties to them. At any time you can make the application compute the description list and then test the Free Menu that you have just created.

### OPENING A FREE MENU CREATOR (FMC) WINDOW

Once you have loaded the application the 'FMCreator' option is added to the background menu. By selecting it you will create a FMC window of the form:

The screenshot shows the Free Menu Creator (FMC) window interface. It is composed of three main panels:

- ITEM PROPERTIES** (Top Left):
  - Buttons: **APPLY**, **SHOW**, **NEW**
  - TYPE: MOMENTARY LABEL
  - ID: [ ]
  - FONT FAMILY: GACHA SIZE: 10 FACE: STANDARD
  - BOX 0: [ ] BOX SHADE: [ ] BACKGROUND: [ ]
  - MENU: (NIL) INIT STATE: NIL
  - CHANGE STATE: (FUNCTION NIL)
  - SELECTED FN: (FUNCTION NIL)
  - DOWN FN: (FUNCTION NIL)
  - HELD FN: (FUNCTION NIL)
  - MOVED FN: (FUNCTION NIL)
  - LINKS: (NIL) INFINITE WIDTH: [ ]
  - MESSAGE: [ ]
- GROUP PROPERTIES** (Top Right):
  - Buttons: **APPLY**, **SHOW**
  - ID: [ ]
  - COLLECTION: [ ]
  - DESELECT: [ ]
  - BOX 1: [ ]
  - BOX SHADE: [ ]
  - BACKGROUND: [ ]
- Free Menu Creator** (Bottom): A large empty rectangular area for the menu content.

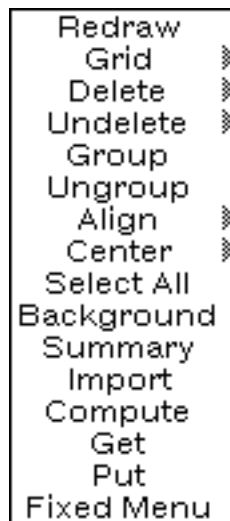
The bottom window is the **main window** : you will add, move ... items to it.

Two menu windows (Free Menus in fact) are attached on top of the main window : the **Item Properties menu** (IP menu) and the **Group Properties menu** (GP menu). They are provided for setting and changing item or group properties.

Between the menus and the main window is the **Prompt window**. Its purpose is mainly to display various information such as messages, prompts ...

## THE RIGHT BUTTON MENU

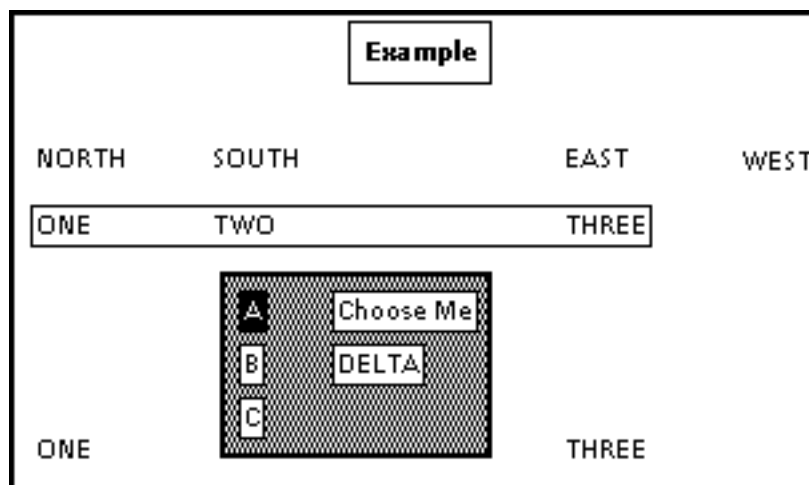
Pressing the right button inside the main window will pop up the following menu:



You can fix this menu on the right edge of the main window by selecting the 'Fixed Menu' option.

## A SAMPLE SESSION

Suppose you want to create the following Free Menu (this example is taken from the FREE MENU Lyric Release Notes):



First create the 'Example' item:

- select **TYPE** in the IP menu : the following pop-up menu appears:

| TYPE      |
|-----------|
| MOMENTARY |
| TOGGLE    |
| 3STATE    |
| STATE     |
| NWAY      |
| EDIT      |
| NUMBER    |
| EDITSTART |
| DISPLAY   |

- choose the type of item you want to create, e.g. DISPLAY
- select **LABEL** in the IP menu and type 'Example'
- choose the font by clicking in the **FAMILY**, **SIZE** and **FACE** items, which will cause the following menus to pop up:

| FAMILY     | SIZE | FACE       |
|------------|------|------------|
| CLASSIC    | 6    | REGULAR    |
| MODERN     | 7    | ITALIC     |
| TERMINAL   | 8    | BOLD       |
| TITAN      | 9    | BOLDITALIC |
| GACHA      | 10   |            |
| HELVETICA  | 11   |            |
| TIMESROMAN | 12   |            |
|            | 14   |            |
|            | 18   |            |
|            | 24   |            |
|            | 30   |            |
|            | 36   |            |

- now, the properties you wanted for that new item are set. So click in the **NEW** field and then move the mouse to the place where you want to put the item.

When you release the mouse button the item is fixed in the window. it is surrounded by a gray rectangle which means that it is **selected**:

**Example:**

If you move the mouse inside the selection rectangle the cursor will change to:

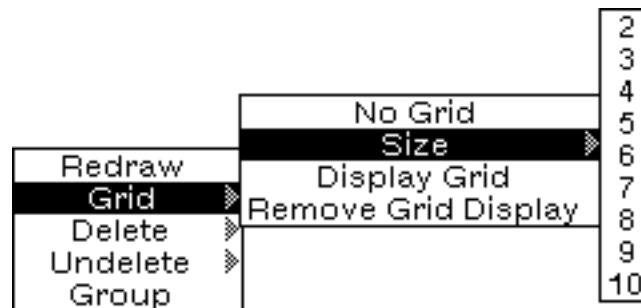


By pressing the left button while the cursor is inside a selection, you can move that selection.

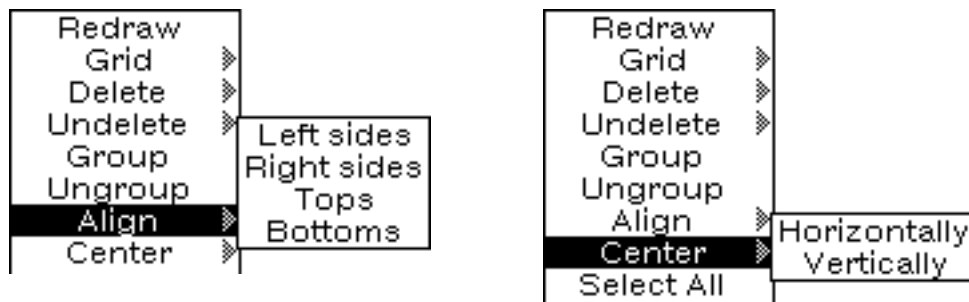
Repeat the same operation for the 'NORTH', 'SOUTH', 'EAST', 'WEST', 'ONE', 'TWO' and 'THREE' items.

To place the items exactly where you want you can use (and combine) the following facilities:

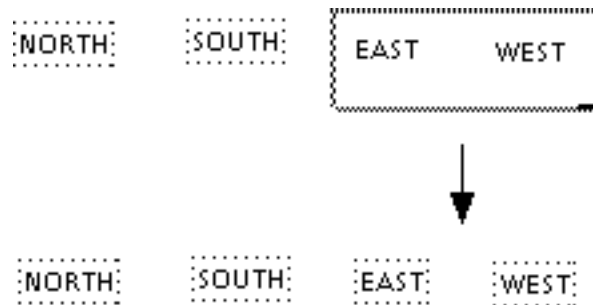
- use the **GRID** (from the right menu):



- align and center a **multiple selection** :

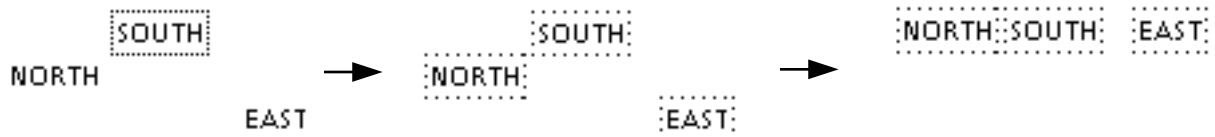


Select multiple items by pressing the left button and the **META** key, while the cursor is above them. You can also extend a multiple selection by pressing Left-Meta outside any item and shaping a ghost region :



Each item of a multiple selection is surrounded by a dotted rectangle. You can move a multiple selection in the same way you move a unique selection.

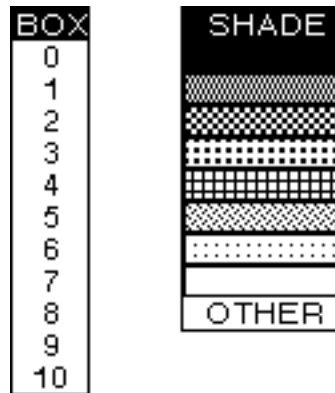
The **Align** and **Center** functions all refer to the first selected item. For example, if you choose 'Align Tops' then all the items except for the first one will be moved so that their top is at the same Y coordinate as that of the first item :



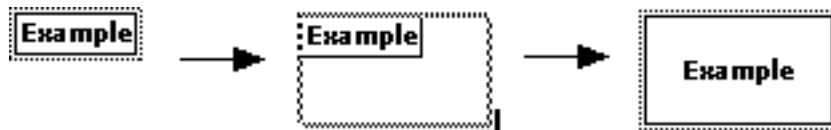
Now, let's box the 'Example' item :

- select the item

- select **SHOW** in the IP menu : this updates the menu according to the item
- set the BOX and BOXSHADE properties :



- select **APPLY** : the item is redisplayed surrounded by a box
- press the middle button inside the item : you can now shape the box (the shaping is controlled so that the space between the item and its box is the same horizontally and vertically) :



Whenever you need to change some items's property (ies) , select the item and then select **SHOW** in the IP menu : this updates the menu according to the item. Then, you can change any property, and update the item according to the IP menu by selecting **APPLY**.

You can also change properties of multiple items : make a multiple selection, set the property (ies) you want to change, then select **APPLY** : the following menu pops up :



Only the selected property will be applied to the items.

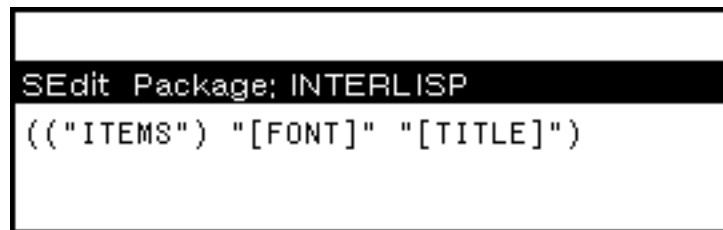
When setting a box property on multiple items, a box is created around each item, not around the whole selection. To wrap a set of items by a box you must first **GROUP** the items. That's what we want to do for the 'ONE', 'TWO', 'THREE' items :

- select the items
- choose **GROUP** from the right menu

Groups are automatically boxed at creation. You can modify groups in the same way as you modify simple items, using the GP menu. Groups can be moved and shaped like simple items.

Now, create the 'A', 'B', 'C' items, setting their TYPE to NWAY. Group the items, select **SHOW** in the GP menu, set COLLECTION to 'COL1', set DESELECT to T - then select **APPLY** : you have created the collection.

Create the 'DELTA' item and then the 'Choose Me' STATE item. For this last item specify a MENU property, selecting **MENU** in the IP menu. This opens the following SEdit window :



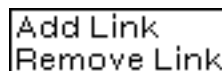
The first element of the list should be an item list, as suitable for standard menus. The second and third elements are optional : FONT should be a list of the form (FAMILY SIZE FACE), TITLE should be a listatom or a string.

Edit the list ((BRAVO DELTA) (MODERN 12 ITALIC)) and close the SEdit window. Now, select **INITSTATE** : this will pop up the following menu :



As you can see the items you just edited are part of this menu. Select DELTA.

One more property is required : the **LINKS** property. Select LINKS : the following menu appears :



Choose 'Add Link' and click on the DELTA item : the link is created.

At any time you can create the actual Free Menu out of your FMC window by selecting the **COMPUTE** option from the right menu. This creates the description list, which is stored in the **FM-DESCRIPTION** global variable. A call to the FREEMENU function is done automatically with FM-DESCRIPTION as an argument, and the Free Menu window is opened.

## REFERENCE GUIDE

### General window behavior

The main window, the prompt window and the properties windows behave as a whole. Moving, shaping, shrinking and closing can be directed from anyone of them. Shaping affects only the main



window. Closing is protected by a confirmation request when unsaved modifications have occurred. The main window is not scrollable.

The right menu, when fixed, can be closed solely.

Hardcopying the main window does not use the standard hardcopy functions. Instead, the contents is 'pretty-hardcopied', but the current version can't print large windows on multiple pages.

### The right menu

**Redraw** : use Redraw to redisplay the contents of the window. This can be useful when items overlap.

**Grid** : selecting the Grid option displays the grid state in the prompt window.

**No Grid** : removes the grid.

**Size** suboptions : specifies a grid size (in pixels). Grid alignment refers to the lower left corner of items.

**Display Grid** : displays the grid in the window.

**Remove Grid Display** : removes the grid display, but the grid remains active.

**Delete** : deletes the selected item(s). Items deleted are saved in a list so they can be undeleted. The number of deleted items is displayed in the prompt window.

**Forget save list** : deleted items are destroyed and can't be undeleted.

**Undelete** : undeletes last deleted item.

**Last** : same effect

**All** : undeletes all deleted items

**List** : pops up a menu of all the deleted items. The selected item gets undeleted.

**Group** : groups a multiple selection. A group can include groups. The components of a group are not individually accessible.

**Ungroup** : unpacks a group. The ungrouping operation works only at the first level, i.e. included groups are not unpacked.

**Align** : the Align suboptions all work on a multiple selection (items and/or groups). The alignment operation refers to the first selected item.

**Left sides** : align left sides of selected items

**Right sides** : align right sides of selected items

**Tops** : align tops of selected items

**Bottoms** : align bottoms of selected items

**Center** : the Center suboptions all work on a multiple selection (items and/or groups). The centering operation refers to the first selected item.

**Horizontally** : center items so their center is on the same X coordinate as that of the first selected item

**Vertically** : center items so their center is on the same Y coordinate as that of the first selected item

**Select All** : selects all the window contents

**Background** : pops up a shade menu for setting the window background shade

**Summary** : creates a TEdit window listing a summary of the window contents. Only interesting properties are described, depending on the item type. Groups contents are indented hierarchically.

**Import** : allows items importation from a Free Menu. This function works but is currently bugged : importing items with a MENU property is not supported ; when importing groups only their contents are imported.

**Compute** : generates a description list of the items suitable for the FREEMENU function. The list is stored in the FMC-DESCRIPTION global variable. Opens a Free Menu built out from FMC.

**Get** : loads the contents of a FMC window, previously stored on disk by a Put operation. The loaded items (groups) are added to the current contents of the window.

**Put** : saves the contents of a FMC window on disk.

**Fixed Menu** : when the right menu is popped up by the right mouse button, attaches the menu to the main window. Once the menu is fixed, has no effect.

### The Item Properties menu

**APPLY** : if the selection is unique sets the selected item properties to the properties described in the menu. If the selection is multiple, pops up a menu of properties and sets the selected property of all the selected items according to the menu.

**SHOW** : updates the menu according to the selected item, thus allowing editing its properties.

**NEW** : creates a new item which properties are described in the menu.

**TYPE** : lets the user specify the TYPE property from a pop-up menu.

**LABEL** : lets the user edit the LABEL property. If no label is edited the item will be displayed in FMC with the pseudo-label '\*NOLABEL\*'.  
Right-buttoning in this field will start label edition, clearing the field first.

Middle-buttoning puts the cursor in the GETREGION state, and sets the label to the bitmap specified by the user.

**ID** : lets the user edit the ID property. The edited string is always MKATOMed in the description list.

**FONT** : FAMILY, SIZE and FACE allow font descriptions from pop-up menus.

**BOX** : BOX=0 means no box.

**BOXSHADE, BACKGROUND** : pop up a shade menu, including an 'OTHER' option for shade editing.

**MENU** : opens a SEdit window where the user specifies the MENUITEMS, MENUFONT and MENUTITLE properties. The list must be of the form (MENUITEMS [MENUFONT] [MENUTITLE]).

**INITSTATE** : pops up a menu of possible INITSTATE values, depending on the item TYPE and possible MENUITEMS.

**CHANGESTATE, SELECTEDFN, DOWNFN, HELDFN, MOVEDFN** : open a SEdit window for the corresponding property. The edited list can be of the form (FUNCTION function-name) or a LAMBDA-expression.

**LINKS** : pops up a menu with 2 items : 'Add Link' and 'Remove Link'. 'Add Link' prompts the user to click on the item that is to get linked (this item must have an ID). 'Remove Link' removes the link.

**INFINITEWIDTH** : this toggle item is suitable for EDIT or NUMBER items only.

**MESSAGE** : starts editing the MESSAGE property (right buttoning clears the field first).

### The Group Properties menu

**APPLY, SHOW, ID, BOX, BOXSHADE, BACKGROUND** behave in the same way as in the IP window.

**COLLECTION** : starts editing a COLLECTION name. Specifying a COLLECTION property is suitable only for groups which items are NWAY.

**DESELECT** : a toggle item whose value is relevant only for COLLECTIONs.

## About item shaping

The effects of middle buttoning inside an item is different whether the item is boxed or not.

If the item is boxed, shaping will be constrained as described in the sample session, i.e. BOXSPACE must be the same horizontally and vertically.

If the item is not boxed, shaping is constrained so that only the item's width can be changed, in fact modifying the MAXWIDTH property.

## Copy functions

The effects of pressing the COPY key and left buttoning when a FMC window has the TTY depend on what is to be copied :

- when clicking in a SKETCH window, the selected objects are made a graphic LABEL
- when clicking in a FMC window, the selected items are copied in the target window (the source and the target can be the same).

---



---

## FONTSAMPLER

---



---

By: Nick Briggs (Briggs.pa@Xerox.com)

This document last edited on April 27, 1987

### INTRODUCTION

The FONTSAMPLER package provides a function which will easily generate sample sheets of your favourite fonts on your favourite printer.

### FUNCTIONS

(FontSample *Fonts CharacterSets Printer StreamType*) [Function]

Creates a sample sheet for the font described by the font descriptor(s) *Fonts* on the image stream generated by passing *Printer* and *StreamType* to OPENIMAGESTREAM. *Fonts* may be either a single font descriptor (the result of a call to FONTCREATE) or a list of font descriptors. *CharacterSets* may be either a single number, or a list of numbers representing character sets. The default value of *CharacterSets* (if NIL is passed) is 0 (the Roman Alphabet and Punctuation). FontSample will create a page for each font and character set combination specified, consisting of all the characters of the font/character set combination, arranged on a 16 by 16 grid, and labelled in the default 12pt font for the stream.

For example, to create a sample sheet for the Modern 12pt regular font, character sets 0 and 239 (general and technical symbols 2) on the Interpress printer Fountain one might do:

```
(FontSample (FONTCREATE 'MODERN 12 'MRR 0 'INTERPRESS) '(0 239) '{LPT}Fountain:)
```

(FontSampleFaked *FontAsList Printer StreamType*) [Function]

There are very often fonts on printers that Interlisp-D cannot speak about, because it is lacking a font width file. FontSampleFaked can produce sample sheets for many of these fonts. *FontAsList* should be a font expressed in the raw form that FONTCREATE takes, *Printer* and *StreamType* are as for FontSample, however passing DISPLAY for *StreamType* will not produce meaningful results.

To produce a sample sheet for the Vintage 10pt bold font on the Interpress printer Fountain one might incant:

```
(FontSampleFaked ' (Vintage-Printwheel 10 BRR) '{LPT}Fountain:)
```

### NOTES

Be careful if you create sample sheets for many fonts at once – especially the larger type sizes (24pt and up) – you may cause an Interpress printer to hang and require rebooting to recover.

Subject: Interlisp to Commonlisp conversion package  
From: darrelj%sm.unisys:COM  
To: info-1100%sumex-aim.stanford

FORMACRO and FORMACRO.DFASL -- Still another portable iteration macro for commonlisp. Its main claims are almost 100% compatibility with the semantics of the Interlisp-Clisp FOR (especially when used the the XFORMS which fix a few incompatibilities); and user extensibility (unfortunately not compatible with IL:I.S.OPR). Embedded keywords (e.g. IN, COLLECT) may be in any package.

---

## FTPSEVER-MULTI- CONNECTIONS

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

Requires: FTPSEVER, FTPSEVERPATCH and DPUPFTPATCH

### INTRODUCTION

This package (actually a complimentary pair of files) extends the capabilities of the Lisp Library FTPSEVER package to support multiple simultaneous connections between Xerox 11xx series AI workstations.

### INSTALLATION

To install this package, load the FTPSEVERPATCH.LCOM file on the 11xx machine(s) that are to be servers (this will load FTPSEVER if it is not already loaded). Then load the DPUPFTPATCH.LCOM file on any of the 11xx machines that are to be clients of these servers. You must set the value of IL:\*FTP.NEGOTIATED.CONNECTION.HOSTS\* on each of the client machines to specify the server machines that support the FTPSEVERPATCH system of multiple simultaneous connections (below).

### VARIABLES

IL:\*FTP.NEGOTIATED.CONNECTION.HOSTS\* [Global Variable]

This variable must be set to specify the server machines that support the FTPSEVERPATCH system of multiple simultaneous connections. Its value is a list of PUP host numbers. (Specifically, it is a list of the values of (CAR (BESTPUPADDRESS <SERVER-HOST-NAME>)) for each of the server machines.)

### HOW IT WORKS

This package modifies the DPUPFTP code of the client machines, so that when it is trying to open an FTP connection BSP stream, it first checks to see if the server host is one of the IL:\*FTP.NEGOTIATED.CONNECTION.HOSTS\*, and if so, it sends a message to the modified FTPSEVER on that system (using PUP type \PT.NEGOTIATED.CONNECTION (= 128) on PUP socket \PUPSOCKET.NEGOTIATED.CONNECTION (=63)). The server machine creates a socket for this connection and starts a standard FTPSEVER listener process on this socket, and returns the socket number to the client. (The process is modified so it will go away when the connection is closed instead of lingering forever.) The client uses the returned socket number for the connection instead of \PUPSOCKET.FTP. If the server is NOT on IL:\*FTP.NEGOTIATED.CONNECTION.HOSTS\*, or fails to respond within 10 seconds with the new socket number, then \PUPSOCKET.FTP is used. When the negotiated connection server is started on the server machine (with the incantation (FTPSEVER) which is the original FTPSEVER start up), it also will start up a *permanent* FTPSEVER listener on \PUPSOCKET.FTP so regular connection requests can be handled.

### ACKNOWLEDGEMENTS

Thanks to Tom Lipkis of Savoir for suggesting this sort of scheme.

---

## GITFNS

---

By Ron Kaplan

This document was last edited in February 2023.

GITFNS provides a Medley-oriented interface for comparing the files in two different branches of a git repository. This makes it easier to understand what functions or other definitions have changed in a Lisp source file, or what text has changed in a Tedit file. This may be particularly helpful in evaluating the changes in a pull request.

Separately, GITFNS also provides tools and conventions for bridging between git's file-oriented style of development and version control and Medley's residential development style with its own version control conventions. GITFNS allows for intelligent comparisons between Lisp source files, Tedit files, and text files in a local git clone and a local Medley-style working directory, and for migrating files to and from the git clone and the working directory.

## Git projects: Connecting git clones to GITFNS capabilities

The GITFNS capabilities operate on pre-existing clones of remote git repositories that have been installed at the end of some path on the local disk. The path to a clone can be used to create a GITFNS "project" for that clone:

```
(GIT-MAKE-PROJECT PROJECTNAME CLONEPATH WORKINGPATH EXCLUSIONS
 DEFAULTSUBDIRS) [Function]
```

where

PROJECTNAME is the name of the project (e.g. MEDLEY, NOTECARDS, LOOPS...)

CLONEPATH specifies the local path to the clone

e.g. {dsk}<users>...>git-medley

WORKINGPATH is optionally the local path to a corresponding Medley-residential working directory

(e.g. {dsk}<users>...>working-medley>)

When the project has a WORKINGPATH:

EXCLUSIONS is a list of files and directories to be excluded from comparisons (including what its .gitignore specifies)

DEFAULTSUBDIRS is a list of subdirectories to be use in working-path comparisons when directories are not otherwise specified.

For convenience, if CLONEPATH is NIL or T (and not a path), then a sequence of probes based on PROJECTNAME attempts to find a clone directory (with a .git subdirectory):

(UNIX-GETENV PROJECTNAME) e.g. (UNIX-GETENV 'LOOPS)

```
(UNIX-GETENV (CONCAT PROJECTNAME "DIR") e.g.{UNIX-GETENV 'LOOPSDIR}
(MEDLEYDIR PROJECTNAME)) a subdirectory of MEDLEYDIR
(MEDLEYDIR (CONCAT "../" PROJECTNAME)) a sister of MEDLEYDIR
(MEDLEYDIR (CONCAT "../git-" PROJECTNAME)
 (a sister of MEDLEYDIR named git-PROJECTNAME, e.g. git-notecards)
```

Thus:

```
If MEDLEYDIR is defined,
 (GIT-MAKE-PROJECT 'MEDLEY) will make the MEDLEY project
If NOTECARDS is defined
 (GIT-MAKE-PROJECT 'NOTECARDS) will make the NOTECARDS project
If NOTECARDS is not defined but the clone >git-notecards> is a sister of MEDLEYDIR, then the
 NOTECARDS project will still be created.
```

If a clone is discovered and a project is created, the value of `GIT-MAKE-PROJECT` is `PROJECTNAME`. Otherwise, `NIL` will be returned if `CLONEPATH` is `T` (= no-error), and `CLONEPATH=NIL` will result in an error.

When they are created, git projects are registered by name on the a-list `GIT-PROJECTS`, and they can otherwise be referenced by their names.

The variable `GIT-DEFAULT-PROJECT`, initially `MEDLEY`, contains the project name used by the commands below when the optional `PROJECTNAME` argument is not provided.

`GIT-MAKE-PROJECT` creates a pseudohost `{projectname}` whose path prefix is the path that resolved to the clone. The file `GITFNS` in the clone `LISPUSERS` directory, for example, can be referenced as `{MEDLEY}<LISPUSERS>GITFNS`.

`GIT-MAKE-PROJECT` will also create a pseudohost `{wprojectname}` for the user's working environment for the project. If `WORKINGPATH` is provided, that will be the prefix for that pseudohost. If `WORKINGPATH` is `NIL` and a directory named `working-projectname` is a sister to the clone directory, the pseudohost will point to that.

(GIT-INIT EVENT) [Function]

`GIT-INIT` creates the default set of projects when `GITFNS` is loaded, as specified in the variable `GIT-DEFAULT-PROJECTS`, initially containing `MEDLEY` `NOTECARDS` `LOOPS` `TEST`. `GIT-INIT` is added to `AROUNDEXITFNS` so that new pseudohost bindings for the default projects will be created if the `sysout` or `makesys` is started on a new machine.

GIT-DEFAULT-PROJECTS [Variable]

Determines the projects that are created (or recreated) by `GIT-INIT`. This is initialized for the `MEDLEY` `NOTECARDS` `LOOPS` `TEST` projects, with `CLONEPATH=NIL`

`GITFNS` also defines two directory-connecting commands for conveniently connecting to the git and working pseudohosts of a project:

`cdg (projectname) (subdir)` [Command]

`cdw (projectname) (subdir)` [Command]



For example, `cdg notecards library` connects to `{NOTECARDS}/library/`.

## Comparing directories and files in different git branches

In its simplest application, GITFNS is just an off-to-the-side add-on to whatever work practices the user has developed with respect to a locally installed git project. Its only advantage is to allow for more interpretable git-branch comparisons, especially for pull-request approval. These comparisons are provided by the `prc` ("pull request compare") Medley executive command:

```
prc (branch) (DRAFT) (projectname) [Command]
```

This compares the files in `branch` against the files in the main branch of the project (`origin/master` or `origin/main`). Thus, suppose that a pull request has been issued on github for a particular branch, say branch `rmk15` of the default project. Then

```
prc rmk15
```

brings up a `lispusers/COMPAREDIRECTORIES` browser for the files that currently differ between `origin/rmk15` and `origin/master`. If the selected files are Lisp source files, the Compare item on the file browser menu will show the differences in a `lispusers/COMPARESOURCES` browser. The differences for other file types will be shown in a `lispusers/COMPARETEXT` browser.

If `branch` is not specified and the shell command `gh` is available, then a menu of open pull-request branches will be provided. If `gh` is not available, the menu will offer all known branches. If the optional `DRAFT` is provided, then the menu will include draft PR's as well as open ones.

If one PR, say `rmk15`, contains all the commits of another (`rmk14`), then the menu will indicate this by

```
rmk15 > rmk14
```

Note that the `prc` comparison is read-only: any comments, approvals, or merges of the branch must be specified using the normal Medley-external git interfaces and commands.

`prc` is the special case of the more general `bbc` command ("branch-branch compare") for comparing the files in any two branches:

```
bbc branch1 branch2 (project) [Command]
```

This compares the files in `branch1` and `branch2`, for example

```
bbc rmk15 lmm12 (local)
```

This will compare the files in `origin/rmk15` and `origin/lmm12` in the `GIT-DEFAULT` project. `branch1` defaults to the origin files of the currently checked out branch, the second defaults to `origin/master`. If `local` is non-NIL, then a branch that has neither `local/` or `origin/` prepended will default to `local` (e.g. `local/rmk15`) instead of `origin/`. `Local` refers to the files that are currently in the clone directory, which may not be the same as the origin files, depending on the push/pull status.

Either of the branches can be specified with an atom `LOCAL`, `REMOTE`, or `ORIGIN`, in which case `bbc` will offer menus listing the currently existing branches of that type.

NOTE: Branch comparison makes use of a git command that has a limit (`diff.renameLimit`) on the number of files that it can successfully compare. A message will be printed if that limit is exceeded, asking whether a larger value for that limit should be applied globally.

The command `cob` ("check out branch") checks out a specified branch:

```
cob branch (next-title-string) (project) [Command]
```

This checks out `branch` of `project` and then executes `git pull`. The `branch` parameter may also be a local branch, `T` (= the current working branch), or `NEW/NEXT` (= the next working branch). The current working branch is the branch named `<initials>nnn`, e.g. `rmk15`. The initials are the value of `INITIALS` as used for `SEDIT` time stamps, and `nnn` is the largest of the integers of all of the branches beginning with those initials.

If `branch` is `NEW` or `NEXT`, then a new initialed branch is created and becomes the user's current branch. Its number is one greater than the largest number of previous initialed branches. If `next-title-string` is provided, then that string will be appended to the name of the branch, after the initials and next number, and two hyphens. Spaces in `next-title-string` will also be replaced by hyphens, according to git conventions.

If `branch` is not provided, a menu of locally available branches pops up.

The currently checked out branch is obtained by the `b?` command:

```
b? (project) [Command]
```

## Correlating git source control with separate Medley development

It is generally unsafe to do Medley development by operating with files in a local clone repository. Medley provides a residential development environment that integrates tightly with the local file system. It is important to have consistent access to the source files of the currently running system, especially for files whose contents have been only partially loaded. A `git pull` or a branch switch that introduces new versions of some files or removes old files altogether can lead to unpredictable disconnects that are hard to recover from. This is true also because development can go on in the same Medley memory image for days if not weeks, so it is important to have explicit control of any file version changes.

GITFNS mitigates the danger by conventions that separate the files in the git clone from the files in the working Medley development directory. The location of the Medley development source tree for a project is given by the `WORKINGPATH` argument to `GIT-MAKE-PROJECT`. If `WORKINGPATH` is `T` or `NIL` and there exists a directory `>working-projectname>` as a sister to the clone, then that is taken to be the `WORKINGPATH` and thus the prefix for a pseudohost `{Wprojectname}`.

When Medley development is carried out in the `WORKINGPATH`, the variable `MEDLEYDIR` should point initially to the working directory, and the directory search paths (`DIRECTORIES`, `LISPUSERSDIRECTORIES`, `FONTDIRECTORIES`, etc.) all have `MEDLEYDIR` (or `{WMEDLEY}`) as a prefix. In that case, the clone for the project, if `PROJECTPATH` doesn't specify it explicitly, should be located at the `>git-medley>` sister directory of `MEDLEYDIR`.

Any back and forth transfer of information between the git clone and Medley development must be done by explicit synchronization actions. Crucially, Medley-updated files do not appear in the clone directories and new clone files do not move to the Medley directories without user intervention.

The files in Medley working tree and the git clone of a project can be compared with the `gwc` ("git-working-compare") command:

```
gwc subdirectories (project) [Command]
```

This produces a browser for all the files in the corresponding `WORKINGPATH` subdirectories that differ from the files in the currently checked out branch of the git clone. If `subdirectories` is omitted, it

defaults to the `DEFAULTSUBDIRS` of the project. If it is `ALL`, then files in all subdirectories that are not found in the project's `EXCLUSIONS` are compared.

In addition to the commands for comparing and viewing files, the menu for this browser also has commands for copying files from the git clone `{projectname}` to `{Wprojectname}` and deleting files from `{Wprojectname}`.

If the master/main branch is the current branch then the menu has no commands to change the clone directory. The browser will show those files that have been updated from a recent merge, and they can individually be copied from the git branch to realign the two source trees with incremented Medley version numbers. If the comparison is with a different branch, say the user's current staging branch, copying files from the working Medley to the git clone or deleting git files will set git up for future commits.

Note that the menu item for deleting Medley files will cause all versions to be removed, not just the latest one, to avoid the possibility that an earlier one is revealed. Deletion for Medley files is also accomplished by renaming to a `{Wprojectname}<deleted>` subdirectory so that they can be recovered if a deletion is in error. Files in the git-clone are removed from the file system immediately, since git provides its own recovery mechanism for those files.

GITFNS does not (yet?) include functions for commits, pushes, or merge for updating the remote repository. Those have to be done outside of Medley through the usual github interfaces, as guided by the information provided by the comparisons.

---



---

## GRAPHCALLS

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: GRAPHER, MSANALYZE (WHERE-IS & HELPSYS optional)

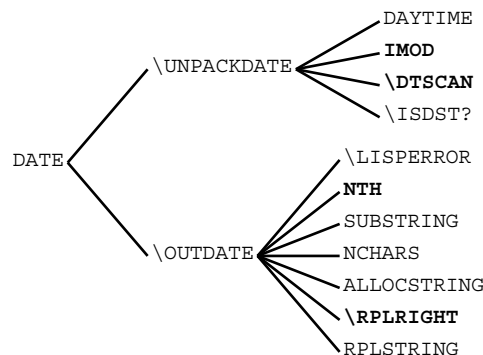
GRAPHCALLS is an extended graphical interface to the Envos Lisp CALLS function. It is to CALLS what BROWSER is to SHOW PATHS in MASTERSCOPE. It allows fast graphing of the calling hierarchy of both interpreted and compiled code, whether or not the source is available (see the CALLS function in the MASTERSCOPE section of the *Lisp Library Modules* manual), allowing examination of both user and system functions. The sources of the functions do not have to be analyzed by MASTERSCOPE first.

Additionally, buttoning a function on the graph brings up a menu of operations that can be done with the function, such as editing, inspecting, further graphing etc.

(GRAPHCALLS *FUNCTION* &REST *OPTIONS*)

[Function]

Graphs the calling hierarchy of *FUNCTION*. Terminal nodes on the graph (those which call no other functions or are undefined) are printed in a bold version of the graph's font indicating that they cannot be graphed further:



The remainder of the arguments, in keyword format, make up *OPTIONS* eg.

```
(GRAPHCALLS 'DATE :FONT '(GACHA 10) :DEPTH 4 :FILTER 'FGETD)
```

Options include:

**:STREAM** An image stream to display the graph on. The options list is saved on the stream.

**:FILTER** A predicate to apply to the functions when building the graph to test their eligibility to appear on the graph. The filter can be any defined function; the default is not to filter. Interesting filters include:

**WHEREIS** Limits the tree to only functions the user as has loaded and prunes out system functions and SYSLOADED files. Quite useful.

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |               |                                     |              |                                             |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|-------------------------------------|--------------|---------------------------------------------|
| <b>FGETD</b>          | Limits the tree to only functions that are actually defined. Thus if you are perusing the tree for BITBLT and do not have and are not interested in the color code, FGETD will remove all of the undefined color bitmap functions.                                                                                                                                                                                                                                                                                                                |               |                                     |              |                                             |
| <b>EXPRP</b>          | Limits the tree to interpreted functions. Useful for graphing functions in the development stage.                                                                                                                                                                                                                                                                                                                                                                                                                                                 |               |                                     |              |                                             |
| <b>CCODEP</b>         | Limits the tree to compiled functions.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |               |                                     |              |                                             |
| <b>NO\</b>            | Keeps low level functions starting with \ (i.e. \OUTDATE) off of the graph. Useful for getting an overview of system functions and when advising system functions (as \ed functions should probably not be advised).                                                                                                                                                                                                                                                                                                                              |               |                                     |              |                                             |
| <b>:DEPTH</b>         | The calling hierarchy is graphed to <i>depth</i> levels (defaults to 2).                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |               |                                     |              |                                             |
| <b>:FORMAT</b>        | Passed to LAYOUTGRAPH and can be any format specification (LATTICE, VERTICAL, REVERSE etc.); defaults to (HORIZONTAL COMPACT REVERSE/DAUGHTERS). In the forest format multiple instances of a function appear on the graph after every calling function and a boxed node indicates the function appears elsewhere on the graph, possibly graphed further. In the lattice format each function gets placed on the graph only once (particularly useful for dynamic graphing, described below), and boxed nodes indicate recursive functions calls. |               |                                     |              |                                             |
| <b>:SEARCHFN</b>      | A function to use to generate the children of a given node. It should return a list whose first item is a list of the children, the other items in the list are ignore. Using this feature, it is possible to graph things other than functions. To graph what files load other files, supply a search function of (LAMBDA (FILE) (LIST (FILECOMSLST FILE 'FILES))) and a file name for the function argument.                                                                                                                                    |               |                                     |              |                                             |
| <b>:ADVISE</b>        | Advises the functions after they are graphed (see <i>Dynamic Graphing</i> below); recognized values are one or both of the following: <table> <tr> <td><b>INVERT</b></td><td>Visually tracks a running program .</td></tr> <tr> <td><b>COUNT</b></td><td>Counts function calls in a running program.</td></tr> </table>                                                                                                                                                                                                                           | <b>INVERT</b> | Visually tracks a running program . | <b>COUNT</b> | Counts function calls in a running program. |
| <b>INVERT</b>         | Visually tracks a running program .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |               |                                     |              |                                             |
| <b>COUNT</b>          | Counts function calls in a running program.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |               |                                     |              |                                             |
| <b>:DELAY</b>         | The delay to use in advised graphs; defaults to 500 milliseconds.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |               |                                     |              |                                             |
| <b>:NAMEFN</b>        | A function to use to generate the node labels on the graph.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |               |                                     |              |                                             |
| <b>:FONT</b>          | The font to use to display the graph; defaults to (GACHA 8).                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |               |                                     |              |                                             |
| <b>:SHAPE</b>         | A boolean that indicates if the window should be shaped to fit the graph; defaults to NIL.                                                                                                                                                                                                                                                                                                                                                                                                                                                        |               |                                     |              |                                             |
| <b>:PRIN2FLG</b>      | A boolean that indicates to use PRIN2 when printing node labels, defaults to NIL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |               |                                     |              |                                             |
| <b>:SUBFNDEFFLG</b>   | A boolean that enables graphing of compiler generated functions; defaults to T.                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |               |                                     |              |                                             |
| <b>:TOPJUSTIFYFLG</b> | Passed to SHOWGRAPH; defaults to NIL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |               |                                     |              |                                             |
| <b>:ALLOWEDITFLG</b>  | Passed to SHOWGRAPH; defaults to NIL.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |               |                                     |              |                                             |

## GRAPH MENUS

The menu that pops up when you left button a function on the graph contains the following items:

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
|------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|-------------------------------------------------------------------------------------------------------------------------------------|----------------|----------------------------------|--------------|--------------------------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>?=</b>        | Print the arguments to the function, if available.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>HELP</b>      | Calls HELPSYS on the function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>FNTYP</b>     | Print the function's FNTYP.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>WHERE</b>     | Do a WHEREIS (with FILES = T) on the function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>EDIT</b>      | Calls the editor on the function if available for editing.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>TYPEIN</b>    | BKSYSBUFs the name of the function into the typein buffer.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>BREAK</b>     | Applies BREAK to the function. Its subitems are: <table> <tr> <td><b>BREAKIN</b></td><td>Breaks the function only in the context of a particular calling function. In lattice format, if the function has more than one function calling it on the graph, the user is prompted to indicate the caller in which to break the function.</td></tr> <tr> <td><b>UNBREAKIN</b></td><td>Undoes BREAKIN.</td></tr> <tr> <td><b>UNBREAK</b></td><td>Applies UNBREAK to the function.</td></tr> <tr> <td><b>TRACE</b></td><td>Applies TRACE to the function.</td></tr> <tr> <td><b>TRACEIN</b></td><td>Traces the function only when called from inside a particular function, like BREAKIN above. Use UNBREAKIN to remove the trace, or else UNBREAK on the window menu.</td></tr> </table> | <b>BREAKIN</b>   | Breaks the function only in the context of a particular calling function. In lattice format, if the function has more than one function calling it on the graph, the user is prompted to indicate the caller in which to break the function. | <b>UNBREAKIN</b> | Undoes BREAKIN.                                                                                                                     | <b>UNBREAK</b> | Applies UNBREAK to the function. | <b>TRACE</b> | Applies TRACE to the function. | <b>TRACEIN</b> | Traces the function only when called from inside a particular function, like BREAKIN above. Use UNBREAKIN to remove the trace, or else UNBREAK on the window menu. |
| <b>BREAKIN</b>   | Breaks the function only in the context of a particular calling function. In lattice format, if the function has more than one function calling it on the graph, the user is prompted to indicate the caller in which to break the function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>UNBREAKIN</b> | Undoes BREAKIN.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>UNBREAK</b>   | Applies UNBREAK to the function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>TRACE</b>     | Applies TRACE to the function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>TRACEIN</b>   | Traces the function only when called from inside a particular function, like BREAKIN above. Use UNBREAKIN to remove the trace, or else UNBREAK on the window menu.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>CCODE</b>     | Calls INSPECTCODE on the function if it is compiled code.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>GRAPH</b>     | Calls GRAPHCALLS to make a new graph starting with function, inherits the original graph's options.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>FRAME</b>     | Inspect the local, free and global variables of the function. These are the last three lists of the CALLS function placed into INSPECT windows. Its subitems are: <table> <tr> <td><b>&gt;FRAME</b></td><td>Like FRAME but for all of the functions on the sub-tree starting at the selected node and only for FREEVARS and GLOBALVARS.</td></tr> <tr> <td><b>&lt;FRAME</b></td><td>Like &gt;FRAME but for all of the functions above the function in the graph, i.e. the FREEVARS and GLOBALVARS in the function's scope.</td></tr> </table>                                                                                                                                                                                                                                       | <b>&gt;FRAME</b> | Like FRAME but for all of the functions on the sub-tree starting at the selected node and only for FREEVARS and GLOBALVARS.                                                                                                                  | <b>&lt;FRAME</b> | Like >FRAME but for all of the functions above the function in the graph, i.e. the FREEVARS and GLOBALVARS in the function's scope. |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>&gt;FRAME</b> | Like FRAME but for all of the functions on the sub-tree starting at the selected node and only for FREEVARS and GLOBALVARS.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |
| <b>&lt;FRAME</b> | Like >FRAME but for all of the functions above the function in the graph, i.e. the FREEVARS and GLOBALVARS in the function's scope.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                  |                                                                                                                                                                                                                                              |                  |                                                                                                                                     |                |                                  |              |                                |                |                                                                                                                                                                    |

Buttoning the graph outside a node give you a menu with these options:

|                |                                                                   |
|----------------|-------------------------------------------------------------------|
| <b>UNBREAK</b> | Does an (UNBREAK), unbreaking all broken functions.               |
| <b>RESET</b>   | Resets the counters for the COUNT option and redisplay the graph. |

#### DYNAMIC GRAPHING

When the ADVISE option is specified with the value(s) of INVERT and/or COUNT, GRAPHCALLS will advise all of the functions on the graph (in the context of their parent) to invert their corresponding node on the graph (as well as delay some to allow it to be seen) and/or follow each function name by a count of the number of times it has been executed. In invert mode, a node remains inverted as long as control is inside its corresponding function and it returns to normal when the function is exited. The

lattice format is best when using the invert feature. Closing the graph window UNADVISEs the functions on the graph.

An example of this is (GRAPHCALLS 'DATE :ADVISE 'INVERT) and then evaluate (DATE).

GRAPHCALLS will not graph or advise any function in the system list UNSAFE.TO.MODIFY.FNS when the advise option is used. Functions which are unsafe to advise should be added to this list.

**CAVEAT PROGRAMMER!** This feature must be used with caution. As a rule, one should not do this to system functions, but only one's own, use WHEREIS as a filter for this. Advising system code indiscriminately will probably crash the machine unrecoverably.

You can, at some risk, interactively break and edit functions on the graph while the code is executing. Also, creating subgraphs of advised graphs will show the generated advice functions not the original functions called, as will creating new graphs of functions in advised graphs. You can create advised graphs of functions already graphed normally on the screen.

## COMMAND WINDOW

| GraphCalls Command Window |         |        |          |       |       |
|---------------------------|---------|--------|----------|-------|-------|
| Command                   | Filters | Flags  | Format   | Depth | Delay |
| <b>Function</b>           | WhereIs | Invert | Lattice  | 0     | 0     |
| Include                   | FGetD   | Count  | Reverse  | 1     | 1     |
| Exclude                   | ExprP   | Shape  | Vertical | 2     | 2     |
| Clear                     | CCodeP  | Edit   | ArgList  | 3     | 3     |
| <b>Graph</b>              | No\     | Prin2  | WhereIs  | 4     | 4     |
|                           |         |        |          | 5     | 5     |
|                           |         |        |          | 6     | 6     |
|                           |         |        |          | 7     | 7     |
|                           |         |        |          | 8     | 8     |
|                           |         |        |          | 9     | 9     |
|                           |         |        |          | 10    | 10    |

(GRAPHCALLSW [REBUILD?])

[Function]

Puts up a command window with menus that will interactively set up calls to GRAPHCALLS. The menus let you set the Invert, Count and Edit flags, select from common filters and formats and set the depth of the graph. You can also change the amount of delay used in the advised functions when doing dynamic graphing. If you specify an advised graph (Invert or Count) and do not specify a WHEREIS filter, you will be asked to confirm with the mouse for your own protection.

More than one item on the filter and flags menus can be selected at a time. Buttoning a selected item on these menus unselects it. The command menu contains the following:

|                 |                                                                                                                |
|-----------------|----------------------------------------------------------------------------------------------------------------|
| <b>Function</b> | Prompts for the name of a function to graph when the <b>Graph</b> item is selected.                            |
| <b>Include</b>  | Adds files or functions to the list of items to allow on the graph, see the Include/Exclude algorithm below.   |
| <b>Exclude</b>  | Adds files of functions to the list of items disallowed on the graph, see the Include/Exclude algorithm below. |

**Clear** Clears all of the settings on the command window to their defaults. Also clears the Include/Exclude lists.

**Graph** Graphs the function by calling GRAPHCALLS with the selected options.

Include and Exclude allow fine tuning of the filter function. If the function passes the filter, then the following are tried until one determines whether or not the function will be on the graph:

If a set of functions has been explicitly excluded, and the function is a member of this set, it will NOT appear on the graph.

If a set of functions has been explicitly included, and the function is a member of this set, it WILL appear on the graph.

If a set of files has been explicitly excluded, and the function is in one of those files, it will NOT appear on the graph.

If a set of files has been explicitly included, and the function is not in one of those files, it will NOT appear on the graph.

The function WILL appear on the graph.

The format menu contains two items that are not passed on to GRAPHER but are used to select alternate NAMEFN options:

**ArgList** Supplies a NAMEFN that will print the function and its arguments (using SMARTARGLIST) as the node label.

**WhereIs** Supplies a NAMEFN that will print the function followed by the file(s) found by doing WHEREIS (with FILES = T) if any .

When the command window is open, middle buttoning a node on a GRAPHCALLS graph will bring up a menu of commands relating to command window and graphs. The menu contains:

**EXCLUDE** Adds the function to the exclude functions list of the command window. This is the only way to exclude system functions which get added to the SYSTEM file exclusion list.

The command window can also be obtained via the background menu. Subsequent calls to GRAPHCALLSW (either directly or via the background menu) will reuse the old command window if there is one. If this window is damaged, and redisplay does not help, then setting *REBUILD?* to T will build a new command window from scratch.

## NOTES

- Function call graphs are constructed using breadth first search but GRAPHER lays out graphs depth first so functions may be expanded in different places on the graph than expected.
- GRAPHCALLS sysloads GRAPHER and MSANALYZE if they are not already loaded.
- In dynamic graphs, variables caused by advising show up in the frame inspections.
- The global variable GRAPHCALLS.DEFAULT.OPTIONS contains all of the defaults for GRAPHCALLS keywords, in property list format.



---

---

## GraphGroup

---

---

By: Nick Briggs (Briggs.pa@Xerox.com)

For internal use only.

The GraphGroup package contains functions for generating a graph showing the structure of a Grapevine distribution list.

### DESCRIPTION

There is a single user-callable function:

(GraphGroup *GroupName InfoStream LayoutOptions ExpandNSGroups*) [Function]

*GroupName* is the group to be graphed.

Because it is rather slow tracking down the whole structure of a group, if *InfoStream* is non-nil GraphGroup will print on *InfoStream* a trace consisting of: a "[" followed by the number of entries in the group each time it starts a (sub-) group, a "." or "?" each time it identifies an individual, and a "]" at the end of a group. T is a good choice for this parameter.

*LayoutOptions* is a list of options, in property list format, to be passed to the Grapher function LAYOUTSEXPR. Properties recognized are: FORMAT, BOXING, FONT, MOTHERD, PERSONALD, and FAMILYD. See the Grapher documentation for a description of how to use these options. In most cases you will get a satisfactory graph by leaving this parameter NIL. Should you be planning to hardcopy the resulting graph on an NS printer it is recommended that if you specify a font you use an NS font (such as Classic or Modern) so that the layout spacing is done correctly.

If *ExpandNSGroups* is T, then GraphGroup will attempt to trace into the NSworld when it finds Grapevine RNames that it recognizes as having pseudo-registries that are really NS domain/organizations.

---



---

## GREP

---



---

By: Larry Masinter (Masinter.pa@Xerox.com)

Requires: BSEARCH

### INTRODUCTION

like FGREP of Unix: searches for strings in files.

(GREP *STRS FILES*) [Function]

STRS is a string or a list of strings. FILES is a file or a list of files. Searches for the given string(s) in the given file(s), showing each line.

(PHONE *name*) [Function]

Calls (GREP name PHONELISTFILES). PHONELISTFILES is initialized to NIL. (The PARC init file resets it to point to the PARC phone list.)

For example,

```
(GREP (QUOTE ED) (QUOTE {INDIGO}<REGISTRAR>PARCPHONELIST.TXT))
```

will print:

```
(from {INDIGO}<REGISTRAR>PARCPHONELIST.TXT;3)
4183 <Endicott>, Fred 35-1354
4435 <Fiala>, Ed 35-2166
4598 <RKennedy>, Ray 34-78
4839 <McCreight>, Ed 35-2146
5759 Pedersen, Jan 32-202
4818 Satterthwaite, Ed * 35-2174
MES Solcz, Edward J. 8* 348-1214
ATA Wahlenmeier, Fred 887-4018
```

---



---

## GRID-ICONS

---



---

By: sML (Lanning.pa@Xerox.com)

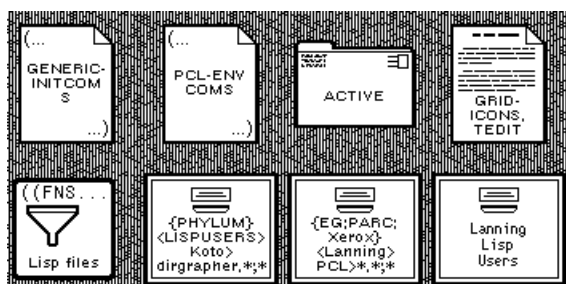
Last edited: September 14, 1987

### INTRODUCTION

Grid-Icons provides the Lisp user with a set of default window icons that resemble those found in the Viewpoint system. There is an option that the user can set to force these icons to be positioned on a grid, instead of the unrestricted positioning allowed by Lisp.

### USING GRID-ICONS

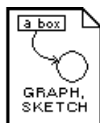
All that is required is loading the file GRID-ICONS. When the file is loaded, it redefines a number of standard window icons in the system.



### REDEFINED ICONS



[TEdit icon]

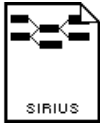


[Sketch icon]

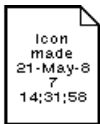
Note that GRID-ICONS not only redefines TEdit and Sketch icons, it also changes the way that the icon title is computed: the host and directory information is removed, so that only the name and extension remain.



[SEdit icon]



[Loops Browser icon]



[Default icon]

In Lyric, it is possible to redefine the standard icon used by the system. GRID-ICONS uses this, and redefines the standard system icon in LYRIC.



[Lafite mail folder icon]

## NEW ICONS

GRID-ICONS defines a handy icon for accessing the list of files that have been loaded into you system.



[Lisp files icon]

Buttoning on this icon will pop up a menu of all loaded files (as determined by the value of the variable FILELST). Selecting a file from this menu will open up an editor on the COMS of that file. There is an additional item on the menu, "\*\* New file \*\*", that can be used create a new file, and then edit its COMS. This icon window is stored in the variable LOADED-FILES-ICON-WINDOW.

## USER FUNCTIONS

The user can declare that any given window stick to grid positions.

(GRID-WINDOW window)

[Function]

Causes window to pay attention to ENFORCE.ICON.GRID; if the value of ENFORCE.ICON.GRID is true, the window will make sure that it is centered on a grid location. By default the spy button and icons produced by the ICONW and TITLEICONW functions pay attention to ENFORCE.ICON.GRID.

## VARIABLES THAT CONTROL GRID-ICONS

There are a few variables that control how GRID-ICONS works.

ENFORCE.ICON.GRID [Variable]

If ENFORCE.ICON.GRID is true, window icons (and any window declared "gridded" by the GRID-WINDOW function) will be restricted to be positioned on a grid. The default value of ENFORCE.ICON.GRID is NIL.

IGNORE.ICON.GRID [Window property]

The IGNORE.ICON.GRID window property provides a way to control icon gridding on an icon-by-icon basis. If the IGNORE.ICON.GRID window property of an icon is true, the icon will not be restricted to grid positions. This window property is checked only if ENFORCE.ICON.GRID is true.

ENFORCE.ICON.REGIONS [Variable]

You can enforce icon gridding in individual regions of the screen by using the variable ENFORCE.ICON.REGIONS. If the value of ENFORCE.ICON.GRID is true, and the icon does not have a IGNORE.ICON.GRID window property, the proposed new position for the icon is tested against the value of ENFORCE.ICON.REGIONS. If ENFORCE.ICON.REGION is NIL, gridding is enforced as described above. Otherwise, ENFORCE.ICON.REGION should be a list of regions; gridding will be enforced only if the proposed position is within one of these regions. The default value of ENFORCE.ICON.REGIONS is NIL. [Thanks/blame to Ramana Rao for this.]

ICON.SIZE [Variable]

ICON.SIZE specifies the maximum size of the icons, for use in computing the grid positions of icons. It is a cons of the maximum width and the maximum height. The default value is (85 . 85), which is the "correct" value for the icons defined in this utility.

ICON.SPACING [Variable]

ICON.SPACING specifies the gap between icons, for use in computing the grid positions of icons. It is a cons of the horizontal gap and the vertical gap. The default value is (5 . 5).

GRID.OFFSET [Variable]

GRID.OFFSET specifies origin of the icon grid. The default value is (0 . 0).

DEFAULTICONFONT [Variable]

The value of DEFAULTICONFONT is the default font used by the system when printing titles in icons. Since the icons defined in GRID-ICONS tend to be smaller than the original icons, you might want to use a slightly smaller font than the default. Personally, I recommend setting DEFAULTICONFONT to (FONTCREATE '(HELVETICA 8)).

---

---

## Hanoi

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

### INTRODUCTION

Ancient graphics demo, upgraded to be idle hack. Adds Hanoi to list of idle displays.

### OPERATION

(HANOI *NRINGS WINDOW FONT ONCE*)

[Function]

Will display in *WINDOW* (or HANOIWINDOW, created first time) a towers-of-hanoi problem and solve it. It periodically blocks so you can run it as a background process. *NRINGS* is the number of rings. If *NRINGS* is a list it is the labels printed on the rings in font *FONT*. It conforms to the window shape if you reshape it. It will run indefinitely unless *ONCE* is non-NIL.

---

---

**HARDCOPY-RETAIN**

---

---

By: Herb Jellinek (Jellinek.pa@Xerox.com)

This document was last edited on December 29, 1987.

**Introduction**

This module adds a "To a file and a printer" option to the Hardcopy item on the system window and background menus. This option sends the contents of a window or screen region to a printer, and simultaneously creates a file containing a printer-ready copy of it (e.g., an Interpress master); this makes it easy to run off multiple copies later.

**Using it**

When loaded, HARDCOPY-RETAIN calls the function `xcl-user::install-option` to side effect `il:|WindowMenuItems|` and `il:|BackgroundMenuItems|` and to set `il:|WindowMenu|` and `il:|BackgroundMenu|` to nil, thus forcing them to be recalculated.

---



---

## HASHBUFFER

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: HASH

HASHBUFFER combines *hash files* with *hash arrays* in order to improve hash file performance when keys are accessed multiple times. This module also defines two functions for moving data between hash files and hash arrays.

The functions below are used in place of the hash file routines. When a hash file is opened, a hash array is created, of a complimentary size. When requests for keys are made, the array is searched, and if a value is found, it is returned. If a value is not found, the file is searched and if a value is found there, it is stored in the array and returned. If a value is not found, a marker is put in the array so that the file is not searched again.

(OPENHASHBUFFER *FILE* [*ACCESS MINKEYS OVERFLOW HASHBITSFN EQUIVFN*]) [Function]

Opens an existing hash file and returns a hash buffer datum which must be given to the other hash buffer functions. Only the *FILE* argument is required; the *MINKEYS* argument is used for the size of the hash array and if not supplied the size of the hash file is used. Setting *MINKEYS* smaller than the size of the hash file allows a fast, small hash array window onto a larger, slower hash file. The *OVERFLOW*, *HASHBITSFN* and *EQUIVFN* arguments are passed to *HASHARRAY*.

(CREATEHASHBUFFER *FILE* [*VALUETYPE ITEMLength #ENTRIES ^ OVERFLOW HASHBITSFN EQUIVFN*]) [Function]

Like *OPENHASHBUFFER* but creates a new hash file. The *FILE*, *VALUETYPE* and *ITEMLength* arguments are passed to *CREATEHASHFILE*; the *OVERFLOW*, *HASHBITSFN* and *EQUIVFN* arguments are passed to *HASHARRAY*. The *#ENTRIES* argument is used for both the file and array.

(CLOSEHASHBUFFER *HASHBUFFER* [*FILEONLY?*]) [Function]

Closes the hash file and sets the hash array to NIL so that it can be reclaimed. If *FILEONLY?* is non-NIL then only the hash file is closed, the hash array will be left alone.

(GETHASHBUFFER *KEY HASHBUFFER*) [Function]

(PUTHASHBUFFER *KEY VALUE HASHBUFFER*) [Function]

Retrieve and store *VALUE* for *KEY* in the hash buffer. If the hash file is only open for input, then storing a key will only affect the hash array. If the hash file is open for output, then storing a key will put it in both the hash array and hash file. If *VALUE* is NIL, then a delete is performed.

(HASHARRAY.TO.HASHFILE *HASHARRAY HASHFILE* [*TESTFN*]) [Function]

Uses *MAPHASH* to move the contents of *HASHARRAY* into a hash file. If *HASHFILE* is a file name, *CREATEHASHFILE* is called; if *HASHFILE* is an open hash file datum, it is used and left open. *TESTFN*, if supplied, is called before each *PUTHASHFILE* on (*KEY VALUE HASHARRAY HASHFILE*) and if it returns non-NIL, the key and value are copied to the file.



(HASHFILE.TO.HASHARRAY *HASHFILE* [*HASHARRAY TESTFN*])

[Function]

Uses MAPHASHFILE to move the contents of *HASHFILE* into a hash array. *If HASHARRAY* is not supplied a new hash array is created. *TESTFN* is called before each PUTHASH on (KEY VALUE HASHFILE HASHARRAY) and if it returns non-NIL, the key and value are copied to the array.

---



---

## HASHDATUM

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: HASH

HASHDATUM facilitates storing random Envos Lisp datatypes on *hash files* using the hashed text feature of the HASH Lisp Library module. The module defines functions which access an item on a hash file as a stream of bytes using user supplied input and output functions. Since the items are stored using text hashing, when rehashing or copying of the file occurs, the data portion of the file is copied correctly.

(GETHASHDATUM KEY HASHFILE READFN) [Function]

(PUTHASHDATUM KEY DATUM HASHFILE PRINTFN) [Function]

Use *READFN* and *PRINTFN* to store and retrieve *DATUM* on *HASHFILE*. The *READFN* takes a stream as its argument, the *PRINTFN* takes the *DATUM* and a stream. The *put* function returns the hash file text pointer record which contains two byte pointers that indicate where the datum begins and ends on the file. The *get* function returns the result of the *READFN*.

The following macros and functions are also defined using the above functions:

(GETHASHGRAPH KEY HASHFILE) [Macro]

(PUTHASHGRAPH KEY GRAPH HASHFILE) [Macro]

Use *GRAPHER* functions *READGRAPH* and *DUMPGRAPH* to store *GRAPH* on *HASHFILE* under *KEY*.

(GETHASHBITMAP KEY HASHFILE) [Macro]

(PUTHASHBITMAP KEY BITMAP HASHFILE) [Macro]

Use *READBITMAP* and *PRINTBITMAP* to store *BITMAP* on *HASHFILE* in a text format.

(GETHASHBINARYBITMAP KEY HASHFILE) [Macro]

(PUTHASHBINARYBITMAP KEY BITMAP HASHFILE) [Macro]

Use *READBM* and *WRITEBM* from *BITMAPFNS* to store *BITMAP* on *HASHFILE* in a binary format.

(GETHASHTEDIT KEY HASHFILE [WINDOW PROPS]) [Function]

(PUTHASHTEDIT KEY TEXTOBJ HASHFILE) [Macro]

Use *OPENTEXTSTREAM* and *TEDIT.PUT.PCTB* from *TEDIT* to store *TEXTOBJ* on *HASHFILE*, preserving both the text and formatting information. *WINDOW* and *PROPS* are optional and are passed to *OPENTEXTSTREAM*. If the *WINDOW* argument is not supplied, the result of the *get* function can be passed to *OPENTEXTSTREAM* along with a window to display the text.

(GETHASHUGLY KEY HASHFILE) [Macro]

(PUTHASHUGLY KEY UGLYVAR HASHFILE) [Macro]

Use HREAD and HPRINT to store random data, like menus, on *HASHFILE*.

---



---

## HEADLINE

---



---

By: D. Austin Henderson, Jr. (AHenderson.pa@Xerox.com)

Last revised: April 1, 1986

**HEADLINE** contains functions for creating and closing windows which contain headlines ("headline windows").

(**HEADLINE** *PHRASE FONT POSITION ALIGNMENT*) [Function]

Creates a headline window with *PHRASE* printed in font *FONT* at position *POSITION* aligned as per *ALIGNMENT*; the window is just large enough to hold the headline. *PHRASE* is any Lisp object. *FONT* defines a font as per FONTCREATE (eg. (TIMESROMAN 18 BOLD) ); if NIL, TimesromanD 36 is used. *POSITION* is a position giving the reference point for placing the window; if NIL, the user is given a chance to position the window with MOVEW. If *POSITION* is given, *ALIGNMENT* gives the alignment of the window with respect to *POSITION* as (xalignment . yalignment) where xalignment is one of LEFT, CENTER, or RIGHT and yalignment is one of BOTTOM, CENTER, or TOP; for convenience, if Position is CENTER then it is taken to mean (CENTER . CENTER), etc.

(**HEADLINE.ARRAY** *TITLES ALIGNMENT SEPARATION POSITION*) [Function]

Creates a set of vertically arranged headline windows. *TITLES* is a list of (phrase font) sublists where phrase and font are as in Headline. *ALIGNMENT* is one of LEFT, CENTER, or RIGHT, indicating how the windows are aligned with each other; defaults to CENTER. *SEPARATION* indicates the spacing between the bottoms of the windows; defaults to 70. *POSITION* indicates where the top (first) of the windows is to appear; defaults to somewhere near the top center of the screen.

(**BILLBOARD**) [Function]

Identical to HEADLINE.ARRAY, left in for backward compatibility.

(**BANNER** *PHRASE FONT POSITION ALIGNMENT*) [Function]

Same as HEADLINE except it prints the phrase vertically.

(**BANNER.ARRAY** *TITLES ALIGNMENT SEPARATION POSITION*) [Function]

Same as HEADLINE.ARRAY except it prints the phrases vertically, left to right.

(**CLOSE.HEADLINES**) [Function]

Closes all the active headline windows.

---



---

## HELPSYS

---



---

By: Doug Cutting (Cutting.PA@Xerox.COM)

additions (CLHS, REPO) by Imm

Uses: DINFO, HASH

The second part of This document last edited on October 7, 1987; now updated August 20.

If given a symbol in the Common Lisp package, the functions below will look up the symbol in the CLHS Index ; these additions work for ? while typing in and the 'man' command.

(CLHS.LOOKUP entry) [Function]

This reads in the CLHS and finds the URL and fragment of the CLHS page there is one. It uses the opener (indicated by CLHS.OPENER) to show the URL if it can.

(CLHS.OPENER) [Function]

This guesses based on your environment what Shell Command can be used to display the CLHS page. One of "wslview" (For Windows with WSL installs), "open" (For Mac OS, "git web--browser" (for various kinds of linux systems or "lynx" (using xterm if available, or CHAT(SHELL) to run lynx. ( a fast limited text-only browser). On debian-based systems "sudo apt install lynx" will install it.

CLHS.OPENER [Variable]

If set, will be used; t the opener you prefer from the choices above.

(REPO.LOOKUP keyword) [Function]

In addition, the lookup program has been extended to also look up functions and variables in Library and LispUsers packages, by looking up in the WHEREIS database and, if there is a TEdit or TXT file for that package, opening that. **This only works for functions and variables that are on the same file (according to WHEREIS) as file name of the tedit (or TXT) file, or follow the MAIN-SUBPART convention.**

=====

## INTRODUCTION

HelpSys is the interface to the online version of the *Interlisp-D Reference Manual*. It provides both sequential perusal of the manual and random access lookup and display of index entries.

### Interlisp-D Reference Manual DInfo Graph

Helpsys uses the **DInfo** library package as a means of accessing the *Interlisp-D Reference Manual*. Once you have loaded Helpsys, selecting **DInfo** from the **Background Menu** will raise a menu which will contain an item named **Interlisp-D Reference Manual**. Selecting this item will start DInfo on the DInfo graph for the manual. (Note: **IRM.HOST&DIR** must be set correctly before HelpSys will work;

see **Installing HelpSys**, below). See the documentation for DInfo for more information on this package.

?<cr>

[EXEC macro]

Helpsys enables the EXEC and TEXEC macro ?<cr>. Typing this (a question-mark followed by a carriage-return) will cause the CAR of the form currently being typed to be looked up. This works much like the more familiar ?=<cr> macro which displays argument lists.

### Lookup!

[DInfo Command]

Selecting Lookup! from DInfo's menu in the **IRM Dinfo Graph** will prompt for and look up a term.

## INSTALLING HELPSYS

Helpsys requires a number of files which should be provided with the LispUsers release. These are the 31 chapter files (CHAP\*.TEDIT) the top node file (IRMTOP.TEDIT), the file containing the DINFOGRAPH (IRM.DINFOGRAPH) and the index hash file (IRM.HASHFILE).

To install HELPSYS you must copy all these files to one directory, and set the variable IRM.HOST&DIR to the name of this directory.

IRM.HOST&DIR

[Variable]

This determines where HelpSys will look for the manual files and hash file. This should be set in your site init file. As file servers can often be quite loaded down, HelpSys will work much faster if you cache these files on the local disk.

IRM.HASHFILE.NAME

[Variable]

If this is NIL, HelpSys will look for the hash file as IRM.HASHFILE on IRM.HOST&DIR, otherwise it uses the value of this variable. Note that the hash file *must* be on a random access filing device (eg. the local disk).

## PROGRAMMERS' INTERFACE

(IRM.LOOKUP *KEYWORD TYPE GRAPH SMARTFLG*)

[Function]

This is the primary function of HelpSys. If *TYPE* is specified, then the primary manual entry for *KEYWORD* of *TYPE*, or the first if there is no primary, will be displayed in *WINDOW*. If *TYPE* is not specified, then a pop-up menu will be raised containing all the manual entries for *KEYWORD*. Primary entries are marked with stars (\*) on either end of the item. *GRAPH* should be the IRM Dinfo graph, and defaults to IRM.DINFOGRAPH.

(IRM.SMART.LOOKUP *KEYWORD GRAPH*)

[Function]

Uses wild card matching if \*'s are in *KEYWORD* (\* matches any substring) or tries spelling correction. A pop-up menu is raised if more than one wild card match is found. Note that the first time a \* appears in a *KEYWORD*, HelpSys will need to load the list of possible keywords for matching against, and only after this list has been loaded will spelling correction be enabled. This is the function called by the Lookup! button in DInfo's FreeMenu. *GRAPH* again defaults to IRM.DINFOGRAPH.

(IRM.RESET)

[Function]

Will reset HelpSys so that everything will be reinitialized.

---

---

## H

---

---

By: Roberto Ghislanzoni (Roberto Ghislanzoni:MKT:RXI)

### INTRODUCTION

The package H allows the user to augment the Interlisp-D environment with an Horn Clauses Theorem prover: in it it's possible to call semantic attachments (SAs) to Lisp (i.e. lisp functions) as FOL does.

### A BRIEF HISTORY

The original idea comes from Chester (Chester,1980), (Chester, 1980): it was revised by Simmons (Simmons, 1984). A prototype of this program was developed at the University of Milan by Vieri Samek Ludovici, Giorgio Tornielli and Roberto Ghislanzoni using VLisp. Currently it is offered on Interlisp-D environment.

### USE OF PACKAGE

Load the file H-LOAD.DCOM. This loads all necessary files. On your machine, you have now two environments: the H developer and the H deliver. In H developer you can plan, construct and edit your HKBs (H Knowledge Bases), and use it from Lisp executive window, simply by H.loading the HKBs created. H is a very good logic paradigm for LOOPS: from executive window, it is possible to have as many calls to the prover as you need.

### THE H DEVELOPER

From the background menu chose H: this offer you a window in which is active a *read-prove-print* loop. Open as many windows as you want: all HKBs are local to the windows. The H Control Window has the following entries:

- **Show Profile** : This shows in the Prompt Window the current settings for environment bound to the window; the MODE of demonstration (FIRST: stop to the first goal proved; ALL: reach all goals: T: interactive mode); the LIMIT of the search tree; the TRACING of prover: the PMTRACING, that shows the pattern matching at work.
- **Show(Axiom)** : shows the clauses that define a predicate; the submenu allows to see the lambda-definition of a semantic attachment. The choice is made from a pop-up menu.
- **Delete(Axiom)** : this erases from database the clauses choosen by the user. Erases also the SA from the submenu.
- **Edit(Axiom)** : it allows to create and edit both predicates and SA, using the standard DEdit facilities.
- **SetLimit** : sets the limit of the search tree.
- **Mode** : chose the mode of demonstration.
- **Shortform** : it enables or disabled the control for occurrence, so it is not possible to unify the variables already bound in that piece of unification to themselves.
- **Trace** : it enables in a separate window the tracing of demonstration.
- **Trace PM** : it enables the tracing of the unifier.
- **LoadHKB** : loads a H Knowledge base in the environment: the name of KB is shown beside the window; do not provide the .HKB extension to the name: the system does it.
- **SaveHKB** : saves the current environment in a KB: don't provide extension.
- **EraseEnv** : erases the entire environment.

— **Exit** : exits and closes window.

All the windows that H uses ("Show window", "Trace window", "PM Trace window") has the middle button capability in order to allow to dribble into a file everything that is printed in the window; it is very similar to CHAT's dribble option.

## THE H DELIVER

There are a lot of functions available from Lisp Executive that allow the programmer to use the HKBs previously created:

(H.erase) [Function]

Erases all environment (i.e., predicates and SAs) previously loaded.

(H.load *database*) [Function]

Loads H database into environment.

(H.save *database*) [Function]

Saves all current environment into a file

(H.? *pred1 ... predN*) [NLAMBDA Function]

Start the demonstration of the predicates specified. Return the list of predicates proved with the variables of the call set .

(H.all *variables conjs*) [Function]

Returns the list of all specified variables that satisfy the predicate(s). Remember that variables must begin with a ':' (semicolon).

(H.any *howmany variables conj*) [Function]

Returns *howmany* instantiation values of variables that make true the predicate(s).

(H.attach *foo lambda-expression*) [NLAMBDA Function]

Defines a SA named *foo* to be the value of the LAMBDA-expression written as the second argument.

(H.addaxiom *axioms-list*) [Function]

Adds new axioms to the existing one for that predicate.

(H.axiom *axioms-list*) [Function]

Defines new axioms for the predicate. Deletes the previous ones.

(H.del *axiom*) [Function]

Deletes a single axiom from the database; the other axioms for that predicates are not touched.

(H.show) [NLAMBDA Function]

Shows the definition of the given predicates.

(H.the *variable conjs*) [Function]

Returns only one value for the variable that satisfies the goal.

(SET.H.MODE *mode num*) [Function]

Set the mode of demonstration: mode may be one of atoms 'first', 'all', 'interactive'. If atom 'limit', then you must provide the number of depth (default 200).

## H PRIMITIVES

Both environments have three important primitives SA:



(set var *expr*)

Sets in the current level of unification the variable to the value of the expression *expr*.

(assert *axiom*)

Assert in the database the given axiom. without erasing the old ones.

(delete *axiom*)

Delete in the database the given axiom.

In the system also is present the cut facility ( ' / '), that has similar behaviour as PROLOG cut ( '! ' ).

## H DEMOS

An example of axioms may be this:

```
(((append () :a :a))
 ((append (:a . :b) :c (:a . :d)) < (append :b :c :d)))
```

You can call from H-executive:

```
(append (1 2 3) (4 5 6) :d)
```

that returns

```
(((append (1 2 3) (4 5 6) (1 2 3 4 5 6)))
```

Also, if you have in your database:

```
(((A 1))
 ((A 2)))
```

and

```
(((B 3))
 ((B 4)))
```

you can call from TTY exec:

```
(H.? (A :1)) --> (((A 1)))
```

or

```
(H.? (A :k) (B :o)) --> (((A 1) (B 3)))
```

Moreover:

```
(H.all ' :k ' ((A :k))) --> (2 1)
(H.all ' (:a :b) ' ((A :a) (B :b))) --> ((2 4) (2 3) (1 4) (1 3))
(H.any 2 ' :j ' ((A :j))) --> (2 1)
```

For demo. load the HKBs H-MAZE,H- BLOCKS and try the following:

```
(showworld)
```

that shows you the block world situation: try then

```
(please (put the red cube on the blue one))
 (please (pick up cube2))
```

and so on

The other examples solve for you the maze problem and other interesting things: discover by yourself how they work ...

## REFERENCES

Chester D., Using HCPRVR, Department of Computer Sciences, University of Texas, Austin, June 1980

Chester D, HCPRVR: a logic program interpreter in Lisp, proc AAAI, Department of Computer Sciences, University of Texas, Austin, June 1980

Simmons R.F., Computations from the English, Prentice Hall, New Jersey, 1984

---

---

## HISTMENU

---

---

Original Fugue version By: Danny Bobrow (Bobrow.PA@Xerox.COM)

2020 Medley 3.5 clean sheet reimplementation By: Michele Denber (mdenber@gmail.com)

2022 Medley merge of *Michele Denber implementation* into the *Original Danny Bobrow version* By:  
Matt Heffron (heffron@alumni.caltech.edu)

### INTRODUCTION

HISTMENU is a Xerox Lisp (Medley, Lyric, or Koto) program that provides quick access to commands recently typed in the Exec window. ~~The original HISTMENU was written by Danny Bobrow but seems to have been lost over time. This version was reverse engineered by Michele Denber from a running instance in an old sysout.~~ The source for the Original Danny Bobrow version was located in the archive of Matt Heffron (with a slight? change that cannot be identified). That version is the *basis* of this implementation.

### OPERATION

Load HISTMENU.LCOM from your local Lispusers directory. Then call

```
(HistoryIcon [histMenuLength] [histMenuPosition] [histIconPosition])
```

or

```
(HistoryMenu [histMenuLength] [histMenuPosition])
```

where

`histMenuLength` optionally specifies the number of commands you want displayed. Default is the value of `HistDefaultSlice` which is set to 30 when HISTMENU is loaded.

`histMenuPosition` optionally specifies a position on the display to place the menu. Default is to place it using the mouse.

`histIconPosition` optionally specifies a position on the display to place the graphic icon for the



History Window. Default is to place it using the mouse.

Calling `HistoryMenu` will create the window with only a *standard* window title style icon.

Clicking Left on any menu item will REDO that command.

Clicking Middle brings up a pop-up menu that lets you issue one of the four Programmer's Assistant commands REDO, FIX, UNDO, ?? plus an option Delete which removes that item from the History Menu.

Clicking Right bring up a pop-up menu with the three standard options Bury, Move, and Shrink plus an option Update which updates the entries in the History Menu to again show the last n commands from the Exec window.

HistMenu works by using BKSYSBUF to "type" the REDO, FIX, UNDO, or ?? command with the associated history event number, followed by a CR, into the window that currently is accepting input. If there was already something typed in that input, the command would be appended/inserted at the point of the caret and the command will not do what would be expected.

Michele Denber made one change to the original HISTMENU. In the original, it would send the command to whatever window had the keyboard focus. So if you were in a TEdit window, clicking a HISTMENU item would place "REDO 354" in the TEdit window. This version only allows commands to go into a selected Exec window. When merging Michele's implementation into the Original version, I (Matt Heffron) discovered that this change makes the History Menu unusable in contexts like a Break or Error window unless it is under an Exec process. To remedy this, I added a global variable HistMenuExecOnly which must be non-NIL to enable the checking for restricting the input to an Exec window. HistMenuExecOnly defaults to NIL.

## OPTIONS

---



---

## HPGL

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

HPGL defines a Lisp *image stream* type that generates output for plotters (and other devices) which use the *Hewlett-Packard Graphics Language*. The module does not define any user functions, the HPGL streams are accessed via OPENIMAGESTREAM and the hardcopy functions.

### PLOTTERS

Some plotters which use HPGL either as their primary language or as an optional extra:

|                                  |                                                  |
|----------------------------------|--------------------------------------------------|
| Hewlett-Packard (most)           | Epson America HI-80 (option)                     |
| Facit 4551                       | Western Graphtec MP1000 and FP5301 (option)      |
| Gould Color-writer 6120 and 6320 | Houston Instrument DMP-29 (option)               |
| Taxan 710                        | Nicolet Zeta 8 (option)                          |
| IBM 7371                         |                                                  |
| Roland DYP-880 and 980           | (source: PC, Volume 4, Number 26, December 1984) |

The file extensions HPGL and PLOT are recognized by the system as plotter output file types.

### OPTIONS

The driver accepts the following in the OPTIONS argument to OPENIMAGESTREAM:

|            |                                                                                                                                                                                                        |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SCALE      | Image scaling; value should be a POSITION record which indicates where the second scaling point should be placed (the initial scaling point is at 0,0). By default, uses (SCREENWIDTH . SCREENHEIGHT). |
| ROTATE     | Paper rotation; value should be 0 or 90, defaults to <i>landscape</i> plotting (0).                                                                                                                    |
| PAPER      | Paper size; value is a small integer, the HP 7475A accepts 3 (A3) or 4 (A4).                                                                                                                           |
| TERMINATOR | Label terminator character; value should be a character, the default is '^A'.                                                                                                                          |
| VELOCITY   | Pen velocity; plotter specific.                                                                                                                                                                        |

### IMPLEMENTATION

The driver was implemented using an HP 7475A plotter but the plotter output conforms to the more restrictive HP 9872 syntax to be more widely applicable. The driver uses the following variables which may need adjustment for other types of plotters:

|                      |                                                                                                                        |
|----------------------|------------------------------------------------------------------------------------------------------------------------|
| HPGL.FONTS           | An ALST of font names and (small integer) plotter font numbers.                                                        |
| HPGL.OPTIONS         | An ALST of plotter specific options that can be passed to OPENIMAGESTREAM and the corresponding HPGL command to print. |
| HPGL.DASHING         | An ALST of HPGL line types (small integers) and dashing lists.                                                         |
| HPGL.FONT.EXPANSIONS | An ALST of font face expansions (REGULAR, COMPRESSED and EXPANDED) and the relative scale of each.                     |
| HPGL.TERMINATOR      | The default end of instruction terminator character, initially '^'.                                                    |

|                      |                                                                                                                                            |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| HPGL.SEPARATOR       | The default parameter separator character, initially ','.                                                                                  |
| HPGL.TEXT.TERMINATOR | The default end of label terminator character, initially '^A'.                                                                             |
| HPGL.CHORD.ANGLE     | The chord angle used by the circle and arc instructions. Defaults to NIL which causes the plotter's default to be used.                    |
| HPGL.PATTERN.LENGTH  | The default pattern length for the hardware dashing. Defaults to NIL which causes the plotter's default to be used.                        |
| COLORNAMES           | System variable used to convert between RGB triples and pen numbers. The order of entries affects the pen number to color correspondences. |

## DASHING

To minimize the complexity of the driver and maximize the speed of plotting, for operations other than DRAWLINE, the driver only uses the built-in dashing types of the plotter. The correspondences between the dashing style and HPGL line type number are kept in the HPGL.DASHING variable which can be modified or extended for plotters with different dashing styles than those displayed below:

|   |       |           |           |           |           |                |
|---|-------|-----------|-----------|-----------|-----------|----------------|
| 1 | .     | .         | .         | .         | .         | (1 49)         |
| 2 | _____ | _____     | _____     | _____     | _____     | (25)           |
| 3 | _____ | _____     | _____     | _____     | _____     | (35 15)        |
| 4 | _____ | . _____   | . _____   | . _____   | . _____   | (39 5 1 5)     |
| 5 | _____ | - _____   | - _____   | - _____   | - _____   | (35 5 5 5)     |
| 6 | _____ | - - _____ | - - _____ | - - _____ | - - _____ | (25 5 5 5 5 5) |
|   | _____ |           |           |           |           | NIL            |

If the driver is loaded after SKETCH, the dashing types are added to SKETCH's dashing menu .

---

---

**IDEASKETCH**

---

---

By: Richard Burton (Burton.Pa@Xerox.com)

Uses: Sketch

This document last edited on 26-Jan-89 19:09:56.

**INTRODUCTION**

Idea Sketch is an adaptation of Sketch that is designed to allow easy jotting down and laying out of ideas. An idea sketch window is actually a sketch window with a command menu tuned for manipulating text and defaults set up for connecting text (i.e. arrowheads are added.)

Selecting the "More Menu" items gets the standard sketch menu.

---



---

## IDLEHAX

---



---

By: Larry Masinter (Masinter.pa@Xerox.com) with contributions by various others.

### INTRODUCTION

This module contains a couple of random demonstration programs, useful as "Idle programs", callable from the background menu. The Idle display options includes Lines Warp-Out Radar Triangles RandAngles Polygons Bubbles and Kaleidoscope.

These are implemented by the following functions:

(POLYGONS *W NOBLOCK TIMER*) [Function]

Calls (POLYGONS) or (POLYGONS window) to perpetually draw polygons in the given window (it (re)uses POLYgonsWINDOW if argument is NIL). To run in the background, you can ADD.PROCESS((POLYGONS (CREATEW). Controlled somewhat by the global parameters POLYGONMINPTS (minimum number of vertices), POLYGONMAXPTS (maximum number of vertices), POLYGONSTEPS (number of steps between min and max), and delays POLYGONWAIT (time between different polygons) and POLYGONWAIT2 (delay between initial display of beginning and end and the movement phase.)

If NOBLOCK is T, it doesn't block at all (runs after but can't run in background.) If TIMER is given, then POLYGONS will stop after TIMER is expired. (Used by the demo system.)

(LINES *W N LCNT STEPS ODDSTEP*) [Function]

Similar to POLYGONS in controls, but draws perpetually changing form using line draw. W defaults a "demo window", but is the window on which the display is drawn, N is the number of endpoints (e.g., 2 draws lines, 3 draws triangles, 7 draws 7-segment figures), LCNT is the "number of lines on the screen at any one time", STEPS is the number of lines to draw between start and end (the higher this number, the closer together the lines are), and ODDSTEP is a flag: if T, then the odd endpoints remain the same every other iteration (try (LINES NIL 3 1 40 T).) The background RandAngles means: (LINES W (RAND 3 7) (RAND 1 16) (RAND 25 100)), while Triangles is (LAMBDA (W) (LINES W 3 1 40)), etc.

(BUBBLES *WINDOW*) [Function]

Perpetually draws circles. Controlled by BUBBLECNT, which is read at startup as the number of circles visible at any one time.

(KAL *W PERIOD PERSISTENCE*) [Function]

Borrowed from the KAL LispUsers package: draws a random symmetric pattern of dots. Pretty. Period affects the style of display, while PERSISTENCE affects how many dots are on the screen at once.

(WARP *W*) [Function]

Draws a sequence of circular patterns that resemble piles of sand. Or not; you decide.

POLYGONS, LINES and BUBBLES adjust themselves to the size of the window, so you can reshape the window in the middle of the demo.



---

---

**INSPECTCODE-TEDIT**

---

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)  
Beckman Instruments, 2500 Harbor X-11  
Fullerton, CA. 92634  
(714) 961-3128



The **INSPECTCODE-TEDIT** package advises the INSPECTCODE facility to have some extended capabilities when the TEDIT and GRAPHCALLS packages are loaded (i.e. it uses TEDIT and GRAPHCALLS).

If TEDIT is not defined, then the standard INSPECTCODE will be used. If TEDIT is loaded, then a read-only TEDIT/INSPECTCODE window will be opened, and will have a special INSPECTCODE menu for **LEFT** or **MIDDLE** buttoning in the titlebar. All of the options, except for **Quit**, in this menu use the current selection in the window. You make selections with the mouse buttons in the standard TEDIT ways. The options in the INSPECTCODE titlebar menu are:

|                           |                                                                                                                                                             |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GraphCalls</b>         | If the GRAPHCALLS package is loaded, then calls <b>GRAPHCALLS</b> on the current selection.                                                                 |
| <b>InspectCode</b>        | Opens a new <b>INSPECTCODE</b> window on the current selection.                                                                                             |
| <b>Inspect</b>            | Does an <b>INSPECT</b> on the value of the current selection. This item has SUBITEMS (see below).                                                           |
| <b>Pretty Print Value</b> | Prompts for region to open a window, and prettyprints the value of the current selection in it. This item has SUBITEMS (see below).                         |
| <b>Quit</b>               | Closes this window and kills the associated TEDIT process. (Closing the window with the WindowMenu, or by calling <b>CLOSEW</b> on it does the same thing.) |

The **Inspect** and **Pretty Print Value** menu options have the following SUBITEMS which affect how the value of the current selection is determined:

|               |                                                                                                                                              |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Freely</b> | The value of the current selection is determined by any binding that a free-reference from the INSPECTCODE window menu handling code (i.e by |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------------|

(**EVALV** selection)). This is the default behavior when a menu selection is made directly from the titlebar menu without using the SUBITEMS menu.

**Globally**

The value of the current selection is determined by its top level (Global) binding.

**In Process Context**

The value of the current selection is determined by its binding in the context of a specified process. A menu of all current processes will be brought up to allow you to specify a process.

**INSPECTCODE-TEDIT** also defines the LISPXMATCHRO **IC** which INSPECTCODE's its argument.

---

---

**KEYOBJ**

---

---

By: Greg Nuyens

Supported by Jan Pedersen (Pedersen.pa@Xerox.com)

KEYOBJ provides a LISP imageobject which mimics a key. The default image looks like this:



These keys are pressed by clicking the mouse inside the key's image. The result of pressing a key is determined (just like the physical key) by the Interlisp-D system function KEYACTION. To enter a KEYOBJ into TEdit type ^o. Inside the window that pops up, call the following function:

(KEYOBJ.CREATE KeyName KeyLabel Abortable) [Function]

KeyName is the key that you want the object to behave like. (CENTER in the example above). KeyLabel is an optional label other than the key whose action it mimics. If KeyLabel is a list of two elements, the first is displayed above the second. Abortable is a flag which indicates that no transitions should be generated if the mouse button is released outside the key image.

KEYOBJ.FONT [Variable]

Determines the font in which the label is created inside the keyobj. Default is Helvetica 10.

---

---

## KINETIC

---

---

By: Anon.

Recompiled for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

### INTRODUCTION

An ancient graphics hack, converted to work with idle.

### OPERATION

(KINETIC *WINDOW*)

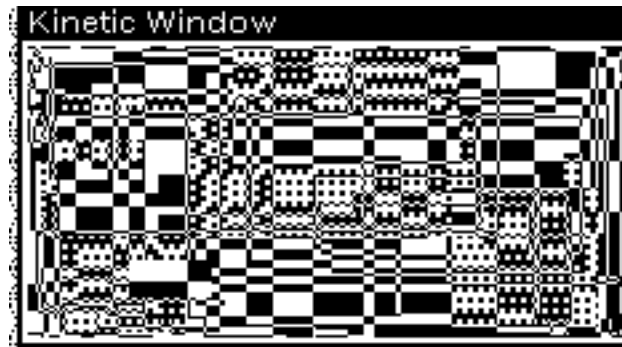
[Function]

to randomly invert rectangles on *WINDOW*, or on KINETICWINDOW (set up first time). Choosing the Kinetic on the Idle Choose Display menu will select the KINETIC function as the Idle display.

CHECKSHADE

[Variable]

If non-nil, CHECKSHADE is a texture which is used for some of the rectangles sometimes. Defaults to 63903.



---

---

**LAMB DATRAN**

---

---

By: >>Your Name<< (>>Your net address<<)

>>Other packages necessary to run this one<<

The purpose of this package is to facilitate defining new LAMBDA words in such a way that a variety of other system packages will respond to them appropriately. A LAMBDA word is a word that can appear as the CAR of a function definition, like LAMBDA and NLAMBDA. New LAMBDA words are useful because they enable the user to define his or her own conventions about such things as the interpretation of arguments, and to build in certain defaults about how values are returned. For example, the DECL package defines DLAMBDA as a new LAMBDA word with unconventional arguments such as the following:

```
(DLAMBDA ((A FLOATP) (B FIXP) (RETURNS SMALLP))
(FOO A B))
```

In order for such an expression to be executable and compilable, a mechanism must be provided for translating this expression to an ordinary LAMBDA or NLAMBDA, with the special behavior associated with the arguments built into the function body. The LambdaTran package accomplishes this via an appropriate entry on DWIMUSERFORMS that computes the translation.

Besides executing and compiling, Interlisp applies a number of other operations to function definitions (e.g., breaking, advising), many of which depend on the system being able to determine certain properties of the function, such as the names of its arguments, their number, and the type of the function (EXPR, FEXPR, etc.). The LambdaTran package also provides new definitions for the functions FNTYP, ARGLST, NARGS, and ARGTYPE which can be told how to compute properties for the user's LAMBDA-words.

A new LAMBDA-word is defined in the following way:

1. Add the LAMBDA-word itself (e.g., the atom DLAMBDA) to the list LAMB DASPLST. This suppresses attempts to correct the spelling of the LAMBDA-word.
2. Add an entry for the LAMBDA-word to the association list LAMB DATRANFNS, which is a list of elements of the form: (LAMBDA-WORD TRANFN FNTYP ARGLIST), where (LAMBDA-WORD is the name of the LAMBDA-word (e.g., DLAMBDA).

TRANFN is a function of one argument that will be called whenever a real definition is needed for the LAMBDA-word definition. Its argument is the LAMBDA-word definition, and its value should be a

conventional LAMBDA or NLAMBDA expression which will become the translation of the Lisp LAMBDA-word form. The free variable FAULTFN is bound to the name of the function in which the LAMBDA-word form appeared (or TYPE-IN if the form was typed in).

*FNTYP* determines the function type of a definition beginning with LAMBDA-WORD. It is consulted if the definition does not already have a translation from which the function type may be deduced. If *FNTYP* is one of the atoms *EXPR*, *FEXPR*, *EXPR\**, or *FEXPR\**, then all definitions beginning with LAMBDA-word are assumed to have that type. Otherwise, *FNTYP* is a function of one argument that will be applied to the LAMBDA-word definition. Its value should be one of the above four function types.

*ARGLIST* determines the argument list of the definition if it has not already been translated (if it has, the *ARGLIST* is simply the *ARGLIST* of the translation). It is also a function of one argument, the LAMBDA-word definition, and its value should be the list of arguments for the function (e.g., (A B) in the DLAMBDA example above). If the LAMBDA-word definition is ill formed and the argument list cannot be computed, the function should return T. If an *ARGLIST* entry is not provided in the LAMBDATRANFNS element, then the argument list defaults to the second element of the definition.

As an example, the LAMBDATRANFNS entry for DLAMBDA is (DLAMBDA DECL EXPR DLAMARGLIST), where DECL and DLAMARGLIST are functions of one argument.

Note: if the LAMBDA-word definition has an argument list with argument names appearing either as literal atoms or as the first element of a list, the user should also put the property INFO with value BINDS on the property list of the LAMBDA-word in order to inform DWIMIFY to take notice of the names of the arguments when DWIMIFYing.

---

---

**LIFE**

---

---

By various folks, including help from Mike Dixon (MikeDixon.PA@Xerox.COM) and Larry Masinter (Masinter.pa@Xerox.com)

This Life program is a translation of the Smalltalk-80 version in the book **Smalltalk-80: The Language and its Implementation**, by Goldberg and Robson.

Input is a window where the "on" pixels are interpreted as living cells. The window is continually updated as life goes on.

Now an "idle" hack: LIFEDEMO as a display function plays life with the bits of the screen (in a copy of them in a window, e.g., it doesn't smash your screen.)

(Lifeldle W N) [Function]

Run Life in window W, using the bits behind W as a starting point. N is optional, and can either be 1, 2, 4 or 8. Its the magnification of the life window.

(Life W N) [Function]

Like Lifeldle but uses the current contents of the window.

---



---

## LOADMENUITEMS

---



---

By: sML (Lanning.pa@Xerox.com)

### INTRODUCTION

Some utility files are so useful that users will always want them in their system: these files are typically loaded from the users INIT file. A (rather large) number of other utilities are only sometimes useful. Users are faced with the choice of either loading these files from their INIT files (slowing down the initialization process and consuming space, whether the utility is needed or not) or having to remember how to load and initialize these files.

LOADMENUITEMS addresses this problem: it defines a new filepackage command that can be used to add entries onto the background menu for easy loading of utility files.

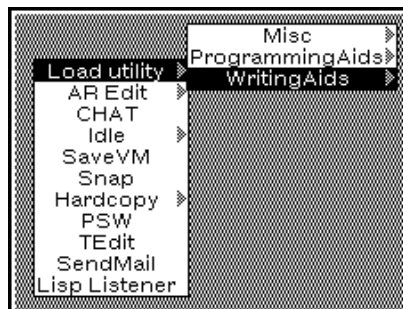
[NOTE: All (advertised) symbols in this utility are in the INTERLISP package.]

### EXAMPLE

The filepackage command

```
(COMS
 ;; Make it easy to load some oft-used utilities
 (FILES LoadMenuItems)
 (LOADMENUITEMS WritingAids Sketch VirtualKeyboards ProofReader)
 (LOADMENUITEMS ProgrammingAids (Spy (SPY.BUTTON)))
 (LOADMENUITEMS NIL VStats Calendar))
```

will add an entry "Load utility" to the background menu. "Load utility" will have three subitems: Misc, ProgrammingAids, and WritingAids:



WritingAids will in turn have three subitems: ProofReader, Sketch, and VirtualKeyboards; Misc will have the subitems Calendar and VStats; ProgrammingAids will have the single subitem Spy.





Selecting any of these final menu items will load the corresponding file. In addition, the `spy` menu item will evaluate the form `(SPY.BUTTON)` after loading the file `Spy`.

## INTERFACE

(LOADMENUITEMS *group utilDescr1 utilDescr2 ...*) [FilePackageCommand]

Dumps out to the file a form that will add items to the background menu for loading *utilDescr1*, *utilDescr2*, ... Each item will be added to the *group* subitem of the "Load utility" item on the background menu; if *group* is NIL it defaults to "Misc".

In the simplest case, *utilDescr* is a LITATOM. This is used when you want to load a file without any extra initialization, and the file is on one of the directories in `DIRECTORIES`. Selecting the resulting item will evaluate `(DOFILESLOAD 'utilDescr)` and print an informative message in the prompt window when the `DOFILESLOAD` is finished. The added item will have the label *utilDescr*.

In the general case, *utilDescr* is a list. This is used when you want to specify an initialization form to be evaluated when the utility is loaded, or when the file description is not a LITATOM. In this case, selecting the menu item will evaluate `(DOFILESLOAD (CAR 'utilDescr))`. If *utilDescr* is a list of two elements, the `CADR` of *utilDescr* will be evaluated after the utility is loaded; otherwise an informative message will be printed in the prompt window. The added item will have as a label the first LITATOM in the `CAR` of *utilDescr*; this is the first file that will be loaded when the item is selected.

In each of the above cases, the item is removed from the background menu after the utility is loaded and initialized.

When a utility is loaded from the "Load utility" menu, the event is added to the history list. This way you can UNDO loading a utility.

Some illustrative examples:

```
(LOADMENUITEMS NIL VStats) ;; This adds the item "VStats" to the "Misc" subitem

 ;; Selecting the "Spy" item will load SPY and call
 ;; SPY.BUTTON to bring up the spy button icon
(LOADMENUITEMS ProgrammingAids (Spy (SPY.BUTTON)))

 ;; This will add the item "GO" to the "Games" group
(LOADMENUITEMS Games (((SYSLOAD FROM {PHYLUM}<Foster>Lisp>) GO)))

 ;; These items are useful for Lafite users, but aren't always
 ;; needed
(LOADMENUITEMS MailTools LafiteFind Undigestify MailScavenge)
```

## FUNCTIONS

(AddLoadMenuItem *group fileDescr startUpForm*) [Function]

Add a menu item to the background menu that will load the files. The item will be added under the top level item "Load utility". *group* is the submenu name for this file; the default is `Misc`. *fileDescr* is a list that can be passed to `DOFILESLOAD` to load the files. *startUpForm* is an optional form that will be evaluated after the `DOFILESLOAD`; the default will print a nice message in the prompt window. The `LOADMENUITEMS` filepackage command described above expands to calls to `AddLoadMenuItem`.

`AddLoadMenuItem` is UNDOable.

(PickLoadUtilityItem *utility-name* &OPTIONAL *group-name no-errors-p*) [Function]

This is the programatic equivalent of selecting the item named *utility-name* from the "Load utility" item on the background menu. If *group-name* is given, only that group under the "Load utility" item is searched for the utility; otherwise the entire menu item is searched. If multiple matching items are found, a continuable error is signaled. Proceeding from this error will let you pick one of the items to execute. If no matching items are found, a continuable error is signaled. The *no-errors-p* flag controls whether or not these errors are actually signaled: if *no-errors-p* is true, `PickLoadUtilityItem` ignores the errors. `PickLoadUtilityItem` return T if the utility was loaded, NIL otherwise.

---

---

**LOGIC**

---

---

By: Roberto Ghislanzoni ("Roberto\_Ghislanzoni".MKTRXI@Xerox.com)

Uses: TABLEBROWSER

This document last edited on 20-Dec-1988 00:52:22.

**NOTES ABOUT LOGIC MEDLEY RELEASE**

This LOGIC release(1.3) is more robust than previous on top of Lyric; there are some bugs fixed. The major enhancement is the possibility of handle multiple SEDIT sessions: there is a global variable, \*LOGIC-CLOSE-ON-COMPLETION-FLG\* (defaulting to T), that controls the behaviour of the editing: NIL means not to close the editing window and not to perform a check on the correctness of the axiom just typed in, T means to check definitions and to close the editing window.

**INTRODUCTION**

This package is devoted to people who want to use a logic paradigm in their programming environment. LOGIC was initially developed in Franz Lisp at the Computer Science Department of the University of Milan: now a modified part of its kernel is running in Common Lisp, so it is possible to use it under every machine running CL: within the Xerox environment, some features are available in order to ease the construction of the programs.

All of the source codes are available: sorry if they are awful! But it's better to have efficiency than syntactic sugar ...

**LOGIC MANUAL**

LOGIC is essentially a theorem prover based on Horn clauses: the user is allowed to create many theories and within these theories to specify some predicates (clauses); as FOL does, it is also possible to specify some *semantic attachments* (SA), in order to use all the capabilities of the environment: in our implementation, these SAs are expressed in Lisp. A goal proof is performed within specified theory(es); the user is allowed to dinamically change the theories involved.

These are the elements of the language:

- a *variable* is an atom beginning with '?'
- an *atomic formula* is a list beginning with the name of the predicate and followed by the terms: (on table book)  
(mother ?x ?y)
- a *clause* is a list beginning with the consequent, followed by the special symbol ':-', and by the sequence of the antecedents:

```
((grandfather ?x ?y) :- (father ?x ?z) (father ?z ?y))
```

- a *set of clauses* is the definition of a predicate

```
((append () ?a ?a)) ((append (?a . ?b) ?c (?a . ?d)) :- (append ?b ?c ?d)))
```

- a *theory* is a set of the definitions of some predicates

## HOW TO LOAD AND INIT LOGIC

In order to use LOGIC, load the files LOGIC and LOGIC-UNIFIER. From within the Xerox Lisp Environment, you can also load the development environment LOGIC-DEVEL.DFASL. After loading the files, call the functions:

- (CREATE-BACKGROUND-THEORY): this function creates the main theory reading the data it needs from the file LOGIC.LGC.

- (CREATE-VARIABLES): this functions creates and initializes the variables used by the unifier; it takes a few time to perform its job. This call is due to a lack of the Xerox garbage collector: since the unifier uses techniques of redenomination, a great number of symbols is generated; the 1186 does not release the space used by these symbols, and so they fill up the GC table. The workaround is to re-use all the symbols generated.

If you want to port this code on another machine running CL, it is a matter of taste to eliminate this piece of code, with a little hacking on the source codes.

These are the functions available from the top-level executive of Lisp:

```
(ALL VARS CONJ THS)
```

[Function]

Returns the *vars* that satisfies the goal (*conj*) in the list of theories (*ths*): the background theory is always used. For example you can ask the system to prove:

```
(ALL '(?a ?b) '((append ?a ?b (1 2 3))) '(append-theory))
```

```
--> ((NIL (1 2 3)) ((1) (2 3)) ((1 2) (3)) ((1 2 3) NIL))
```

```
(ANY HOW-MANY VARS CONJ THS)
```

[Function]

Returns *how-many vars* that satisfies the goal:

```
(any 2 '?a '((append ?a ?b (1 2 3))) '(append-theory))
```

```
--> (NIL (1))
```

```
(ATTACH SA-NAME DEFINITION THEORY-NAME)
```

[Function]

Allows to create semantic attachments:

```
(ATTACH 'createw '(lambda (name) (IL:CREATEW () name)) 'my-theory)
```

and now:

```
(ANY 1 () '((createw "Kiss me on my lips")) 'my-theory)
```

--> ;;creates a window on the screen

(CREATE-THEORY *THEORY-NAME*) [Function]

Creates a brand-new theory with that name: return the name of the theory created, not the theory itself.

(LIST-ALL-THEORIES) [Function]

Return a list of the defined theories, currently available.

(LOAD-THEORY *THEORY-NAME*) [Function]

Loads from the current directory the specified *theory-name*; the name of the theory file has the extension .LGC , and it must be previously created by the corresponding function SAVE-THEORY

(LOGIC-ADDA *PRED CLAUSES THEORY-NAME*) [Function]

Adds to the definitions of the predicate *pred* the specified *clauses*, that holds in the theory *theory-name*: the clauses are put in front of the already existing clauses:

(LOGIC-ADDA 'C '(((C 1)) ((C ?x) :- (A ?x))) 'my-theory)

(LOGIC-ADDZ *PRED CLAUSES THEORY-NAME*) [Function]

Adds to the definitions of the predicate *pred* the specified *clauses*, that holds in the theory *theory-name*: the clauses are put at the end of the already existing clauses:

(LOGIC-ADDZ 'C '(((C 2)) ((C ?x) :- (A ?x) (B :y))) 'my-theory)

(LOGIC-ASSERT *PRED CLAUSES THEORY-NAME*) [Function]

Replaces all previous definitions of the predicate *pred* with *clauses*.

(LOGIC-DELETE *PRED-OR-SA THEORY-NAME*) [Function]

Erases from the theory *theory-name* the definition of *pred-or-sa*, that may be either a predicate or a semantic attachment

(LOGIC-DELETE-FACT *FACT-NAME FACT-CLAUSE THEORY-NAME*) [Function]

Erases from the definition of the clauses on the predicate *FACT-NAME* the specified clause *FACT-CLAUSE*, within the theory *THEORY-NAME*.

(MERGE-THEORIES *NEW-THEORY-NAME &REST LIST-OF-THEORIES*) [Function]

Creates the new theory *NEW-THEORY-NAME* made up by all the predicates and sas that hold in all the theories *LIST-OF-THEORIES*: now no control is performed on the consistency in the merging of the theories

(PROVE CONJ *THS*) [Function]

Calls the prover on the specified goals *conj*. *THS* is a list of the theory(es) used. PROVE returns only T or NIL

(SAVE-THEORY *THEORY-NAME*) [Function]

Writes on the local directory the contents of the theory *theory-name*. You will find later a file whose name is composed by the theory name and the extension LGC.

The format of the contents of the file is the following:

```
theory-name
number of semantic attachments
<sa name1> <sa definition>
..
<sa nameN> <sa definition>
number of predicates
<predicate name 1> <clauses for predicate 1>
..
<predicate name N> <clauses for predicate N>
```

A theory file (with .LGC extension) may be created by the user employing a text editor like Emacs or VI (on Symbolics, SUNs etc.), avoiding the saving of the theory every change he performs.

(SHOW-DEFINITION *ELEMENT THEORY-NAME*) [Function]

Shows the definition of *element*, either a predicate or a semantic attachment.

(SHOW-THEORY *THEORY-NAME &OPTIONAL VERBOSE*) [Function]

Shows the contents (name of predicates and sas) of the theory *theory-name*; if *verbose* is T, all the definitions are shown.

## THE BACKGROUND THEORY

In the background theory, many interesting primitive predicates are available:

! [Predicate]

The cut predicate, well-known to the PROLOG programmers: a typical example of its use can be the definition of the predicate NOT:

```
((not ?formula) :- (wff ?formula) ! (fail))
((not ?formula)))
```

|                                                                                                                                                                                                                                                  |                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------|
| <i>(TRUE)</i>                                                                                                                                                                                                                                    | [Predicate]<br>This predicate always succeeds     |
| <i>(FAIL)</i>                                                                                                                                                                                                                                    | [Predicate]<br>The predicate that never succeeds  |
| <i>(PRINT ?arg)</i>                                                                                                                                                                                                                              | [Predicate]<br>Prints the argument ?arg passed by |
| <i>(EVAL&amp;PRINT ?arg)</i><br>This predicate evaluates and print the result of evaluation of the form ?arg:<br>(prove '((eval&print (+ 3 4))) '(my-theory))<br>7<br>T                                                                          | [Predicate]                                       |
| <i>(LOGIC-ADDA ?PREDICATE-NAME ?CLAUSES ?THEORY-NAME)</i><br>Adds in front of the clauses that define the predicate ?PREDICATE-NAME in the theory ?theory-name the other clauses ?CLAUSES                                                        | [Predicate]                                       |
| <i>(LOGIC-ADDZ ?PREDICATE-NAME ?CLAUSES ?THEORY-NAME)</i><br>Adds to the end of the clauses that define the predicate ?PREDICATE-NAME in the theory ?theory-name the other clauses ?CLAUSES                                                      | [Predicate]                                       |
| <i>(LOGIC-ASSERT ?PREDICATE-NAME ?CLAUSES ?THEORY-NAME)</i><br>Replaces all definition for the predicate ?PREDICATE-NAME in the theory ?THEORY-NAME with the new clauses ?CLAUSES                                                                | [Predicate]                                       |
| <i>(LOGIC-DELETE ?PREDICATE-OR-SA--NAME ?THEORY-NAME)</i><br>Deletes all definition for predicate (or sa) ?PREDICATE-OR-SA-NAME in the theory ?THEORY-NAME                                                                                       | [Predicate]                                       |
| <i>(LOGIC-DELETE-FACT ?FACT-NAME ?FACT-CLAUSE ?THEORY-NAME)</i><br>Erases from the definition of the clauses on the predicate <i>FACT-NAME</i> the specified clause <i>FACT-CLAUSE</i> , within the theory <i>THEORY-NAME</i> .                  | [Function]                                        |
| <i>(SET ?var value)</i><br><br>With this predicate it is possible to set a variable within the demonstration (remind that a variable always starts with a '?'):<br>(prove '((set ?x (list 'a 'b 3))(print ?x)) '(my-theory))<br>--> (a b 3)<br>T | [Predicate]                                       |

(RETRACT ?theory-name) [Predicate]

Tells the interpreter that it must use no more the theory ?theory-name during the ongoing demonstration; this elision is made only on the current active node of the demonstration tree

(USE-THEORY ?theory-name) [Predicate]

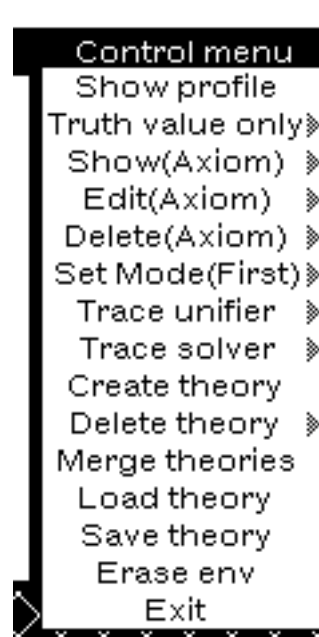
Tells to the interpreter that it must use the theory ?theory-name for the ongoing demonstration.

(WFF ?formula) [Predicate]

This is a second order predicate that allows you to prove the truth value of the well formed formula ?formula

If you load only the LOGIC files, this is the environment you have. On Xerox machines, you can also load the file LOGIC-DEVEL, that allows you to have the development environment: a new entry in your background menu is created, and so you are able to open a logic demonstration window.

This is the control menu:



SHOW-PROFILE: shows the current profile: the MODE of demonstration (FIRST, ALL, INTERACTIVE), and the tracing flags on unifier and solver

TRUTH VALUE ONLY: this flag controls if the prover returns all the goals with the variables instantiated or only the values T or NIL

SHOW AXIOM: shows the definition of an axiom or of a semantic attachment

EDIT AXIOM: edits the definition of an axiom or of a semantic attachment



DELETE AXIOM: deletes the definition of an axiom or of a semantic attachment

SET MODE: sets the mode of the demonstration: this may be ALL, FIRST or INTERACTIVE

TRACE SOLVER: the solver is the procedure of the interpreter that takes as arguments a tree, a formula and the clauses for that formula, and gives back the new tree obtained by the resolution operation; its behaviour is traced on a debugging window which has the middle menu capability of dribbling; the output file has the extension TRC.

TRACE UNIFIER: traces the going on of the unifier on a debugger window; this window too has the middle menu capability of dribbling its output. The pattern, the datum and the unification environment will be shown to the user.

CREATE THEORY: creates a new theory

DELETE THEORY: all the theories loaded are showed in a tablebrowser at the left of the main window; when you select one or more theories, this means that you want to use them for your demonstration; this command deletes the selected theories; you can however undelete or expunge them with the subitems of this entry. Remember that, for undeleting the selected theories with the tablebrowser mark(), you must click middle button on it and press CTRL (PROP) key

MERGE THEORIES: merges the selected theories in a new theory; the user is prompted for the name of the new theory

LOAD THEORY: loads a theory from a file in the current directory

SAVE THEORY: save the selected theory(ies) on the corresponding files

ERASE ENV: erases all the environment of the window

EXIT: exits from the environment

Remember that, for every demonstration requested, there must be at least one theory selected in the tablebrowser at the left of the main window

I hope these notes help you to use LOGIC.

You can find some examples in the theory file LOGIC- EXAMPLES.LGC.

Any suggestion is welcome: since it is not fully tested, please notify every kind of error or bug you will find.

## EXAMPLES

Choose LOGIC from the background menu: a new window will appear: choose LOAD THEORY from the control menu and type in LOGIC-EXAMPLES at the request in the prompt window: mark the theory loaded in the theories window and try:

```
((APPEND ?A ?B (1 2 3)))
```

the system will respond you

```
((APPEND NIL (1 2 3) (1 2 3)))
```

Click now on SHOW PROFILE: you will see

```
MODE: FIRST /Unifier: NOTRACE /Solver: NOTRACE /Values: NIL
```

Choose SET MODE ALL (submenu) and retry the same goal as before: you get the answer:

```
((APPEND NIL (1 2 3) (1 2 3)))
```

```
((APPEND (1) (2 3) (1 2 3)))
```

```
((APPEND (1 2) (3) (1 2 3)))
```

```
((APPEND (1 2 3) NIL (1 2 3)))
```

NIL

In the theory LOGIC-EXAMPLES a simple little maze is described: type in the goal:

```
((search a g))
```

that will find a path from the room 'a' to the room 'g'.

|   |   |   |   |
|---|---|---|---|
| M | I | E | F |
| B | C | D | H |
| A | N | G | L |

Here are other examples of goals you can try:

```
((sa-member 3 (1 2 3 4 5)))
```

```
((logic-member 3 (1 2 3 4 5)))
```

The first one is a SA, the latter is a predicate.

```
((NOT (A 1))) --> T
```

```
((NOTMEMBER 2 (1 3 4))) --> T
```

and so on.

Try now all the other features of the language.

You can ask the system for the same goals from the lisp listener:

```
(load-theory 'logic-examples)
```

```
(prove '((APPEND ?X ?Y (1 2 3 4)) '(logic-examples)) --> T
```

```
(all '(?X ?Y) '((APPEND ?X ?Y (1 2 3 4)) '(logic-examples))
```

```
--> ((NIL (1 2 3 4)) ((1) (2 3 4)) ((1 2) (3 4)) ((1 2 3) (4)) ((1 2 3 4) NIL))
```

Have fun!

---

---

**LOGTIME**

---

---

Johannes A. G. M. Koomen  
Koomen.wbst@Xerox  
Koomen@CS.Rochester.edu

October 12, 1989

**SUMMARY**

LOGTIME is a facility for keeping track of how you spend your time. Clicking on the background menu entry "Log Time" loads a data file and starts up a process. The process wakes at regular intervals and prompts the user for the activity engaged in during the latest interval. The default response is the previous activity. Clicking the right mouse button on the prompt window brings up a menu of known activities. Control-E in the prompt window causes the latest interval to be ignored. Clicking on the background menu entry "Log Time" when a LOGTIME process is already active causes the process to be awoken immediately. Rollout menu entries provide a way to edit the current data, to report on the data accumulated for the current day, and to stop keeping track of time (either "Quit" which updates the datafile, or "Abort" which throws away current data).

**DESCRIPTION**

(LOGTIME.START *datafile*) [Function]

Starts and returns a new LOGTIME process to keep track of time, using *datafile* as the file to load and store data, unless a LogTime process is already running, in which case NIL is returned. If *datafile* is NIL, it defaults to LOGTIME.DATFILE (see below).

(LOGTIME.UPDATE) [Function]

Wakes up the LOGTIME process, if any, causing an immediate prompt for an activity.

(LOGTIME.STOP *abortflg*) [Function]

Stops and returns the LOGTIME process currently running, if any. If *abortflg* is NIL, the datafile indicated in the preceeding call to LOGTIME.START is updated.

The "Quit" roll-out entry on the background menu invokes (LOGTIME.STOP).

The "Abort" roll-out entry on the background menu invokes (LOGTIME.STOP T).

(LOGTIME.EDIT *olddatafile newdatafile*)

[Function]

Loads *olddatafile* if necessary. Invokes the Lisp editor on a list of entries, where each entry is a list of starting date, ending date and activity, e.g. ("10-Oct-89 13:07" "10-Oct-89 13:12" "Mail").

If all entries can be reparsed upon return from the editor the dataset is modified accordingly. If the data did not come from the LOGTIME process currently running (if any) the data are written back to *newdatafile* if given, otherwise to *olddatafile*.

The "Edit" entry on the background menu invokes (LOGTIME.EDIT).

(LOGTIME.REPORT *bydateflg verboseflg fromdate todate datafile reportfile*)

[Function]

Generates a report of the data in *datafile* (defaults to LOGTIME.DATAFILE). If *bydateflg* is non-NIL the report is sorted by date, otherwise by activity. If *verboseflg* is non-NIL, the report lists each single entry in the dataset, otherwise only cumulative time (per date or per activity) is given. If *fromdate* is given (a string acceptable to IDATE) all entries prior to this date are skipped. If the time is omitted from *fromdate* it is assumed to be 0:00. If *todate* is given (a string acceptable to IDATE) all entries past this date are skipped. If the time is omitted from *todate* it is assumed to be 23:59. The report will be generated on *reportfile* (which defaults to LOGTIME.REPORTFILE).

The "Report" entry on the background menu invokes (LOGTIME.REPORT T T <today>).

LOGTIME.INTERVAL

[Variable]

The number of minutes the LOGTIME process will sleep in between prompts. Initial value is 15.

LOGTIME.DATAFILE

[Variable]

The name of the default datafile. Initial value is (CONCAT (DIRECTORYNAME) "LogTime.Data").

LOGTIME.REPORTFILE

[Variable]

The name of the default reportfile. Initial value is T.

LOGTIME.PROMPT.URGENCY

[Variable]

The value used for the PROMPTFORWORD argument *urgency.option*. Initial value is 'TTY'.

---

**LOOKUPINFILES**

---

By: dgb (Bobrow.pa@Xerox.com)



## INTRODUCTION

The LOOKUPINFILES package is a facility for building quick and easy access to on-line files. It allows search for a target string through all files in a specified list. It finds the target, and brings up the file in a window, with the target selected in inverse video. The file can then be used as the source for text for other documents. It is the basis for the user facilities of ADDRESSBOOK and FIND-CITATION. Its interface is defined by the function:

(MakeLookupWindow fileList processName mainWindowRegion iconBM iconMask iconPosition iconTitle)

These arguments are used as follows:

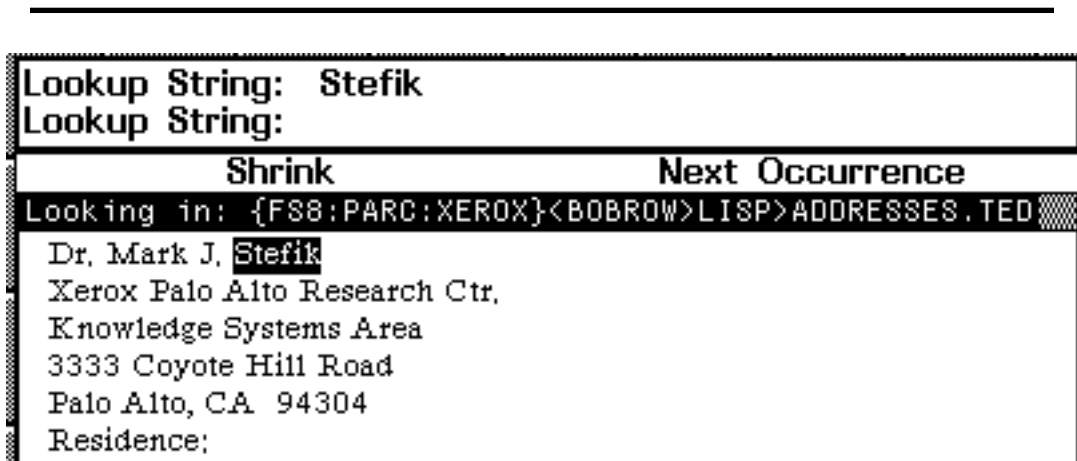
|                  |                                                              |
|------------------|--------------------------------------------------------------|
| fileList         | List of file names. Search goes through these files in order |
| processName      | Name appearing in PSW for this lookup process                |
| mainWindowRegion | Region for window showing text found                         |
| iconBM           | Bit map for icon when mainWindow is shrunk                   |
| iconMask         | Mask for icon                                                |
| iconPosition     | Position for icon                                            |
| iconTitle        | Title put under icon to distinguish lookup operation         |

Arguments other than fileList are optional. Calling MakeLookupWindow will construct a Lookup window, and shrinks it to the icon provided. Opening this icon shows the

window interface to the search process. To find any string in one of the files, type the string followed by a return. The program will quickly search through the files and show you an occurrence of the string typed. The located string is shown in inverse video. The title of the window will contain the name of the file in which the entry was found. The search ignores case; e.g. "bobrow" matches "Bobrow". The text of the document is scrollable, and any portion can be shift selected into another document.

Type carriage return, ^X, or click on **Next Occurrence** to search further in the files for the same string. If no (further) occurrences are found, the text window will display a message indicating the failure. Searching again after failure will start the search from the beginning of all the files, using the same lookup string. Typing a new string can be repeated as many times as you like. When you are done, just SHRINK the window back to its icon by using the Shrink selection in the title bar .

The window below is taken from the use of this package as an online address book.



Example LOOKUPINFILES window

---

## Notes

### Caching Files

When you first create the window, the program will copy the files to {CORE}, significantly speeding up queries. Bugging in the title of the main window with the left or middle mouse button will produce a menu with an option to recache all these files.

### **Editing Your Files**

To edit the file in which a string is found, click middle button in the title of the main window, and select the option "Edit file named in window title". A TEDIT process editing the file will be set up. This process is independent of the lookup process. To select the file to be edited, rolloff the above item, and select "Select file to edit". A menu of files used by the Lookup process will be presented to you. Selecting one will cause that file to be edited.

To make editing changes visible to the lookup process, PUT the file in TEDIT; when it is done, recache the the file in core. To recache just the file edited, (the one specified in the title bar of the window), select the option "Recache file named in window title" in the middle button title bar menu. You can recache all files by selecting the option "Recache all files" in the title menu (a subselection of the item "Recache file named in window title").

### **Adding to the List of Files**

To add to the list of files being used for lookup, select the option "Add new file" in the title bar menu. This file will be added to the beginning of the list of files to be searched, and cached in core.

### **Deleting a file from the List of Files**

To delete from the list of files being used for lookup, select the option "Delete file from list" in the title bar menu. This file will be deleted from the list of files to be searched.



---

---

**MAGNIFIER WINDOW**

---

---

By: Richard R. Burton (Burton.PA @ Xerox.Com)

This document last edited on March 17, 1986.

**INTRODUCTION**

File: MAGNIFIER.LCOM

Tired of giving demos in which only the two people sitting next to you can see the screen? This small package implements magnifying windows, windows that show an enlarged copy of that portion of the screen that is around the cursor. A magnifying window can be created either by calling the function `MAGNIFYW` or by selecting the item "Magnifier" from the background menu. A magnifying window can be made to any size and is distinguished by its large border. Once a magnifying window has been created, it can be activated by clicking the left button in it. While activated, the cursor will be replaced by a black rectangle and the contents of the rectangle will be displayed in the magnifying window enlarged by a factor of 4. The contents will continue to track the location of the cursor until the left button is clicked a second time. The magnifier can be reshaped.

Suggested use: When six people drop into your office unannounced for a demo, create a magnifying window across the top or bottom of your screen (so the people in the back can see it easily). When it is important for people to read what you are talking about, move the cursor into the magnifier, click the left button, move the cursor over the area of interest and, when the image in the magnifier has what you want, click the left button again. This will leave an enlarged part of the screen in the magnifier and free the mouse of other things. You can leave magnifier active but it will not block (so no other processes get to run) and if you move the cursor, the image in the magnifier will move too.

---



---

## MakeGraph

---



---

By: D. Austin Henderson, Jr.

Supported by: Nick Briggs (Briggs.pa@Xerox.com)

### INTRODUCTION

MakeGraph is a module which sits on top of Grapher and helps one create graphs depicting a data structure by walking through it. The central idea is that each point in the walk (and node in the graph) is characterized by a datum/state pair and motion is defined by a graph specification in the form of state transition function. This function is specified by a collection of state specifications, each of which indicates how to display (label and font) the datum when one is in that state and how to find the datum/state pairs which are the sons of that node. Also the state specification may specify additional roots for the walk. The generation of a branch of the graph ceases when either there are no sons of a node, or an already encountered node is revisited (identical datum and identical state). The module contains a function for creating such graphs and an example of its use: a function which graphs the graph specifications themselves. Comments are welcomed

### FUNCTIONS

The main functions are:

(MAKE.GRAPH *WINDOW TITLE GRAPH.SPECIFICATION ROOTS CONTEXT*  
*LEFTBUTTONFN MIDDLEBUTTONFN TOPJUSTIFYFLG DEPTH*) [Function]

Creates a MAKEGRAPH window. If *WINDOW* is NIL, then a new one will be created and the user will be prompted to position it. Otherwise, the graph will be shown in *WINDOW*. The window will be titled with *TITLE*, will call *LEFTBUTTONFN* and *MIDDLEBUTTONFN* on nodes selected (or NIL if selection is made where no node is positioned), and will be justified as indicated by *TOPJUSTIFYFLG* (a la Grapher). The button functions are defaulted to MAKE.GRAPH.LEFTBUTTONFN (which scrolls the window so that the selected node is in the middle of the window, or if the left shift key is depressed, prints out information about it) and MAKE.GRAPH.MIDDLEBUTTONFN (which provides a menu of two choices: INSPECT - inspects the datum of the node selected, or if the left shift key is depressed, inspects the node itself; and SUB.GRAPH - which opens another MAKEGRAPH window with the same parameters as this one, but with graph starting at the selected node). The arguments to MAKE.GRAPH are added as properties to the window under their argument names. Selecting in the title invokes the functions which are the values of the window properties TITLE.LEFTBUTTONFN and TITLE.MIDDLEBUTTONFN (not in the calling sequence; set by the user if desired; called with a single argument - *WINDOW*; defaulted to a function which provides a menu of UPDATE and SHOW.GRAPH.SPEC (see functionality below)). The graph is created according to the *GRAPH.SPECIFICATION* (see below) to depth *DEPTH*, starting from *ROOTS* which are (DATUM . STATE) pairs. *CONTEXT* is an extra argument which is passed along to all accessing expressions.

GRAPH.SPECIFICATION [Parameter]

A GRAPH.SPECIFICATION is a property list of STATE.SPECIFICATIONs where the properties are the state names.

## STATE.SPECIFICATION

[Property list]

A STATE.SPECIFICATION is a property list whose properties and values are as follows (in this, EXPR means a LISP form which will be evaluated in an environment in which DATUM is bound to the node's datum, STATE to the node's state, and CONTEXT to context):

## LABEL

[Property]

an expression returning something which will be printed as the label of the node; if no LABEL property is provided, the string "???" will be used.

## FONT

[Property]

an expression returning the font to be used for this node; if no FONT property is provided, the default font for the grapher will be used.

## SONS

[Property]

a form indicating a list of (DATUM . STATE) pairs to be used in generating the sons of this node; the acceptable forms are any of the following:

(data-expression state-expression)

[Property value]

where data-expression returns a list of datum's for the son nodes, and state-expression is evaluated in the context of each of these in turn to produce the corresponding state of each.

(LIST (data-expression state-expression) ...)

[Property value]

a template of expressions which are evaluated individually to produce a list of sons of the same form, viz. (DATUM . STATE) pairs.

(EVAL expression)

[Property value]

the expression returns a list of (DATUM . STATE) pairs of the sons.

(UNION sons-spec ...)

[Property value]

where each sons-spec is any of these forms (recursively).

(TRACE sons-spec)

[Property value]

a device for helping debug graph specifications; the value is the value of sons-specs; the user is given the chance to INSPECT them after they have been generated.

## ROOTS

[Property]

like SONS, except the resulting (DATUM . STATE) pairs are used as possibly additional roots of the graph.

(MAKE.GRAPH.CONSTRUCT *GRAPH.SPECIFICATION INITIAL.ROOTS CONTEXT DEPTH*)[Function]

This is the functional heart of MAKE.GRAPH broken out for those who wish to handle their own interactions with grapher and the window package. It produces a list of graphnodes with labels and sons as specified by *GRAPH.SPECIFICATION* (see MAKEGRAPH), starting from *INITIAL.ROOTS* which are (DATUM . STATE) pairs. *CONTEXT* is an extra argument which is passed along to all accessing expressions. Returns the list of graphnodes.

(MAKE.GRAPH.FIND.ROOTS *GRAPH.SPECIFICATION INITIAL.ROOTS CONTEXT DEPTH*)[Function]

Finds the real roots from a set of initial roots, using the same processing as MAKEGRAPH uses. This is helpful when you want to hand a "correct" set of roots of a structure to MAKEGRAPH without having to explore the dependencies within that structure. As with MAKEGRAPH, the data structure is processed according to the *GRAPH.SPECIFICATION* (see MAKEGRAPH), starting from *INITIAL.ROOTS* which are (DATUM . STATE) pairs. *CONTEXT* is an extra argument which is a passed along to all accessing expressions. Returns the real roots as a list of (DATUM . STATE) pairs.

### Supporting Functions

(MAKE.GRAPH.UPDATE.WINDOW *WINDOW*) [Function]

Uses the window properties (which may have been changed) to reinvoke MAKE.GRAPH on the window.

(MAKE.GRAPH.SHOW.SPEC *GRAPH.SPECIFICATION*) [Function]

Uses MAKE.GRAPH to produced a graph of a *GRAPH.SPECIFICATION*. It uses as the graph.specification for this layout the value of the variable MAKE.GRAPH.SPEC.SPEC which presents *GRAPH.SPECIFICATION* (reflectively) as a graph.specification. (MAKE.GRAPH.SPEC.SPEC can serve as a template for other graph.specifications. It is a fairly complex 9-state specification. For a simpler example see below under MAKE.GRAPH.SHOW.LIST.)

(MAKE.GRAPH.EXAMPLE.1) [Function]

Calls MAKE.GRAPH.SHOW.SPEC on MAKE.GRAPH.SPEC.SPEC.

(MAKE.GRAPH.SHOW.LIST *OBJECT*) [Function]

Uses MAKE.GRAPH to produced a graph of an arbitrary Lisp object. It uses as the graph.specification for this layout the value of the variable MAKE.GRAPH.LIST.SPEC which presents *OBJECT* as a tree whose nodes are LISTPs and whose leaves are non-LISTPs.

MAKE.GRAPH.LIST.SPEC [Variable]

MAKE.GRAPH.LIST.SPEC is the simple 1-state specification below, included here as an example of a graph.specification

```
(OBJECT (DOC (ANY LISP OBJECT) - some documentation
 LABEL (COND ((LISTP DATUM) " () ")
 (T DATUM))
 SONS ((COND ((LISTP DATUM) DATUM)
 (T NIL))
 (QUOTE OBJECT))
)
)
```

For a more complex example see above under MAKE.GRAPH.SHOW.SPEC.

(MAKE.GRAPH.EXAMPLE.2) [Function]

Calls MAKE.GRAPH.SHOW.LIST on MAKE.GRAPH.LIST.SPEC; that is, produces a graph of this simple graph.specification as a list. Notice that selecting the title command UPDATE in this window will yield a different graph of the same structure, viz. as a GRAPH.SPECIFICATION.

Other useful functions

(MAKE.GRAPH.DATUM *NODE*) [Function]

Returns the DATUM associated with the graph node *NODE*.

(MAKE.GRAPH.STATE *NODE*) [Function]

Returns the STATE associated with the graph node *NODE*.

(MAKE.GRAPH.FATHER *NODE*) [Function]

Returns the graph node which is the father of the graph node *NODE*.

---

---

## MANAGER

---

---

By: Jay Ferguson, Larry Masinter and Andrew Cameron III

Maintained by Ron Fischer (Fischer.pa@Xerox.com)

Revised by Matt Heffron (heffron@alumni.caltech.edu)

Uses: MASTERSCOPE, FILEBROWSER, DATABASEFNS, and COMMON-MAKE

### INTRODUCTION

In its latest incarnation Manager supports MasterScope and improves its performance. It can use DATABASEFNS for managing MasterScope databases, per file. It will use COMMON-MAKE (from Lispusers) to write files in CommonLisp format; and it *borrow*s some lower-level functionality from FILEBROWSER.

### USING MANAGER:

Manager provides a way to perform most common File Manager operations onscreen using menus, both pop-up and permanent. Activity centers around the filelst, or main, menu, and menus of items of a type in the file (like all FNS, or all VARS).

Printing and interaction is done through the Manager Command Activity Window. The first time it is needed you'll be prompted to size it onto the display. Thereafter, it will be used as needed. If shrunk before use it will wait 10 seconds after an operation and then shrink down again.

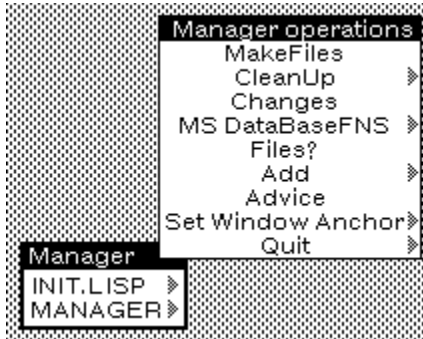
### The FILELST menu

The manager provides a menu of the FILELST:



The names in the FILELST menu can be copy selected.

Middle buttoning on the title bar of the FILELST menu pops up a menu of operations which are applied to all loaded files:



These operations are the same as the similarly named functions in the File Manager interface, except for the following slide off options:

CleanUp:

Set default: TCOMPL, the default compiler will be TCOMPL.

Set default: CL:COMPILE-FILE, the default compiler will be CL:COMPILE-FILE.

MS DataBase FNS:

*various MasterScope database flags can be set*

Add, notice a file via:

LOADFNS

LOADFROM

LOAD

ADDFILE\*

Edit FILELST, edit the FILELST directly in a lisp editor window.

Quit:

Quit\*, shut down the Manager, all menu caches cleared, windows closed.

Reset, shut down and turn on the Manager again.

Left buttoning on a file in the FILELST menu (without sliding off) pops up a menu of operations on that file:



See:

fast\*, prints the source of the file.

scrollable, displayed in a scrollable TEdit window.

(Re)Load:

Load\*, use current DFNFLG settings.

Sysload, load with File Manager turned off.

MakeFile, dump the file

MakeFile\*, dump the source of the file by remaking it.

New, dump the source without copying unchanged defs from existing file.

Fast, dump source without prettyprinting (fast).

CommonLisp, dump source in commonlisp format.

List, list the source file on the default printer.

CleanUp:

CleanUp\*, dump the file according to CLEANUPOPTIONS.

Set default compiler: TCOMPL.

Set default compiler: CL:COMPILE-FILE.

MasterScope:

Analyze\*, analyze the fns on the file.

Check, check the file for problems.

Show Paths, show paths of function calls on this file.

DatabaseFNS, display the database property for this file:

Set to ASK, ask about saving MS DB information.

Set to ON, automatically maintain MS DB information.

Set to OFF, do not save MS DB information.

Load DB, load an existing MS DB for this file.

Dump DB, dump the current MS DB for this file.

Compile:

Compile\*, compile the file based on the current settings.

CL:COMPILE-FILE, compile the file with CL:COMPILE-FILE.

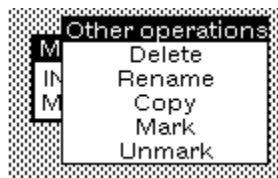
Changes:

Brief\*, prints the changes that have been made to this file.

Everything, prints the complete list of files changes.

Edit PL, brings up a lisp editor on the file's property list.

Middle buttoning on a file in the filelst menu (without sliding off) pops up a menu of generic operations on that file:



Delete, removes the file object from the system.

Rename, prompts for a new name and renames the file.

Copy, prompts for a new name and copies the file under that name.

Mark, mark the contents of the file as changed.

Unmark, unmark the contents of the file as changed.

Left buttoning on a file and sliding off to the right pops up a menu of types in the file:





Releasing on one of these places a menu of items of that type on the file:



This menu is not pop-up and remains on the display.

### The items of a type menu:

These menus contain the names of all instances of a particular type on a file. Names of items in these menus can be copy selected.

Left buttoning on an item name pops up a menu of operations on that type:



Edit, brings up the source text of the item in a lisp editor.

PrettyPrint:

Show\*, prints the source text of the item quickly.

Value, prints the global value of the item's name (assumed a symbol).

Function Def, prints the global function definition of the item's name (assumed a symbol).

Property List, prints the global property list of the item's name (assumed a symbol).

Documentation:

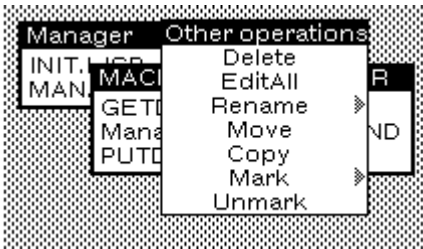
Documentation\*, prints the item's documentation string.

Describe, calls describe on the item's name (assumed a symbol).

The menu of item operations shown above is the general one. There are special menus for the following types:

FNS, FUNCTIONS, RECORDS, VARS

Middle buttoning on an item name pops up a menu of generic operations on that type:



Delete, removes this item from its file.

EditAll, edits all occurrences of this item's name in the latest source file (uses EDITCALLERS).

Rename:

Rename\*, rename this item in its file and update all uses of the name.

CopyDef, copy this item under a new name.  
 Rename All, rename this item in \*ALL\* loaded files.  
 Move, move this item into another file.  
 Copy, copy this item into another file.  
 Mark:  
     Changed\*, mark this item as changed by being edited.  
     Defined, mark this item as changed by being defined.  
     Deleted, mark this item as changed by being deleted.  
 Unmark, unmarks the source of this item as being changed (marks it "unchanged").

The file's makefile-environment has its readtable argument used to bring up thelisp structure editor properly on objects in the file. When SEdit is the lisp editor, the package used depends on SEdit's "correct package" heuristic (usually that of the symbol naming what is being edited).

### Loading and controlling Manager:

Just load the file. Manager can be started either from the background menu or by calling the FNS MANAGER (see below).

### Programmer's interface to Manager:

(MANAGER *POSITION*) [FNS]

Starts up the manager. If *POSITION* is given, the filelst menu will be appear there.

(MANAGER.RESET *RESTARTFLG*) [FNS]

Shuts down the manager. If *RESTARTFLG* is true, manager will be immediately restarted after the shutdown.

Manager.SORTFILELSTFLG [INITVAR]

If true, the FILELST will be sorted, without side effecting the actual FILELST variable. If unset, defaults to T.

Manager.MENUROWS [INITVAR]

Maximum number of rows in a manager menu. If unset, defaults to 20.

MANAGER-MARKED-SHADE [INITVAR]

The shade used to indicate that an item has been marked as changed. If unset, defaults to MENUBOLDFONT.

Manager.WINDOW-ANCHOR [INITVAR]

This selects which corner of the FILELST menu window remains anchored so that, as files are loaded, its growth will attempt to remain within the Medley screen.

It must be one of ANCHOR-TL, ANCHOR-TR, ANCHOR-BL, or ANCHOR-BR, where ANCHOR-BL is the default. These correspond to the Top Left, Top Right, Bottom Left, and Bottom Right corners, respectively.

---

## Change History

This is a history of edits made to the Manager. Please add your initials and a short description of what you changed to the END of the file. Be sure to include the name of the definition you modified.

andyiii- All menus are sorted now.

andyiii - Appropriate sub-menu update when something is changed that they contain.

andyiii- un-marking a file in the main menu now works and updates all the sub-menus of that file.

andyiii - added option to MAKEFILE menu item for files to write CommonLisp source using common-makefile.

andyiii - added commonlisp DESCRIBE for items

andyiii - Added a way to add files to the file managers main menu

andyiii - Can edit files property list from CHANGES menu

andyiii - Can now mark a whole file from main menu

andyiii - Can chose between TCOMPL (.LCOM files) and compile-file (.dfasl files) This is awkward since it uses the global variable \*default-cleanup-compiler\*

andyiii - Can get CommonLisp documentation string and descriptons

andyiii - Can now PrettyPrint a value, function def, or prop list and also show how the item would be written to a file

andyiii - Cleaned up specialized menus for FNS, FUNCTIONS, VARS and PROPS

andyiii - All dialog now goes through the MANAGER ACTIVITY WINDOW

RAF 7/31/87 - Fixed the rename option to not specify a source file, uses the ? search (core then file).

RAF 7/31/87 - Added an "edit all occurances of item's name" option to file relations menu.

RAF 7/31/87 - Manager.ACTIVEFLG is now a special that is bound by all advice to avoid redundant updates inside of themselves. This is a big speed improvement!

RAF 7/31/87 - Fixed Manager.HASITEM and Manager.HIGHLIGHT to use SASSOC, so that list items in menus get highlighted properly.

RAF 7/31/87 - Middle button on Manager file menu now brings up rename, etc. Used to bring up coms to edit (inconsistent).

RAF 7/31/87 - Main menu flashes if bad button/command is given.

RAF 8/4/87 - MANAGER-ADDTFILES? now initialized to NIL, reducing redundant updates.

RAF 8/14/87 - In Manager.ALTERMARKING: removed extra code which tracked the files containing updated menus. Removed call to Manager.CHECKFILE. Made call to Manager.MAINUPDATE pass T if the reason for marking was DEFINED or DELETED; these cases also call Manager.COMSOPEN.

RAF 8/15/87 - In Manager.DO.COMMAND: moved binding of ACTIVITY-WINDOW-WAS-SHRUNK into the form eval'ed in the process where references are made. Moved setting of ACTIVITY-WINDOW-WAS-SHRUNK after the spot where its referent ACTIVITY-WINDOW is initialized.

RAF 8/16/87 - Advice for LOAD and LOADFNS now call Manager.CHECKFILE instead of Manager.MAINUPDATE (latter only does highlight updating, former can rebuild main menu). Advice for ADDTOFILES? now doesn't disable manager inside of its advised form, so that the ADDTOCOMS and DELFROMCOMS advice will work.

RAF 8/17/87 - Added Manager.FILESTCHANGED? (which is tricky, since sorting in the main menu changes its order). Manager.CHECKFILE now tests whether the file being checked is in the main menu. If not the main menu is rebuilt. MANAGER fns disables manager around its call to UPDATEFILES. Manager.GETFILE takes a prompt argument (which is now passed in by Manager.DO.COMMAND).

RAF 8/18/87 - Manager.REMOVE.DUPLICATE.ADVISE now disables the manager when it manipulates the advice (to avoid animating the changes in the menus). The advice on LOAD and LOADFNS now call Manager.REMOVE.DUPLICATE.ADVISE.

RAF 8/20/87 - Fixed Manager.MAKEFILE.ADV to handle atomic cleanup options. Also made the top level Manager.RESET call take Manager.ACTIVEFLG, so that manager stays on when reloaded if it was on already. Manager.REMOVE.DUPLICATE.ADVISE now removes \*all\* duplicates of the first piece of advice (rather than only the second).

RAF 8/21/87 - Made MANAGER-WINDOWS be an initvar so that Manager.RESET from top level sees the right thing on first startup.

RAF 9/2/87 - Changed the manager shrunken bitmap to something more respectable. Added ADVISE and UNADVISE menu options for the ADVISE definer. Added a "Show all advice in effect" option to the manager main window middle button menu. Changed the messages printed out by Manager.DO.COMMAND to all use printout and lambdafont for highlighting.

RAF 9/3/87 - Added a clause in the startup fns MANAGER which reports when FILELST is empty and manager can't start. Also fixed a bug in where marking a file didn't bold the main menu entry (added an updatefiles in Manager.ALTERMARKING). Also caused the advice on the "redundant" call to (MARKASCHANGED :IN DEFAULT.EDITDEFA0001) to fire when FILELST is being edited (seems it was the only way to call markaschanged in that one case).

RAF 11/18/87 - Changed the call to EDITDEF in Manager.DO.COMMAND to include a :DONTWAIT option. The tracks a change in SEdit for the Mototwn release.

RAF 11/18/87 - Added some type checking to the sort testing function Manager.SORT.COMS so that it doesn't convert its arguments to strings unless they're not LITATOMS. This should make menu generation alot faster.

MTH 10/13/2023 - Added Manager.WINDOW-ANCHOR to anchor the window for keeping growth on screen. Added the ICON for the shrunken FILELST window. Moved previously conditional FILESLOAD operations to be always. Fixed a multiple-occurrence typo in the MasterScope database operations: LOADBFLG should be LOADDBFLG.

---

---

**MISSILE**

---

---

By: Anonymous

Maintained by: Frank Shih (Shih.envos@Xerox.com)

**INTRODUCTION**

MISSILE is a Lisp version of the video game, Missile Command. It was discovered on a file server after a Lisp class.

**STARTING MISSILE**

Load MISSILE.LCOM, and then call (IL:INIT-MISSILE). Try to destroy incoming missiles by clicking the mouse button in the sky. The rest should be obvious. Warning: the game makes lots of noise.

**RESTARTING MISSILE**

After a game of MISSILE, it can be restarted from the background menu.

---

---

## MODERNIZE

---

---

By Ron Kaplan

This document was last edited in May 2022

MODERNIZE is a simple Lispusers package that changes the mouse actions on Medley windows so that moving and shaping can be done in a way that approximates the behavior of windows on modern platforms, Mac, Windows, etc. It also adds some meta keys to emulate more conventional behavior.

Thus, for a window that has been created or transformed in this way, you can move the window by left-clicking in the title bar and dragging the window's ghost region. Or you can reshape by clicking in a corner of the title bar or near the bottom of the window to drag out the ghost region by that corner.

The menu behavior for other buttons or buttons clicked in other positions is unchanged.

For bottom corners, "near" means inside the window within MODERN-WINDOW-MARGIN (initially 25) pixels above or to the left/right of the corner.

For top corners, "near" means within the title bar and within the margin from the left/right edges. (Windows that don't have a title-bar, like Snap windows, can be set up so that moving can happen by clicking anywhere, and shaping at the top is determined by the margin inside the window region.

The function MODERNWINDOW.SETUP establishes the new behavior for classes of windows:

```
(MODERNWINDOW.SETUP ORIGFN MODERNWINDOWFN ANYWHERE TITLEPROPORTION)
```

ORIGFN is either the name of the BUTTONEVENTFN for a class of windows (e.g. \TEDIT.BUTTONEVENTFN for Tedit windows) or it is a function that creates windows of a particulate kind (e.g. SNAPW or ADD-EXEC).

MODERNWINDOW.SETUP moves the definition of ORIGFN to the name (PACK\* 'MODERN-ORIG-ORIGFN). It then provides a new definition for ORIGFN that does the window moving or reshaping for clicks in the triggering locations, and otherwise passes control through to the original definition.

If ORIGFN is a button event function, then MODERNWINDOWFN should not be specified. In that case a new definition for ORIGFN is constructed to provide the desired windowing behavior.

Otherwise, if ORIGFN is the function that creates windows of a class (e.g. SNAPW), then a MODERNWINDOWFN should be provided to create such windows (by calling (PACK\* 'MODERN-ORIG-ORIGFN)). The definition of MODERNWINDOWFN replaces the original definition of ORIGFN.

TITLEPROPORTION may be a number between 0 and .5. It specifies the proportion from either edge of the title bar where clicks will be interpreted as window operations. For example, if TITLEPROPORTION is .25, then a title click that is up to a quarter of the way from the left or right edge of the window will trigger the moving operation. Clicks in the middle 50% of the title bar will always pass through.

If the flag ANYWHERE is non-NIL, especially for windows without a title bar, then the moving behavior is triggered by a click anywhere in the window (except the corners).

Because this works by redefining existing functions, it is important that the `MODERNIZE` package be loaded AFTER Tedit and Sedit, if those are not already in the sysout. And it should be called to upgrade the proper functions for other window classes that might later be added.

Provided these capabilities are already loaded, the following window classes are "modernized" when `MODERNIZE` is loaded:

```
TEDIT
SEdit
INSPECTOR
SNAP
DEBUGGER
EXEC
TABLEBROWSER
FILEBROWSER
FREEMENU
GRAPHER
PROMPTWINDOW
SPY
TOTOPW
```

If it is not known or it is inconvenient to systematically upgrade a button function or a window-creation function, the new behavior can be provided after a particular window has been created, by invoking

```
(MODERNWINDOW WINDOW ANYWHERE TITLEPROPORTION)
```

This saves the windows existing `BUTTONEVENTFN` as a window property `PREMODERN-BUTTONEVENTFN`, and installs a simple stub function in its place.

If things go awry:

```
(UNMODERN.SETUP ORIGFN
```

is provided to restore the original behavior for windows whose `buttonevent` function is `ORIGFN`.

```
(UNMODERNWINDOW WINDOW)
```

restores a modernized window to its original state.

`MODERNIZE` also augments the Tedit keyboard commands, to increase compatibility with modern interfaces:

```
Meta,q is a synonym for Quit
Meta,a is a synonym for Select All
```

## Known issues:

Clicking at the bottom of an `EXEC` window running `TTYIN` is effective only when the input line is empty.

---

---

## MONITOR

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: COURIERSERVE, BITMAPFNS

MONITOR is a remote screen monitor which shows a scaled down version of the entire remote screen and a small section at full size which can be moved around.

The module contains the code for the client and the server and must be loaded on both. The program supports multiple instances of the tool, even at different scale factors, and works correctly between machines with different size displays.

The lower, full screen window is mouse sensitive. Pressing the left button in the window updates the upper, closeup window to contain the portion of the remote screen indicated by the cursor. Pressing the middle button in the full screen window causes the compressed image of the remote screen in the lower window to be updated.

(MONITOR *HOST* [*SCALE*])

[Function]

Opens a remote screen monitor onto *HOST*, where *HOST* is any specification that COURIER.OPEN accepts. *SCALE* is optional and determines the amount of compression of the remote screen bitmap as well as the amount of area covered by the closeup.

The useful range of scale factors is from **2** to about **8**; a scale factor of **N** will compress the remote screen by **1/N** in width and height and the closeup will cover **1/N<sup>2</sup>** of the area of the remote screen.

MONITOR.SCALE = **3**

[Variable]

If not specified, the *SCALE* argument to MONITOR defaults to the value of MONITOR.SCALE.

### KNOWN PROBLEMS

- The Courier program number that the MONITOR Courier program uses is unregistered.
- The monitor does not yet correct for VIDEOCOLOR (which affects both the client and server).





---

---

## NEATICONS

---

---

By: Peter Schachte (quintus!pds@Sun.com)

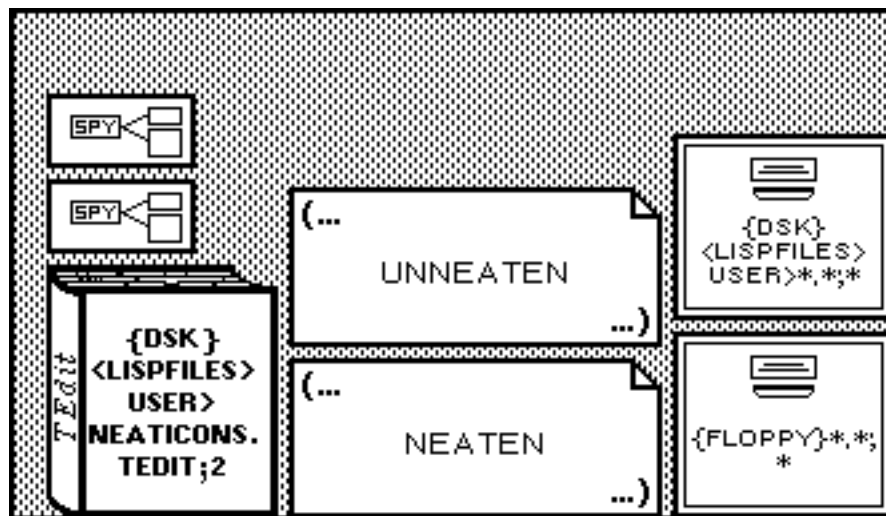
### INTRODUCTION

If you like to keep your icons neatly arranged on your screen, NEATICONS is for you. After this package is loaded, whenever an icon is created by shrinking a window, that icon will be "neat." But what *is* a neat icon? A neat icon is one that is lined up with with another icon or window, or the edge of the screen. The easiest way to see this is to load the package, shrink a few windows (creating a snapshot and shrinking it is easy), and move them around. When a neat icon is moved near another icon or window or the edge of the screen, it is "grabbed" and moved neatly near it.

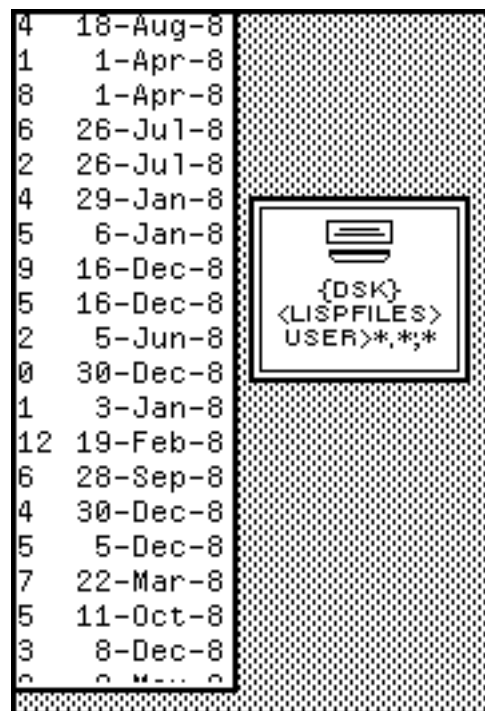
Neat icons line themselves up in a variety of ways. They will flush themselves with the edge of the screen. They will move themselves a fixed number of pixels from the edge of another window. Or they will align one of their edges with the corresponding edge of another window. When you move a neat icon, it will try to find a "neat" position near where you placed it, and place the window there instead. It may find a nearby position that is horizontally neat but not vertically, or vice versa. In any case, it will move the window into the nearest neat position it can find, or leave it where you put it if it can't find any nearby neat places.

### EXAMPLES

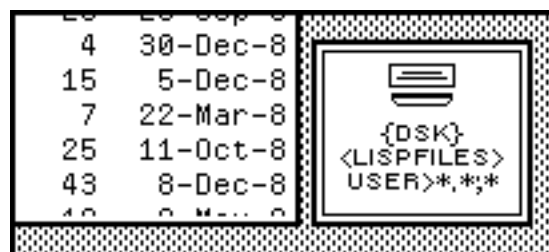
Here are a few examples of how your icons will be arranged. A typical cluster of neat icons:



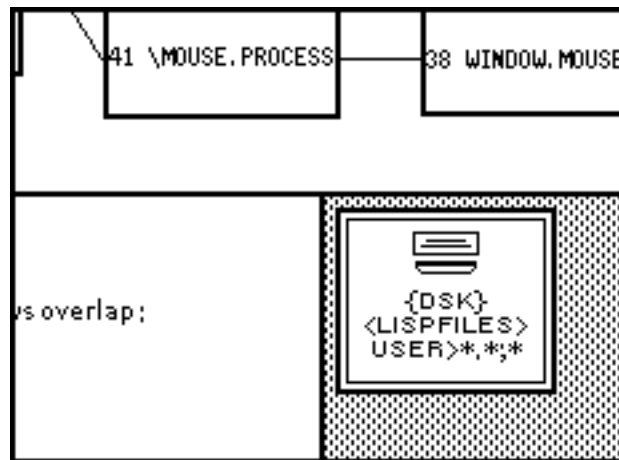
Neat icons can align themselves with the side of a window:



But more typically, they will align themselves with the corner of a window:



And they occasionally align themselves near the corner where two windows overlap:



## DETAILS

You can just load this module and forget about it, and it will behave as advertised. It does have two user-settable parameters, and two user-callable functions, however. These are documented below.

Please note that the NEATICONS module is contained entirely in the NEATICONS package. This package exports the following symbols.

### How near is near?

NEATICONS:DEFAULT-TOLERANCE [Variable]

This global parameter determines how many pixels vertically and horizontally an icon will be moved in order to make it neat. This defaults to 100.

### Spacing between icons

NEATICONS:DEFAULT-SPACING [Variable]

This global parameter determines how many pixels apart neat icons will be placed. The default is 5.

### Making a window neat

Loading the NEATICONS module causes SHRINKW to be advised, so every time a window is shrunk, the icon is made neat. So icons created before you load NEATICONS will not be neat, but they can be made neat by expanding and then re-shrinking them.

But suppose you want to make a regular window neat?

(NEATICONS:NEATEN &OPTIONAL WINDOW) [Function]

makes WINDOW neat. WINDOW defaults to (WHICHW), so you can point at a window with the mouse and type (NEATICONS:NEATEN).

### Making a window sloppy

(NEATICONS:UNNEATEN &OPTIONAL WINDOW) [Function]

makes WINDOW no longer neat. It behaves just like a normal, sloppy, vanilla window. WINDOW defaults to (WHICHW).

**Other MOVEFNs**

The NEAT ICONS module uses each window's MOVEFN prop. If you wish to have another MOVEFN on a neat window, you can. If a window has MOVEFN when it is NEAT ICONS:NEATENed, it will be preserved. If you wish to add a MOVEFN to an existing neat window, you should put it on the window's

NEAT ICONS:USERMOVEFN

[Window Prop]

prop. This prop should hold a list of functions. When a neat window is moved, first it finds the nearest neat place. Then the first function on the window's NEAT ICONS:USERMOVEFN prop is called with the neat position as argument. If this function returns IL:DON'T, the window won't be moved. If it returns NIL, the position argument it was passed is passed to the second function on the list. If it returns a position, this position is passed to the second function on the list. The result of each function on the list is treated similarly, until all the functions have been called. The latest position is used as the position to move the window to.

---

---

## NOTEPAD

---

---

By: D. Austin Henderson, Jr. (AHenderson.pa@Xerox)

Compiled for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

### Auxiliary file: NOTEPAD-CORESTYLES

**NOTEPAD** is a module for creating NOTEPAD windows - windows in which one can do artwork at the bitmap level. The ideas in this module come pretty directly from other people's work, both inside and outside Xerox, including Markup, Draw and Smalltalk. Notepad as it stands is the product of about a man-week of work, using standard Interlisp-D as released and the EDITBITMAP module of bitmap manipulation functions. It provides a nearly unusable interface to some distinctly interesting functionality. Comments and suggestions are welcomed.

### NOTEPAD (BITMAP COLORFLG)

Creates a NOTEPAD window. If BITMAP is NIL, then you are prompted for region for the window. Otherwise, a region is defined from the size of BITMAP, and you are prompted to move it to a desired position. If COLORFLG is non-NIL, the NOTEPAD window will be used only for control (for menu's, etc.), and all the painting will take place on the color screen.

There are two menus: one in the title of the window, and one in the window proper. Both are invoked by buttoning with either left or middle buttons.

#### Title menu

This menu gives access to commands for manipulating the window as a whole:

- New Notepad
- Copy Notepad
- Save as a bitmap.

#### Window menu

This menu has two columns of commands: those in the left column of the menu are for painting/erasing material into/from the bitmap (if this menu is invoked with the left button, painting is implied; if with the middle button, erasing); those in the right column of the menu are for changing the style in which the operations work.

#### Painting/Erasing

You can paint/erase using trajectories, or using (or editing) single objects. The commands in the left column of the menu are divided into two sets which reflect this division.

## Trajectories

Sketch: follows the mouse to define a trajectory of points at which to sketch.

Line: prompts for endpoints (fix and rubberband) and uses the points on the line as a trajectory.

Circle: prompts for center and a point on the circumference, and uses the points on the line as a trajectory.

Ellipse: prompts for center, end of semi-major axis and end of semi-minor axis, and uses the points on the line as a trajectory.

Open curve: prompts for first point and one or more subsequent points. You indicate that you are finished by depressing the left shift key on the last point defining the curve. A smooth curve is fitted through these points and used as a trajectory.

Closed curve: Like open curve, except that the fitted curve is closed, starting at the last point given, and proceeding to the first and then subsequent points.

## Objects/editing

Text: prompts for text, and permits positioning it with the mouse.

Area of the screen: Prompts for a (rectangular) region of the screen and places permits placing them where you want in the window.

Shade rectangle: prompts for a region, and paints/erases it with the current shade (a shade of black does complete paint and erase).

Fill: Prompts for a region within which to fill, and a point within the area to be filled; fills the area (not necessarily a rectangle, but defined by being closed rectilinearly) with the current shade.

Edit area: prompts for a region of the window and invokes the (Trillium) bitmap editor (standard Interlisp-D bitmap editor is the "hand-edit" choice on the submenu; others allow reflecting, rotating, shifting, inverting, putting on borders, etc.) on it. If the bitmap resulting from the edit is the same size as the original, it replaces the original region; if not, you are prompted for a place to put it.

## Style

Notepad operations (see above) are carried out in a style. The style at any given moment is given by a collection of characteristics. The current style can be saved (use the command SAVE.STYLE) under a name and restored (RESTORE.STYLE). Styles may be deleted (DELETE.STYLE) from the collection of saved styles. The style collection is currently part of notepad. Consequently, moving styles between loadups is not directly supported. (It is always possible to save it on a separate file. The styles are stored as the value of NOTEPAD.STYLES. It includes bitmaps, and must therefore be added as an UGLYVARS.)

The characteristics in the style are:

Brush: A bitmap which is either painted or erased at each point on or resulting from (see symmetry) a trajectory (see the operations paint, line, circle, ellipse). DEFINE.BRUSH prompts for a region which will then become the brush. EDIT.BRUSH permits editing of the brush bitmap (using the same editor as the operation EDIT.AREA - see above). BRUSH=COOKIE.CUT.WITH.MASK also defines a brush (see mask, below).

Use mask: An indication of whether or not to use the masking function (see mask, below) before painting/erasing. USE.MASK toggles this setting.

Mask: A bitmap which is used to clear out an area before the brush is used. The mask is erased if the brush is painting, and visa-versa. DEFINE.MASK prompts for a region which will then become the mask. EDIT.MASK permits editing of the mask bitmap (using the same editor as the operation EDIT.AREA - see above). MASK=OUTLINE.OF.BRUSH defines the mask to be the same size as the brush, with a pattern (of black) which is the filled-in outline of the brush. BRUSH=COOKIE.CUT.WITH.MASK allows you to move the mask over a section of the window and define the brush as the points so covered.

Use grid: An indication of whether or not to use the gridding function (see grid, below) while painting/erasing. USE.GRID toggles this setting.

Grid: An origin and the point (1, 1) of a grid to be used to "attract" the points used in following a trajectory. DEFINE.GRID prompts for the origin and (1, 1) point of the grid.

Use symmetry: An indication of what sort of symmetry function to use while painting/erasing. USE.SYMMETRY permits setting it as you choose. The choices are none, left/right, up/down, 4-fold (both left/right and up/down) and 8-fold (4-fold plus reflecting about the 45-degree diagonals). The brush/mask used when painting/erasing symmetrically can themselves be either identical to the brush/mask in use or symmetrically reflected. USE.SYMMETRIC.BRUSH/MASK toggles this setting.

Point of symmetry: The point with respect to which the symmetry functions are defined. POINT.OF.SYMMETRY prompts for this point.

Text font: The font in which text is printed. DEFINE.FONT permits choosing one of the fonts already loaded or OTHER (in which case you can type in the font description (family size face)).

Shade: The shade used for the rectangle and fill operations. EDIT.SHADE permits you to edit the shade (using the standard Interlisp-D shade editor).



---

---

## NOVAFONT

---

---

By: Nick Briggs (Briggs.pa@Xerox.com)

### INTERNAL

By Nick Briggs  
With prodding from Larry Masinter

This utility file allows Lisp to use in fonts in the NOVAFONT format, which is used by Viewpoint.

NovaFont files have in them both the display bitmaps for all sizes of the font, and also the printer widths.

NovaFont files need to be explicitly noticed.

(notice-novafont-file filename) [Function]

After calling notice-novafont-file, the fonts in the given file name will be "known" by the environment. For example,

```
(notice-novafont-file
"{eris}<lispcore>xeroxprivate>fonts>optimamedium.novafont")
```

After this, FONTCREATE will get the bits from the file.

The novafont reader is crafted in such a manner that it only ever reads the file forwards, so you can load fonts from an NS server. For PARC users, note that there are a bunch of NovaFonts (a few malformed...!) on

{starfile public:}<vp applications>\*.novafont

including all the "printwheel" fonts, all the Japanese and Chinese character sets (60Q thru 140Q approx), the PC fonts, Quartz .

Unfortunately, it currently doesn't die "gracefully" on the malformed files. It can't just call VP-FONT-P on the files and continue on because it would mean backing up the file pointer.

Instead of loading NovaFonts on demand, all of the display fonts in a NovaFont file can be loaded at once by calling

(load-novafont-file filename) [Function]

---



---

## NSDISPLAYSIZES

---



---

By: Bill van Melle (vanMelle.pa@Xerox.com)

The NS font families all have screen fonts that display at approximately their nominal point size. This means that there is a closer congruence between their appearance on the display and on hardcopy than there is for, say, the Press fonts. Unfortunately, this means that the NS display fonts are "too small"—a size that is quite readable on hardcopy can be uncomfortably small on the display. The module NSDISPLAYSIZES attempts to ameliorate this problem by fooling FONTCREATE into using bigger fonts for display without changing anyone's belief in the nominal size of the font.

Loading NSDISPLAYSIZES.LCOM modifies FONTCREATE's font file lookup procedure so that a request for NS display font of size  $n$  actually reads the font file for size  $n+2$ . For example, (FONTCREATE 'MODERN 10) will actually read the display font file belonging to Modern 12, but FONTCREATE will still believe the resulting font is Modern 10. Font sizes greater than 12 are not affected, on the grounds that those fonts are already big enough to read, and not all fonts are available in size  $n+2$  for large  $n$ ; hence, for example, Classic 12 and Classic 14 will end up using the same actual font for display. Also, since Terminal 14 does not yet (as of this printing) exist, Terminal 12 remains Terminal 12.

A font is considered an NS font if its name is a member of the list NSFONTFAMILIES, whose initial value is (CLASSIC MODERN TERMINAL OPTIMA TITAN).

Loading the module clears the internal font cache of all NS display fonts, so that subsequent calls to FONTCREATE will not erroneously return a font cached earlier under the default lookup procedure. Of course, if someone has already set some font variable to (FONTCREATE 'MODERN 10), that font descriptor will not be affected.

Note that this module has no effect on hardcopy—a font is always printed at the size you named it. And you can still use TEdit's Hardcopy display mode to see how a piece of text will be formatted on the printer.

If the VIRTUALKEYBOARDS module is present, then loading NSDISPLAYSIZES automatically edits its keyboard specifications so that it continues to use Classic 12 in its keyboard displays. Without this fix, the keyboard display routines will try to create Classic 12, which NSDISPLAYSIZES coerces to Classic 14, and as a result the keyboards will be poorly displayed and lack many characters (since Classic 14 is not as complete as 12). If you load VIRTUALKEYBOARDS *after* NSDISPLAYSIZES, you should call (VKBD.FIX.FONT) yourself to make the change.

Note that other modules can have similar problems if they have hardwired into them a specific size of NS font as being appropriate for their display configuration. For such modules to operate correctly in the presence of NSDISPLAYSIZES, they might want to be aware of the function used to coerce sizes:

(NSDISPLAYSIZE *FAMILY SIZE FACE EXTENSION*) [Function]

Returns a font size that (FONTCREATE *FAMILY SIZE FACE*) will use instead of *SIZE*. *EXTENSION* must be a member of DISPLAYFONTEXTENSIONS in order that we know we are doing this for the display. Follows the rules described above. For example, (NSDISPLAYSIZE 'MODERN 12 'MRR 'DISPLAYFONT) returns 14. (NSDISPLAYSIZE 'GACHA 12 'MRR 'DISPLAYFONT) returns 12.

In the simplest case a module could just test for the existence of `NSDISPLAYSIZE` and choose one size or another. For example,

```
(SETQ MYFONT (FONTCREATE 'MODERN (if (GETD 'NSDISPLAYSIZE)
 then 10
 else 12)))
```

---

---

## NSPROTECTION

---

---

By: Bill van Melle (vanMelle@Xerox.com)

### INTRODUCTION

The module NSPROTECTION provides a tool that enables you to easily change the protection of files and directories on Xerox NS file servers.

To install the module, load the file NSPROTECTION.LCOM. Also, Your NS file server must be running Services release 10.0 or later.

### THE PROTECTION MECHANISM

An NS File Server maintains a protection for each file and (sub)directory on the server. In most cases, the protection is not specified explicitly, but rather is inherited from a file's parent directory, making it easy to maintain consistent protection over an entire branch of the file system hierarchy.

The protection is specified as a set of pairs <access rights, name>. The name can be the name of an individual user or a group. The name can also be a pattern of the restricted form \*:domain:organization, \*\*:organization, or \*:\*. The access rights granted to any particular user are the most general of those in the pairs that match the user's name (by exact match, pattern or membership).

The following five kinds of access rights are independently specified (the term "file" here can also denote a directory in the places where that makes sense):

- Read    The user may read the file's content and attributes. In the case of a directory, the user may enumerate files in it.
- Write    The user may change the file's content and attributes, and may delete the file. In the case of a directory, the user may change the protection of any of the directory's immediate children.
- Add    (Applies only to directories) The user may create files in the directory (i.e., add children).
- Delete    (Applies only to directories) The user may delete files from the directory (i.e., remove children).
- Owner    The user may change the file's access list.

In the case of directories, it is also possible to independently specify the directory's own protection and the protection that its children inherit by default. In most cases, the latter simply defaults to the former, and it is usually best to keep it that way for simplicity. However, there might conceivably be cases where, for example, you would want a user to be able to read the files in a directory, but not be able to enumerate it, or vice-versa.

Note that there can be problems when giving a more lenient protection to a file or directory than to its parents, depending on what software is going to be used to gain access to the file. For example, if your default directory protection grants access only to you, and you want to allow a user to read a

particular file stored in your directory, then you can change the protection on just that file to allow Read access. However, the user will have to know the exact name of the file in order to read it, since she won't be able to enumerate the directory to search for the file. Specifying the exact file name works fine from Lisp, but other software that gets to a file by starting at the top and working its way down through the hierarchy would be unable to get to the file.

## USER INTERFACE

To use the tool, select "NS Protection" from the background menu (if your menu has a "System" item, it's a subitem underneath it), or call the function (NSPROTECTION). You are prompted for a place to position the tool's window. Be sure to leave space below the window for the protection information that will follow.

| NS File Protection Tool |                  |                              |                       |
|-------------------------|------------------|------------------------------|-----------------------|
| <b>Show</b>             | <b>New Entry</b> | <b>Apply</b>                 | <b>Set to Default</b> |
| <b>Type:</b> Principal  |                  | <b>Check:</b> New Names Only |                       |
| <b>Host:</b>            |                  |                              |                       |
| <b>Dir/File:</b>        |                  |                              |                       |

The tool window has four command buttons across the top, two switches labeled **Type** and **Check**, and two fill-in fields for the host and file name. Holding a mouse button down over any of these items for a couple of seconds will display a help message in the prompt window.

To view or change the protection of a file or directory, first fill in the **Host** and **Dir/File** fields. You can edit these fields by clicking with the mouse anywhere inside the existing text (if any), or by clicking with the LEFT button on the boldface label. If you click with RIGHT on the label, then any existing text is first erased. Typing the Next or Return key moves to the next field. [See the FreeMenu documentation for more information about text editing.]

You can either enter the host and directory separately, e.g.,

**Host:** Phylex  
**Dir/File:** Carstairs>Lisp

or enter a file name in the usual Lisp syntax in the **Dir/File** field, e.g.,

**Host:**  
**Dir/File:** {Phylex:}<Carstairs>Lisp>

This latter form is intended to make it easy to copy-select the name of the directory or file from another source, such as a FileBrowser window; the host in the full name overrides any name in the **Host** line.

To see the protection of a file or directory, click on the command **Show**. The protection is displayed as a series of editable one-line windows beneath the main window. In each line is a set of access rights and a Clearinghouse name or pattern to which those rights are granted; for example,

| NS File Protection Tool           |                  |                              |                       |                                                               |
|-----------------------------------|------------------|------------------------------|-----------------------|---------------------------------------------------------------|
| <b>Show</b>                       | <b>New Entry</b> | <b>Apply</b>                 | <b>Set to Default</b> |                                                               |
| <b>Type:</b> Principal            |                  | <b>Check:</b> New Names Only |                       |                                                               |
| <b>Host:</b> Phylex;Research;ACME |                  |                              |                       |                                                               |
| <b>Dir/File:</b> <Carstairs>Lisp> |                  |                              |                       |                                                               |
| <b>Read</b>                       | <b>Wrt</b>       | <b>Add</b>                   | <b>Del</b>            | <b>Own All</b> <b>to:</b> Jonathan Q. Carstairs;Research;ACME |
| <b>Read</b>                       | <b>Wrt</b>       | <b>Add</b>                   | <b>Del</b>            | <b>Own All</b> <b>to:</b> LispDevelopers;Research;ACME        |
| <b>Read</b>                       | <b>Wrt</b>       | <b>Add</b>                   | <b>Del</b>            | <b>Own All</b> <b>to:</b> *;Research;ACME                     |

The highlighted buttons indicate which of the five access rights (Read, Write, Add, Delete, Owner) are granted to the name on the right. If the displayed protection was inherited from its parent subdirectory, rather than having been explicitly set, this fact is noted in the prompt window.

To change the protection of a file or directory, set up the protection entries as desired, then click on the command **Apply**. The usual procedure is to use the **Show** command to see the current protection, then edit one or more entries. Clicking on one of the first five buttons toggles it; clicking on **All** either sets all five (if **All** was previously unhighlighted) or clears all five. In addition, setting either **Write** or **Add** also sets **Read**, since they are of little use without read access (you can, however, clear **Read** if you really meant it). The name following **to** is edited in the same manner as the **Host** and **Dir/File** items above. As with most other places in the system, the name you type can omit the domain and organization, in which case the tool will fill in the local defaults; you can also use nicknames, which will be replaced by the Clearinghouse full names (assuming checking is on).

To add an additional entry, click on the command **New Entry**. This adds a new line to the existing set of protection entries, which you can edit as appropriate. To remove a set of access rights completely for an existing name, either clear all five access buttons (most easily done by clicking once or twice on **All**), or clear the name from the **to** field (by clicking on it with the RIGHT mouse button). Any such cleared lines will be removed by the **Apply** command.

You can also change the protection of a file back to "default" by clicking on the command **Set to Default**. Following this command, the protection of the specified file is inherited from its parent directory. This is usually the best way to "undo" a changed protection, because then any changes to the protection of its parent, or parent's parent, etc., will have the expected effect on all its children.

For the **Apply** and **Set to Default** commands, you may also specify a group of files, rather than a single file, by giving a file pattern—a name with asterisks serving as wild cards to match zero or more characters. Any pattern acceptable to the File Browser can be used. The tool enumerates the specified set of files and applies the specified protection to each. The enumeration is made to all levels (infinite depth), so affects files both in the immediate directory and also in its subdirectories, and subdirectories of those, etc. The enumeration does not, however, include the top-level subdirectory itself; e.g., "<Carstairs>Lisp>\*" matches all files (including subdirectories) anywhere in the directory <Carstairs>Lisp>, but does not include <Carstairs>Lisp> itself.

Note that applying a protection to a directory is different from applying the same protection to the files in it, because of defaulting. If you apply a protection to <Carstairs>Lisp>\*, it changes the protection of every file currently in the directory, but any new files added after the change still inherit the protection of the directory <Carstairs>Lisp>. On the other hand, applying a protection to the directory <Carstairs>Lisp> itself affects all current and future files in the directory, *except* any files that already have an explicit protection currently set. To reduce confusion, it is thus preferable to apply protections

to subdirectories, rather than individual files, if you want to control a whole group of files. If you have a subdirectory containing files of miscellaneous protection that you would like to make uniform, the best procedure is to set the desired protection on the subdirectory itself, and then use the **Set to Default** command with a pattern (e.g., <Carstairs>Lisp>\*) to reset all the individual files to defaulted.

The **Apply** command looks up in the Clearinghouse each of the names in the individual protection entries to make sure that they are valid, and replaces aliases (nicknames) with the canonical names. It then tells the file server to change the protection as indicated. The extent to which the **Apply** command checks names is controlled by the **Check** item in the second line of the tool window. It has four possible settings:

- |                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| New Names Only   | This is the default setting. The tool checks any names that you have entered or changed, but assumes that names returned by the Show command were correct.                                                                                                                                                                                                                                                                                                                      |
| All Names        | The tool checks all names, regardless of source. You might want to do this to convert an existing protection entry into canonical form, or check that all the names are still valid.                                                                                                                                                                                                                                                                                            |
| Never            | The tool never checks names; it assumes you meant exactly what you typed. You might want this setting, for example, if one of the names you are entering is registered only in a distant Clearinghouse not currently accessible.                                                                                                                                                                                                                                                |
| I really mean it | Not only does the tool not check the names, it also doesn't balk if you tell it to take certain unlikely actions, such as changing a top-level directory to default protection, setting a completely null protection, or setting a protection in which nobody has Owner rights (which means the protection can only be changed by someone with Write access to the parent, if any). This setting is "one-shot"—it reverts to "New Names Only" after you issue the next command. |

The **Type** item in the second line of the tool window controls which of a directory's two protection attributes is displayed or set. The initial setting is "Principal" and is the one that should normally be used (it coincides with the Lisp file attribute PROTECTION, or "Access List" in NS Filing parlance). The other setting is "Children Only". When the protection type is set this way, the tool deals with the protection that is inherited by default by the directory's children, the attribute called "Default Access List" in NS Filing parlance. Ordinarily, this attribute is defaulted, in which case the directory's principal protection is also used as its children's default protection. Using the **Apply** command changes the Default Access List to the value you specify; using the **Set to Default** command changes it back to defaulted. The **Show** command displays the directory's Default Access List if it has one; otherwise, it displays the principal protection and notes this fact in the prompt window.

The **Type** item is irrelevant for non-directory files (and, in fact, the tool sets it back to "Principal" if it has been changed). When the file is a pattern, the tool always sets the Principal protection; in the case of any subdirectories matching the pattern, it sets the Principal protection to that specified in the window and the Default Access List to "default".

As an additional convenience feature, when you request to **Show** the "Principal" protection of a top-level directory, the tool also displays in the prompt window the directory's current page usage and allocation.

---

## OBJECTWINDOW

---

By Becky Burwell and Ron Kaplan

This document edited on December 21, 2021

An "object window" is a window that contains a sequence of arbitrary image objects arranged either vertically or horizontally. The `OBJECTWINDOW` package provides the functions for creating such a window, adding objects and manipulating them in various ways, and invoking their `IMAGEFNS` functions according to mouse or other signals.

An object window is created by the function `OBJ.CREATEW`:

```
(OBJ.CREATEW WINDOWTYPE REGION/WINDOW TITLE BORDERSIZE NOOPENFLG SEPDIST BOXFN DISPLAYFN
BUTTONINFN) [Function]
```

The arrangement of objects is determined by the obligatory `WINDOWTYPE`, either `VERTICAL` or `HORIZONTAL`. The other arguments are optional. `REGION/WINDOW`, `TITLE`, `BORDERSIZE`, and `NOOPENFLG` are passed to `CREATEW` to create the window. If `REGION/WINDOW` is a window, it is converted to an object window with `TITLE`. Otherwise, a new window with region `REGION/WINDOW` if non-NIL. The objects in the window will be separated (vertically or horizontally) by `SEPDIST` points, defaulting to 0. The arguments `BOXFN`, `DISPLAYFN`, and `BUTTONINFN` are provided as default functions if is convenient to insert objects whose `IMAGEFNS` are not fully fleshed out. `HARDCOPYFN` overrides the default hardcopy function for the window, and `HCPYHEADING` is used instead of `TITLE` for hardcopy output.

```
(OBJ.WINDOWP WINDOW) [Function]
```

True if `WINDOW` is an object window.

```
(OBJ.ADDTOW WINDOW OBJECT) [Function]
```

Adds `OBJECT` at the end of the object sequence in `WINDOW`.

```
(OBJ.ADDMANYTOW WINDOW OBJECTS) [Function]
```

Equivalent to calling `OBJ.ADDTOW` for each object in `OBJECTS`.

```
(OBJ.INSERTOBJECTS WINDOW NEWOBJECTS OLDOBJECT WHERE) [Function]
```

Inserts `NEWOBJECTS` at position `WHERE` (`BEFORE` or `AFTER`) with respect to `OLDOBJECT`.

```
(OBJ.CLEARW WINDOW) [Function]
```

Clears the visible objects in `WINDOW`.

```
(OBJ.DELFROMW WINDOW OBJECT) [Function]
```



OBJECT is removed from WINDOW.

(OBJ.REPLACE WINDOW OLD.OBJECT NEW.OBJECT DONT.REDISPLAY.FLG)

[Function]

Replaces OLD.OBJECT with NEW.OBJECT in WINDOW, redisplaying the visible objects unless DONT.REDISPLAY.FLG.

(OBJ.FIND.REGION WINDOW SEARCHOBJECT)

[Function]

Returns the region in WINDOW occupied by SEARCHOBJECT.

(OBJ.MAP.OBJECTS WINDOW MAPFN)

[Function]

Applies MAPFN to each object OBJ in WINDOW. If MAPFN returns an image object, that object replaces OBJ in WINDOW.

(OBJ.OBJECTS WINDOW)

[Function]

Returns the list of objects in WINDOW.

---

---

**PACMAN**

---

---

By: Michel Denber (Denber.WBST@Xerox.COM)

## INTRODUCTION

PACMAN is a Xerox Lisp (Medley, Lyric or Koto) implementation of the arcade game PACMAN. It is fairly faithful to the arcade version, in terms of screen appearance and game dynamics. It runs equally well on Suns, 1132's, 1108's, 1186's, and 1100's (Maikos, Dorados, Dandelions, Doves, and Dolphins). It can run in color for 1132's and 1100's with color boards. Several different methods of user input are supported.

This document describes operational details only. I assume that you know what Pacman is and how to play it.

Load PACMAN.LCOM from your local Lispusers directory. To start the game, type (PACMAN). It will prompt you via menus for input mode, speed, color, and high scores. The menus are described in the next section.

## OPTIONS

*Input mode:* there are currently four ways to control the Pacman, depending on what hardware you have. Although you are asked to pick a mode whenever you start the game, you may switch modes freely at any point during the game.

**Mouse** - to use this mode, move the mouse in the direction you want the Pacman to go. You only need to move the mouse when you want to change directions.

**Keyboard** - uses the I, J, K, and space bar as cursor control keys. I is up, J is left, L is right, and space-bar is down. If you look at the arrangement of these keys on the keyboard, you will see that they form an approximate cross. Using the space bar for down (rather than M or ";") lets you control each key comfortably with your right hand; use your thumb for space, and index, middle, and ring fingers for J, I, and L respectively. Note that *you must use the upper-case letters*.

**Joystick** - this mode lets you connect an Atari joystick to the keyset port on the Alto keyboard of your 1100 or 1132. See the appendix below for details on constructing the interface cable needed. This is the input mode of choice, since the arcade game also uses a joystick. You can, if you want, plug a keyset (if you can find one) into the keyset port, but it is not clear that this offers any particular advantages for data input. I have heard that it is possible to kludge a keyset port on an 1108, but I don't know how to do that.

**Voice** - perhaps the most exotic input mode, this requires that you have an Interstate Electronics voice recognition board connected to the RS-232 port of your machine. If you possess such a device, call me for specific interfacing details. The commands are "up",

"down", "left", and "right". People walking by in the hall will wonder what's going on in your office.

*Speed:* You can choose one of four speeds for the game. Some effort has gone into making these values similar across different machines, but there's no guarantee they will stay that way. Note that the speed of the game itself will increase (just like the arcade version) as you progress from board to board.

**Fast** - challenging.

**Medium** - not unreasonable.

**Slow** - mellow mode.

**Snail** - mainly for debugging or the very patient.

*Color (y/n):* will run the game on your color display (if you have one). The b/w display is blanked out during color mode.

*Read high scores (y/n):* this option is only available to users on the Xerox Internet, since the high score repository is on this net. If you are an outside or stand-alone user, always answer no to this. The high score list contains the names and score of the top ten players. Note that with the more recent addition of a speed option, this list is somewhat bogus anyway.

Once you have selected your menu options, the game window will appear, along with the message "Insert quarter to start game". You will note a "quarter icon" in the window. Click this to start the game, or click anywhere else to exit. You can skip the introduction screen by clicking the mouse anywhere inside the window while the introduction is in progress.

All of the standard Pacman boards are supported, along with bonus fruit and relative ghost "blue times". You get the correct number of points for eating various objects. On the b/w screen, the ghosts simply invert-video when you eat an energizer; in color they turn blue.

This game is adapted from the author's PDP-11 Fortran version. Call me if you want that one. Please direct all comments, questions, and bug reports to [Denber.WBST@Xerox.COM](mailto:Denber.WBST@Xerox.COM).

### KNOWN BUGS

- Pieces of characters are occasionally left lying about the board following collisions.
- Keyboard steering **MUST** be in uppercase (at least in Medley), lowercase has no effect.

### LIMITATIONS

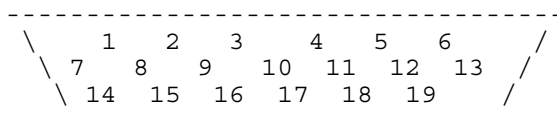
- No sound (that's planned)
- The ghosts do not gain on you as they're chasing you. (You however, can gain on them when you're chasing them).

- The ghosts do not go around turns slower than you do.
- You can't scare ghosts off by feinting at them.
- Only the first intermission is implemented.
- The top ten score probably no longer works, since it depended on an account on {ICE}.
- Color Pacman does not currently work in the Medley release.
- Mouse steering does not currently work in the Medley release.

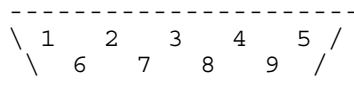
## APPENDIX: PACMAN CONTROL VIA JOYSTICK

1. You must use a switch-closing type joystick (not a pot type). The Atari 2600 (ie. "Video Computer System") or 400/800 computer joystick works fine. The pinouts given below are for this joystick. Signal descriptions are given too, so you can use other similar devices.
2. You will need a male mouse connector (the pins are embedded in the shell, so it's hard to the the sex of this type of connector), a 9 pin male D connector, and a short length of six conductor cable.
3. I will use the following convention for pin numbers since the connectors are not clearly labeled. The views below are from the pin-side of the connectors, ie. not the side that the wires are soldered to.
4. I recommend you check the completed assembly for shorts with a VOM and by running KeyTest (available to 1100's and 1132's from the NetExec) before trying it out on the game. Shorts or opens won't hurt the machine, but they will keep the joystick from working properly (obviously).

### Keyset connector:



### Atari connector:



### Description:

| Atari |        | Keyset |        |
|-------|--------|--------|--------|
| Pin   | Signal | Pin    | Signal |
| 2     | East   | 9      | PAD1   |
| 3     | West   | 10     | PAD2   |
| 4     | South  | 11     | PAD3   |
| 5     | North  | 12     | PAD4   |
| 7     | Ground | 13     | PAD5   |
| 9     | Fire   | 14     | Ground |

### Connections:

| Atari | Keyset                 |
|-------|------------------------|
| 2     | 10                     |
| 3     | 11                     |
| 4     | 12                     |
| 5     | 13                     |
| 7     | 14                     |
| 9     | 9 (not used in PACMAN) |

---

---

## PAGEHOLD

---

---

By: Jon L White

Currently maintained by: Bill van Melle (vanMelle.pa@Xerox.com)

### INTRODUCTION

Loading PAGEHOLD.LCOM redefines the function `PAGEFULLFN` to alter the behavior that occurs when a tty window fills. Rather than inverting the window and waiting indefinitely for type-in, the PAGEHOLD module indicates the hold by an independent notification, and waits for only a specified interval before continuing. Thus, filling the window is no longer a cause for a program to hang indefinitely.

The default behavior of the PAGEHOLD module is to raise a "button" at the corner of the tty window flashing a message, alternating between



and



indicating that output to the window is being held. While in this state, you can release the hold by any of the following means:

Typing any character (of course, the window must own the tty process). This is the same as the old behavior;

Depressing the CTRL key;

Depressing and releasing either SHIFT key;

Clicking with LEFT on the button that announces the hold (clicking instead with MIDDLE gets a menu of options);

Waiting until the timeout has passed (initially, 20 seconds).

When you depress one of the SHIFT keys, the button stops flashing. Output will continue to be held indefinitely as long as one of the SHIFT keys is depressed, even if the timeout passes. If while holding down SHIFT, you depress the CTRL key for a second or so, the button will start flashing again; you may now release CTRL and then SHIFT, and the hold will be maintained without your needing to hold down SHIFT. You can release the hold by any of the means listed above.

If the CTRL key is down when a window fills, output is not held at all. Depressing the CTRL key immediately releases any hold in progress.

The remainder of this document describes ways of tailoring the behavior further.

### Controlling the timeout

One of the primary motivations for the PAGEHOLD module is so that printout to a TTY window does not hang indefinitely when one "page" has filled up. The default release time is in the global variable `PAGE.WAIT.SECONDS`, which comes initialized to 20 seconds; a value of 0 causes immediate release (unless a SHIFT key is already depressed). If a window being held has a `PAGE.WAIT.SECONDS` property, then that value is used instead of the global default.

However, if `PAGE.WAIT.SECONDS` is set to `STOP`, then the hold will not be released by any automatic timeout, nor will it be sensitive to the SHIFT or CTRL key actions. This mode most closely approximates the current Lisp design, except that a pop-up button signals the hold rather than a video inversion (mousing the button will, nevertheless, still effect a release). The message

"Scrolling Stopped"

appears in the button rather than one of the several "holding" messages.

### The Pop-up "Buttons"

A secondary motivation for this facility is to have a pop-up "button" that interactively signals the user of a holding condition on a particular window without obscuring the window's contents, as video inversion does. In addition, the button permits the selective release of a particular window by mousing the button (note that holding down SHIFT, on the other hand, would affect *all* windows currently being held). There are three styles of buttons—`WINKING`, `FLASHING`, and `NIL`—and the selection is determined by the value of the global variable `PAGE.WAIT.ACTIVITY`, which comes initialized to `WINKING`. If a window has a `PAGE.WAIT.ACTIVITY` property, then that value is used instead of the global default, thus allowing different types of buttons on different windows.

A `WINKING` button is a fairly hefty pad—approximately 1/2" by 2 1/2"—which pops up just over the right side of the window's title bar; it will alternately print and clear two short holding messages: one in the upper half of the "button" and one in the lower half. A `FLASHING` button is about the same width, but half the height, and will alternately print the two holding messages. A `NIL` button merely shows the message "Release SHIFT for more".

LEFT-mousing any button causes an immediate release of the hold; MIDDLE-mousing the button brings up a menu offering several options. One of these is "Release this hold!", same as using LEFT. Other menu options permit conversion of the hold to indefinite "hold" or to `STOP` mode; additionally, five options are offered for setting the window's specific `PAGE.WAIT.SECONDS` property.

The `WINKING` button has a different pattern of activity when the hold is placed into indefinite hold mode, but the other button styles do not visibly distinguish this state. If there isn't room to place the button down over the right side of the title bar (because, for example, the window is too close to the screen top), then it will be placed over another corner of the window.

### Keyboard Input and Typeahead

Consistent with Lisp's current action, there will be no holding on a window which is its process's `TtyDisplayStream` and for which there is typeahead in that process's TTY input buffer. This action can be overridden by setting `PAGE.WAIT.IGNORETYPEAHEAD` to a non-NIL value: typeahead does not inhibit the hold, character input does not release the hold, and no input is ever discarded (note that depressing the SHIFT and/or the CTRL keys does not generate character input). This feature is intended for those who dislike not knowing whether a keystroke will be consumed by the `PAGEFULLFN`—under the default behavior, if the TTY input buffer is empty, then the first character you

type will either (a) release a hold already in progress and be discarded, or (b) prevent subsequent holds and be retained, all depending on when exactly you type the character.



---

---

**PHONE-DIRECTORY**

---

---

By: smL (Lanning.pa@Xerox.com)

Requires: Lispuser package GREP

**INTRODUCTION**

The PHONE-DIRECTORY package provides quick and easy access to on-line phone directories.



When you load the PHONE-DIRECTORY package, an icon () will appear on your screen. Opening this icon will start a directory search program (see example below). To look up a person, type in a name, followed by a return. The PHONE-DIRECTORY program will search the available phone lists and print out those entries that contain the name. This can be repeated as many times as you like. When you are done, just SHRINK the window back to its icon.

---

```
Phone directory
Name: lennon
(from {CORE}<REGISTRAR>PARCPHONELIST.TXT;1)
4243 Lennon, John 35-1312
Name:
```

Example PHONE-DIRECTORY window

---

**Variables**

PHONELISTFILES

[Variable]

PHONELISTFILES is a list of files that contain phone lists. This is usually set in the INIT.LISP file. The PHONELISTFILES should be unformatted files with a single entry per line. Blank lines are permitted. A typical value for PARC users (as defined in PARC-INIT) is

```
(({ INDIGO } <REGISTRAR> PARCPHONELIST.TXT
 { INDIGO } <REGISTRAR> ISDNORTHPHONELIST.TXT) .
```

\*Phone-Directory-Pos\*

[Variable]

\*Phone-Directory-Pos\* is the initial POSITION for the PHONE-DIRECTORY icon. This is defined as an INITVAR in the file, so you can set it before loading the file. The default value is

```
(create POSITION XCOORD ← 15
```

```
YCOORD ← (DIFFERENCE SCREENHEIGHT 75)).
```

This places the icon in the upper left corner of the screen, just above the exec window.

\*Phone-Directory-Region\*

[Variable]

\*Phone-Directory-Region\* is the initial REGION for the PHONE-DIRECTORY window. This is defined as an INITVAR in the file, so you can set it before loading the file. The default value is

```
(CREATEREGION 15 (DIFFERENCE SCREENHEIGHT 258)
```

```
400 250) .
```

This places the window in the upper left corner of the screen.

#### Notes

When you first open the PHONE-DIRECTORY window, the program will copy the PHONELISTFILES to {CORE}, significantly speeding up queries. Bugging in the title of the PHONE-DIRECTORY window with the left or middle mouse button will produce a menu with an option to recache the files from PHONELISTFILES.

---

---

## PICK

---

By: Larry Masinter (lmm@iacm.org)  
Uses: GITFNS

This document last edited on 10ug 2022

### INTRODUCTION

When I get frustrated trying to decide what to work on, I use 'pick' to choose a project. It has a bunch of different choices.

### MODULE EXPLANATIONS

|                                        |            |
|----------------------------------------|------------|
| (PICK <i>option choices</i> )          | [Function] |
| pick <i>option choice1 choice2 ...</i> | [Command]  |

### Examples

|                                   |                                                                           |
|-----------------------------------|---------------------------------------------------------------------------|
| pick                              | choose something to work on                                               |
| pick oneof chicken beef tofu pork | choose one of the named items                                             |
| pick issue                        | choose an issue and display it                                            |
| pick issue 23                     | if you give an issue number, just displays it                             |
| pick file                         | choose a file at random: is it where it belongs? can you load it? use it? |
| pick file lispusers               | pick a file on the given (medleydir) directory                            |
| pick project                      | choose a project / repo to work on (from GITFNS GITPROJECTS)              |
| pick file notecards               | chooses a notecards repo subdirectory                                     |

---



---

## Piece-Menus

---



---

By: D. Austin Henderson, Jr. (AHenderson.pa@Xerox.COM)

### INTRODUCTION

This module provides two solutions to the problem of menus with too many items. Which is useful will depend on the inherent structure of the items. 1) CHUNK-MENUS: For use in which the items have the simple structure of one long list: Break the items up into chunks following the ordering of the items in the list, and then provide a menu which presents one of those pieces and a way of getting other menus containing the other pieces. 2) KEYWORD-MENUS: For use when it is possible to associate keywords with the items: Break the items up into the sets which share the same keyword, and then provide a menu which presents one of those chunks and a way of getting other menus with other keywords. As with standard Interlisp-D menus, these specialized menus are created by a "create" function (cf. (CREATE MENU)), and are used with a single "invoke" function (cf. (MENU menu)). The items and other information are as with standard menus, except that the other information is presented to the create function in the form of a Plist.

### CHUNK MENU

A Chunk Menu appears as a single menu, but is really a data structure encompassing a number of pieces, each represented by a menu, between which the operator can move to find the item desired. Each piece is in three parts: a set of "required items" which will appear in all chunks, a set of indicators for all the chunks, and the items in this piece. Each chunk has up to 30 items in it (it can be varied using the CHUNK.COUNT property). If there are fewer than one chunk's worth of items, the CHUNK-MENU behaves just like a standard menu.

(CHUNK.MENU.CREATE *ITEMS PROPERTIES REQUIRED.ITEMS*) [Function]

Creates and returns a Chunk Menu, a data structure of menus each containing some of the (presumably large number of) items.. All items are as with standard menus. The properties understood are: CHUNK.COUNT (see above), TITLE, CENTERFLG, MENUFONT, ITEMWIDTH, ITEMHEIGHT, MENUBORDERSIZE, and MENULAYOUTSIZE. The actual menus are created only when needed.

(CHUNK.MENU.INVOKE *CHUNK.MENU POSITION*) [Function]

Carries out the interaction with the user to select an item from a Chunk Menu. If *POSITION* is non-NIL, the menu will appear at that position, otherwise it will appear under the mouse. All interactions are as with the standard menus. Returns the value produced when a selection is made. Selecting outside any of the menus appearing in the interaction cancels the interaction and returns NIL.

### KEYWORD MENU

A keyword menu appears as a single menu, but is really a data structure encompassing a number of menus which the operator can move between to find the item desired. Each piece is in two parts: the set of keywords for all the pieces, and the items in this piece (having this keyword). Each piece is a Chunk Menu, so that if a particular keyword has many items, its piece is itself broken into pieces. The items in a keyword menu are computed from a list of objects, the things which presumably are being selected among. A function is provided for computing the keywords of each object, and another for computing an item from an object.

(KEYWORD.MENU.CREATE *OBJECTS KEYWORDFN PROPERTIES ITEMFN*) [Function]

Creates and returns a Keyword Menu, a data structure of pieces associated with the keywords of the *OBJECTS* as determined by *KEYWORDFN*. Each piece of the Keyword Menu is a Chunk Menu and contains the items for the objects with the associated keyword. The item is computed from the object by applying the *ITEMFN* to that object; *ITEMFN* should return an item appropriate for standard menus. The menus are determined by the values on the PList *PROPERTIES*. The properties understood are: CHUNK.COUNT (see Chunk Menus), TITLE, CENTERFLG, MENUFONT, ITEMWIDTH, ITEMHEIGHT, MENUBORDERSIZE, and MENULAYOUTSIZE. The actual menus are created only when needed.

(KEYWORD.MENU.INVOKE *KEYWORD.MENU POSITION* ) [Function]

Carries out the interaction with the user to select an item from a Keyword Menu. If *POSITION* is non-NIL, the menu will appear at that position, otherwise it will appear under the mouse. All interactions are as with Chunk Menus, which is just that for standard menus for small numbers of items. Returns the value produced by the items when a selection is made. Selecting outside any of the menus appearing in the interaction cancels the interaction and returns NIL.

---

---

## PLOT

---

---

By: Jan Pedersen (pedersen.PA @ Xerox.com)

Uses: TWODGRAPHICS and PLOTOBJECTS

PLOT is a module designed to assist in the production of analytic graphics. PLOT provides automatic scaling, labeling, incremental modification, generalized selection, and a collection of standard graphics primitives which may be combined to produce interactive plots of great diversity.

PLOT is to some degree object-oriented. The primitive components of a plot are plot objects (e.g. points, lines, etc.). A plot manager maintains a display list of plot objects which are individually responsible for displaying themselves, highlighting themselves, etc. The user constructs a plot incrementally, adding plot objects, while the plot manager handles details such as the appropriate scale for the plot. Each plot object is active, in the sense that it is selectable and may have a menu associated with it. In addition, the plot manager may be directed to modify the appearance of the entire plot through a command menu.

The module is open, in the sense that most default behaviors may be overridden by the user, although it is hoped that the defaults will be sufficient for most applications. A functional interface is provided for programmatic access to all of PLOT's facilities.

The plot manager is abstracted as a datatype of type PLOT, along with a collection of functions which operate on PLOT's. Functions are provided to create PLOT's, manipulate their display lists, and modify default menus. Plot objects are abstracted as instances of datatype PLOTOBJECT. A set of default plot objects are provided, along with a mechanism of defining new plot objects.

Plots exist independently of their representation on the screen. Indeed, it is intended that plots may be displayed on ANY imagestream. However, the most common usage is to display a plot in a window, and a PLOT does have an associated WINDOW which may be opened, closed, etc.

Plots may be hard copied, made into image objects, and dumped to file.

The lispuser's module PLOTEXAMPLES contains a few examples of how PLOT may be used to create high level plotting facilities.

### BASIC OPERATION

A plot is abstracted as an instance of datatype PLOT which includes a display list, a property list, and an associated window, among other things. PLOT's may be create via the function CREATEPLOT.

(CREATEPLOT *openflg region title border*)

[Function]

Returns a PLOT. If openflg is T then the PLOT's associated window is opened with an empty plot. The other arguments are treated as in CREATEW.

An empty plot is initialized to have a world coordinate system extending from 0.0 to 1.0 on either axis, with no labels or tic marks displayed. As objects are added to the plot, the world coordinate system is grown to accommodate the new objects.

A PLOT has an associated window, which is closed by default. The window is used as the primary display device and may be manipulated with the following functions.

(OPENPLOTWINDOW *plot*) [Function]

Opens the plot's associated window.

Returns the associated window.

(CLOSEPLOTWINDOW *plot*) [Function]

Closes the plot's associated window.

(REDRAWPLOTWINDOW *plot*) [Function]

Redraws, by running down the current display list, the contents of the associated window. Opens the window if it is closed.

(GETPLOTWINDOW *plot*) [Function]

Returns the window associated with plot.

(WHICHPLOT *x y*) [Function]

Returns the PLOT associated with the window (or icon) at position (*x . y*), or at the current cursor position if *x* and *y* are defaulted. *x* may be a WINDOW, in which case the associated PLOT is returned.

A plot object is abstracted as an instance of datatype PLOT OBJECT. A point plot object is an instance of PLOT OBJECT whose data component describes a point. That is, a point plot object is a subtype of PLOT OBJECT; all plot objects satisfy (type? PLOT OBJECT FOO), but only a point plot object satisfies in addition (PLOT OBJECT SUBTYPE? POINT FOO). A collect of standard plot objects has been implemented, including point, curve, polygon, line, and filled rectangle plot objects. The module is designed so that new objects may be defined at any time, but that mechanism is described in a separate document.

PLOT OBJECT's may be added to or deleted from a PLOT. The following functions provide an add facility for the standard objects.

(PLOTPOINT *plot position label symbol menu nodrawflg*) [Function]

Only the plot and position arguments are required. Position is a POSITION in world coordinates. Label is an expression which will be PRIN1 'ed whenever a label is required (typically an atom or a string). Symbol is a BITMAP which will be plotted centered at position. The litatoms CROSS, CIRCLE, STAR are bound to convenient BITMAPS. Symbol defaults to STAR. Menu is either a MENU, a litatom, in which case a MENU of that name must be cached on plot (more about this later), or an item list which may be coerced into a MENU.

If nodrawflg is non-NIL then a point object will be added to the display list of plot, but the associated window will not be updated. If Nodrawflg is NIL, and the plot's associated window is not open, it will be opened.

Returns a POINT PLOT OBJECT.

(PLOTPOINTS *plot positions labels symbol menu nodrawflg*) [Function]

As above except that positions is a list of POSITIONS and labels may also be a list. Reasonable things happen if positions and labels are of unequal length.

Returns a list of POINT PLOBJECT's.

(PLOTCURVE *plot positions label style menu nodrawflg*) [Function]

The list of POSITION's defines a piecewise linear curve. Style may be an integer which specifies the line width (in pixels) or a list of (linewidth dashing color), any of which may be NIL; defaults to one. For convenience the atoms DOT, DASH and DOTDASH have been bound to a few dashing patterns.

Returns a CURVE PLOBJECT.

(PLOTPOLYGON *plot positions label style menu nodrawflg*) [Function]

As in PLOTCURVE, although a polygon is a closed figure

Returns a POLYGON PLOBJECT.

(PLOTTEXT *plot position text label font menu nodrawflg*) [Function]

Text should be a STRING to be printed at position.

Returns a TEXT PLOBJECT.

(PLOTFILLEDRECTANGLE *plot left bottom width height label  
texture borderwidth menu nodrawflg*) [Function]

Texture must be TEXTURE. SHADE1, ..., SHADE8 are bound to some convenient textures. Defaults to SHADE3.

Returns a FILLEDRECTANGLE PLOBJECT.

The following two functions add analytic plot objects to the display list of a PLOT. Analytic objects differ from points, curves, etc. by having infinite extents; their appearance on a plot depends on the current world coordinate scale, but adding an analytic object to a plot will not effect the current scale.

(PLOTLINE *plot slope constant label style menu nodrawflg*) [Function]

Slope and constant define an analytic line,  $y = \text{slope} * x + \text{constant}$ . If slope is NIL, it is taken to be infinite; i.e. the line is vertical.

Returns a LINE PLOBJECT.

(PLOTGRAPH *plot graphfn nsamples label style menu nodrawflg*) [Function]

Graphfn should be a function of one variable which defines a graph (or the graph of a function) to be drawn on plot. Nsamples is the number of equispaced points along the x-axis of plot at which graphfn is to be sampled when drawn; defaults to 100.

Returns a GRAPH PLOBJECT.

Complex objects may be built up from the preceding primitives by defining a compound plot object, which is simply a collection of other plot objects, including other compound objects.

(PLOTCOMPOUND *plot component1 ... componentn typename label  
menu nodrawflg*) [NoSpread Function]

A compound plot object is specified by listing its components. In addition, a compound plot object may have its own menu and label. The typename field is supplied to allow different compound objects to be differentiated. Drawing a compound object amounts to drawing its components recursively. In general, operations on compound objects are applied recursively.



Components 1 through n are plot objects. Typename is required and serves to tag this compound object, and is accessible via the function COMPOUNDSTYPE. Label and menu are as in other plot objects.

Returns a COMPOUND PLOBJECT.

All plot objects may be created independently of the previous functions. This is useful if it is desired to create a plot object without entering it on a PLOT's display list. The following functions create and return the standard plot objects.

(CREATEPOINT *position label symbol menu*) [Function]

Returns a POINT PLOBJECT.

(CREATECURVE *positions label style menu*) [Function]

Returns a CURVE PLOBJECT.

(CREATEPOLYGON *positions label style menu*) [Function]

Returns a POLYGON PLOBJECT.

(CREATETEXT *position text label font menu*) [Function]

Returns a TEXT PLOBJECT.

(CREATEFILLEDRECTANGLE *left bottom width height label texture style menu*) [Function]

Returns a FILLEDRECTANGLE PLOBJECT.

(CREATELINE *slope constant label style menu*) [Function]

Returns a LINE PLOBJECT.

(CREATGRAPH *graphfn nsamples label style menu*) [Function]

Returns a GRAPH PLOBJECT.

(CREATECOMPOUND *compoundtype components label menu*) [Function]

Components must be a list of PLOBJECT's.

Returns a COMPOUND PLOBJECT.

Each PLOT has a display list which is nothing more than a list of plot objects. The display list may be manipulated directly via the following functions.

(ADDPLOBJECT *plotobject plot nodrawflg*) [Function]

Interns plotobject on the display list of plot, and updates the associated window. The update is suppressed if nodrawflg is non NIL.

One might think of PLOTPOINT as being equivalent to:

```
(ADDPLOBJECT (CREATEPOINT position) plot nodrawflg)
```

Interns plotobject on the display list of plot, and updates the associated window. The update is suppressed if nodrawflg is non NIL.

Returns plotobject.

(DELETEPLOT OBJECT *plotobject plot nodrawflg nosaveflg*) [Function]

Deletes *plotobject* from the display list of *plot*, and updates the associated window accordingly. The update is suppressed if *nodrawflg* is T. If *nosaveflg* is T, then the deleted object will not be saved for possible later undeletion.

Returns *plotobject* if it was deleted from the display list, else NIL.

A PLOT has collection of properties, some of which are maintained by the plot manager, and others which may be used to cache arbitrary user data. All plot properties are accessed via the function PLOTPROP.

(PLOTPROP *plot prop newvalue*) [NoSpread Function]

If *newvalue* is absent then the current value of *prop* is returned. If *newvalue* is supplied (even if it is NIL) then the value of *prop* is set and the old value returned. The distinguished props PLOT OBJECTS, PLOTSCALE, SELECTED OBJECT, PLOTWINDOW, PLOTWINDOWVIEWPORT, PLOT PROMPTWINDOW, and PLOTSAVELIST refer system maintained properties *plot*, and should be treated as read only. Compiles open in some cases.

For example, The display list of *plot* may be accessed by the expression.

```
(PLOTPROP plot 'PLOT OBJECTS)
```

For convenience in manipulating the property list of a PLOT, the following functions are provided.

(PLOTADDPROP *plot prop itemtoadd firstflg*) [Function]

If the value of *prop* is a list then *itemtoadd* is added to the end of the list. If the value of *prop* is NIL, it is set to (LIST *itemtoadd*). *Firstflg* indicates that the new item is to be the first in the list rather than the last. Works only for user defined properties.

Returns the new value.

(PLOTDELPROP *plot prop itemtodelete*) [Function]

If *itemtodelete* is a member (MEMB) of the *prop* value, it is deleted. Works only for user defined properties.

Returns NIL if nothing was deleted, else the new value of *prop*.

(PLOTREMPROP *plot prop*) [Function]

Destructively removes *prop* from property list of *plot*. Works only for user defined properties.

Each plot object also has a property list. As with PLOT's, some of the properties are maintained by the system, but the rest may be used to store arbitrary data objects. The property list of a plot object is accessed through the function PLOT OBJECTPROP.

(PLOT OBJECTPROP *object prop newvalue*) [NoSpread Function]

As in PLOTPROP. The distinguished props are OBJECTMENU, OBJECTLABEL, and OBJECTDATA. The property, OBJECTMENU, may be set as well as read; if the *newvalue* is a list of items, it will be coerced into a menu.

(PLOT OBJECTADDPROP *object prop itemtoadd firstflg*) [Function]

As in PLOTADDPROP. *Firstflg* indicates that the new item is to be the first in the list rather than the last.

(PLOTOBJECTDELPROP *object prop itemtodelete*)

[Function]

As in PLOTDELPROP.

### DEFAULT MOUSE BUTTON ACTIONS

The user may interact with a plot through its associated window. A plot provides two default menu's, the RIGHT menu, which pops up if the right mouse button is depressed within a plot's window, and typically contains items relevant to the plot as a whole, and the MIDDLE menu, which pops up if the middle mouse button is depressed, and typically contains items relevant to the currently selected plot object. The left mouse button is used exclusively for selection. The right menu may optionally be fixed to the right hand side of the plot window for easy reference. In summary:

#### Left Button

While depressed will select the closest plot object.

#### Middle Button

Pops up a menu of default actions on the selected object

#### Right Button

Pops up a menu of default actions on the plot as a whole

### DEFAULT MIDDLE MENU ITEMS

#### Label

Label the selected object. Either a default location for the label is selected (for point plot objects), or the user is queried for a location.

#### Unlabel

If the object is label, remove the label.

#### Relabel

Change the object's label

#### Delete

Remove the object from the plot. May be undeleted later.

### DEFAULT RIGHT MENU ITEMS

#### Layout

Create a SKETCH of the contents of the PLOT. Requires SKETCH and SKETCHSTREAM to be loaded.

#### Redraw

Redraw the plot

#### Rescale

Compute a new scale for both the X and the Y axis based on the objects currently displayed. May also rescale the X or Y axis separately.

**Extend**

Extend the axes slightly on either side so plot objects occurring on the borders may become visible. May be applied separately to either axis.

**Labels**

Change the marginal labels. May either Choose a margin explicitly, or respond to query.

**Tics**

Enable or disable marginal tics.

**Undelete**

Restore the last plot object deleted. Subsidiary items allow selected objects to be restored.

**Deselect**

Deselects the current selected object.

The default menus may be altered or superceded altogether. Each plot object may either use the default middle menu, another cached menu, or provide its own individual menu.

Menus are described by item lists of the form (label function helpstring [(subitems ....)]). Function may be a litatom in which case the function is called with one argument, plot, for right menu items, or two arguments, plotobject and plot, for all other menus. If function is a list the CAR of the list is a APPLIED to (CONS PLOT OBJECT (CONS PLOT (CDR list))), etc.

The following functions facilitate modifying existing menus, and creating new menus.

(PLOTMENU *plot menuname newmenu*) [NoSpread Function]

Plot and menuname are required. If newmenu is not present, then the current value of menu menuname is returned. Menuname may be RIGHT or MIDDLE, in which case the default menus are referred to, or any LITATOM, in which case the cached menu by that name is referred to. Menus other than RIGHT or MIDDLE will typically be specialized menus for particular plot objects. If present, newmenu must be a MENU.

(PLOTMENUITEMS *plot menuname menuitems*) [NoSpread Function]

Plot and menuname are required. If menuitems is not present, then the current item list for the MENU menuname is returned. If menuitems is present, then menu menuname is replaced with a new menu with items list menuitems. All the properties (if any) of the old menu are copied over. Menuname may be one of RIGHT or MIDDLE, in which case the operations refer to the default right or middle mouse button menus or any other LITATOM, in which case the operations refer to a menu cached on plot by that name. Menus other than RIGHT or MIDDLE will typically be specialized menus for particular plot objects.

(PLOTADDMENUITEMS *plot menuname itemstoadd*) [Function]

Itemstoadd must be a list of menu items. Adds each item in itemstoadd to the end of the item list for menu menuname and replaces menu menuname with a new MENU having the appropriate item list.

Returns the the new item list for menuname.

(PLOTDELMENUITEM *plot menuname itemstodelete*) [Function]

Itemstodelete must be a list of items. For each element of itemstodelete, if it is a LITATOM, then deletes the item whose CAR is EQ to it. If it is a LISTP, then deletes the item EQUAL to it. Replaces menu menuname with a new MENU having the appropriate item list.

Returns NIL if no items were deleted, else the new item list.

(PLOT.FIXRIGHTMENU *plot fixedflg*) [NoSpread Function]

Fixedflg is optional. If not present that the current state of the right menu of plot is returned; T implies the right menu is fixed. If Fixedflg is supplied the right menu state is correspondingly changed.

The middle button menu for a particular plot object is a property of that plot object, and may be accessed via the function PLOTOBJECTPROP. For example, the expression,

```
(PLOTOBJECTPROP object 'OBJECTMENU)
```

will return the current middle button menu for object. If the OBJECTMENU property is NIL, then the system default MIDDLE menu is used, if it is a LITATOM, then a specialized cached menu by that name is used, finally, if it is a MENU, then that menu is used.

Two default fonts are provided, a large font for labels and a small font for tic marks. Both may be reset and that aspect of a plot will change accordingly with the next redraw.

LARGEPLOTFONT [Variable]

Default value: (Gacha 12 BRR)

SMALLPLOTFONT [Variable]

Default value: (Gacha 8 MRR)

### Detailed Operation

Most visible aspects of a PLOT may be changed programmatically. The following functions allow the user to specify labels, etc., as well as override the default algorithms for drawing tics, etc.

(PLOTLABEL *plot margin newlabel nodrawflg*) [NoSpread Function]

Plot and position are required. Margin must be one of TOP, BOTTOM, LEFT, OR RIGHT. If newlabel is absent, then the current margin label is returned (may be NIL). If newlabel is present then the margin label is set to newlabel. The display is automatically updated unless nodrawflg is non NIL.

(PLOT TICS *plot margin newvalue nodrawflg*) [NoSpread Function]

Plot and margin are required. Margin must be one of TOP, BOTTOM, LEFT, OR RIGHT. If newvalue is absent, returns the tic status of that margin. NIL implies no tics or labels, T implies both. If newvalue is present, then sets margin's tic status. The display is automatically updated unless nodrawflg is non NIL.

The appearance of the tic marks will also depend on the tic generation method employed. The default is simply to make down tics at "pretty" intervals from the max to the min of each axis in world coordinates. However, non-numeric tic marks, and other behaviors are user specifiable by the function PLOTTICMETHOD.

(PLOTTICMETHOD *plot margin newmethod nodrawflg*)

[NoSpread Function]

Plot and margin are required. Margin must be one of TOP, BOTTOM, LEFT, OR RIGHT. If newmethod is absent, returns the current tic method for margin margin. Newmethod may be one of NIL, implying the default tic method, a list of CONS pairs ( value . label ), in which case label (if non-NIL) will be printed at value, or a list of numbers, which is equivalent to ((value . value) ...) or a function which will be called with args, margin plotscale plot, and should return a list as above. Plotscale is a datatype which describes the current scale of the plot.

(DEFAULTTICMETHOD *margin plotscale plot*)

[Function]

The result depends on the ticinfo field of plotscale, which should be an instance of the PLOTSCALE datatype. The ticinfo field will be an instance of datatype TICINFO. If its ticinc field is a number (the usual case) then it returns a list of numbers, starting at ticmin and ending at ticmax in increments of ticinc, otherwise returns ticinc (should be a list).

When a plot object is added to a plot, the scale of the plot is adjusted so that the object is visible. This is accomplished by comparing the extent (in world coordinates) of the object with the current scale of the plot. If the scale needs to be enlarged, a new interval is chosen for each axis which is guaranteed to include the object and also be some multiple of a "round" increment -- in other words, a pretty tic interval. The default behavior of this scaling algorithm may be altered in several ways.

The pretty tic interval is determined by the TICFN for each axis. The default uses the function SCALE to find a suitable interval. This may be altered by supplying a TICFN other than the default.

Given a pretty tic interval, the default is to simply use the end points of that interval as the endpoints of the scale for each axis. This may be altered by supplying a SCALEFN other than the default.

In other words the actually displayed interval (for each axis) in world coordinates (what I will call the plot interval) is separated from the pretty tic interval (for each axis). The pretty tic interval is computed first, then the plot interval is computed in the presence of that information. This separation is useful if the user wishes to plot objects in a coordinate system different from the one used to display tic marks.

The current state of each axis of a PLOT is cached in the plot property plotscale, whose value is an instance of datatype PLOTSCALE. A PLOTSCALE has three fields for each axis, one which contains an instance of AXISINTERVAL, describing the actual plot interval for that axis, another which contains an instance of TICINFO, which describes the pretty tic interval for that axis, and a third which is a simply a place to cache a user supplied TICFN and SCALEFN.

(PLOTICFN *plot axis ticfn nodrawflg*)

[NoSpread Function]

Ticfn is optional. If not present the current ticfn for the indicated axis is returned. If supplied, the state of that axis is correspondingly updated. A ticfn is called with args min, max, and plot and should return an instance of TICINFO. If the state of plot is changed, the appropriate axis is rescaled. A value of NIL implies the default ticfn.

(DEFAULTTICFN *min max -- -- --*)

[Function]

The default ticfn for each axis. Uses the function SCALE to find a suitable pretty tic interval.

(PLOTSCALEFN *plot axis scalefn nodrawflg*)

[NoSpread Function]

Scalefn is optional. If not present the current scalefn for that axis of plot is returned. If supplied, the state of that axis is updated. A scalefn is called with four arguments, the min and max extent (in world coordinates) on that axis of the plotobjects currently displayed, the TICINFO for that axis, and the plot;

the scalefn should return an AXISINTERVAL which will determine the scale for that axis of plot. A value of NIL implies the default scalefn.

(DEFAULTSCALEFN *min max ticinfo*) [Function]

The default scalefn for each axis.

Returns an AXISINTERVAL with endpoints identical to the endpoints of ticinfo.

(ADJUSTSCALE? *extent plot*) [Function]

Determines whether extent will fit into the current viewing area of plot. If so, returns NIL. If not, returns T and updates the plotscale of plot.

(EXTENTOFPLOT *plot*) [Function]

Computes the current extent of plot by mapping EXTENTOBJECT down the display list. Returns an EXTENT.

To be precise, the scaling algorithm operates as follows; a min and max extent of the data is computed (via EXTENTOFPLOT or entered manually in the case manual rescaling), then CHOOSSETICS is called, which returns an instance of TICINFO. CHOOSSETICS either uses a default TICFN, or one supplied by the user. The default TICFN, calls SCALE repeatedly to find an "optimal" tic interval in world coordinates. Once the TICINFO instance has been computed, CHOOSSCALE is called with the original min, max and the TICINFO, and returns an instance of AXISINTERVAL, which will determine the actually displayed plot interval. Again, CHOOSSCALE either uses a default SCALEFN, or one supplied by the user. The default SCALEFN simply uses the end points of the passed in pretty tic interval as the end points of the AXISINTERVAL which it returns. Finally, the PLOT is redrawn with the new scale -- notice that the plot interval may either be larger or smaller than the pretty tic interval; the margin drawing routines are robust enough to deal with all cases.

For example, suppose the world coordinates are in centigrade and it is desired to produce a pretty tic interval in units of Fahrenheit (this is an easy case since the transformation between scales is linear -- more about that later). The user would then supply a TICFN which would transform the incoming min and max to Fahrenheit, apply the default TICFN on the transformed min and max, obtain a TICINFO in Fahrenheit, transform the fields of that record back to Centigrade, and return that record. Note, it is always assumed that the fields of a returned TICINFO are in the units of the world coordinate system. The rest of the machinery would then go through as before.

A trickier example is one in which it is desired to produce unequispaced tic marks. Suppose the data were plotted on a log scale (that is, log was applied BEFORE plotting the data). The default algorithm would produce a pretty tic interval in the log scale. It might be desired instead to produce one pretty in the original scale. The user would then supply a TICFN which would exponentiate the incoming min and max, apply the default TICFN on the transformed min and max, obtain a TICINFO in the original scale, then return a TICINFO in the logscale. Note; since equispaced tic marks in the original scale are not equispaced in the log scale, the TICINC field of the returned TICINFO would be a list of the unequispaced tic marks values, rather than a number.

The plot scale of each axis may be manipulated directly through the following functions.

(PLOTAXISINTERVAL *plot axis newinterval nodrawflg*) [Function]

Plot and axis are required. Axis must be one of X, or Y. If newinterval is NIL, returns the current AXISINTERVAL for that axis. If newinterval is non-NIL it must be an AXISINTERVAL.

(PLOTTICINFO *plot axis newticinfo nodrawflg*) [Function]

Plot and axis are required. Axis must be one of X, or Y. If newticinfo is NIL , returns the current TICINFO for that axis. If newticinfo is non-NIL it must be a TICINFO.

On occasion it is useful to clean out an existing plot instead of creating a new one.

(PLOT.RESET *plot xscale yscale flushmargins flushprops nodrawflg*) [Function]

Returns plot to a pristine state. If xscale and yscale are provided, the scale of the plot is set accordingly.

Finer control over the behavior of plot objects is possible through the following functions.

(TRANSLATEPLOT OBJECT *plotobject dx dy plot nodrawflg*) [Function]

Moves plotobject dx, dy in world coordinates and updates the associated window accordingly. The update is suppressed if nodrawflg is non NIL.

(DRAWPLOT OBJECT *plotobject plot*) [Function]

Draw plotobject in the window associated with plot. As with all the display functions, the window should be opened beforehand. DRAWOBJECT does NOT check that the window is open.

APPLY's the plotobject's DRAWFN.

(ERASEPLOT OBJECT *plotobject plot*) [Function]

APPLY's the plotobject's ERASEFN

(HIGHLIGHTPLOT OBJECT *plotobject plot*) [Function]

Invoked when a plotobject is selected

(LOWLIGHTPLOT OBJECT *plotobject plot*) [Function]

Invoked when a plotobject is deselected

(EXTENTOFPLOT OBJECT *plotobject plot*) [Function]

Computes the extent of plotobject in world coordinates.

Returns an EXTENT, which has fields MAXX, MINX, etc.

(DISTANCETO PLOT OBJECT *plotobject streamposition plot*) [Function]

Returns the "distance" to plotobject from streamposition in stream coordinates. Value returned may be a FIXP or a FLOATP, but is always a distance in stream coordinates.

(CLOSESTPLOT OBJECT *plot streamposition*) [Function]

Returns the "closest" plotobject on plot's display list to streamposition.

(DESELECTPLOT OBJECT *plot*) [Function]

Deselects the current selected object of plot

Plot objects also have "afterfns". That is, functions which are optionally invoked after some standard operation. These are stored as plot object properties with distinguished names, and invoked with at least two args, the plotobject and the plot.



WHENADDEDNFN [Property]

The WHENADDEDNFN is called with three arguments, `plotobject`, `plot`, and `nodrawflg`

WHENDELETEDNFN [Property]

The WHENDELETEDNFN is called with four arguments, `plotobject`, `plot`, `nodrawflg`, and `nosaveflg`.

WHENDRAWNFN [Property]

The WHENDRAWNFN is called with three arguments, `plotobject`, `viewport` and `plot`.

WHENERASEDNFN [Property]

WHENHIGHLIGHTEDNFN [Property]

WHENLOWLIGHTEDNFN [Property]

WHENTRANSLATEDNFN [Property]

A PLOT has two associated windows, the mainwindow in which the graphics, labels, tics, etc. are displayed and an attached promptwindow. The mainwindow is cached as `plot` property and may be accessed via the function `PLOTPROP`. A function is provided for easy access to the prompt window.

(`PLOTPROMPT text plot`) [Function]

Text is output in the one character high prompt window of `plot`.

PLOT's may be drawn in ANY imagestream (but only interacted with in the PLOT's associated window). The following function is the fundamental draw primitive.

(`DRAWPLOT plot stream streamviewport streamregion`) [Function]

Stream is any imagestream. Streamviewport is a viewport on that stream that defines the the world to stream transformation. Streamregion is a region in stream coordinates that will contain the entire image (for a window it will be the `CLIPPINGREGION`). Streamviewport is usually the result of `ADJUSTVIEWPORT`.

For more information about viewport, consult the documentation for the **TWODGRAPHICS** module.

(`ADJUSTVIEWPORT viewport streamregion plot`) [Function]

Viewport is a `VIEWPORT` whose `parentstream` is the imagestream of interest. Streamregion is a region in stream coordinates that will contain the entire image.

Adjusts the `Streamsubregion` and `Worldregion` of viewport to reflect the current scale and margin setting of `plot`.

(`MINSTREAMREGIONSIZE stream plot`) [Function]

Returns a `CONS` pair (`minwidth` . `minheight`) of the plot in stream coordinates.

A plot has "afterfns" for two major operations, opening and closing the plotwindow. These are stored as plot properties with distinguished names. The values of these properties may be a single function or a list of functions which are called in sequence with the plot as an argument.

WHENOPENEDNFN [Property]

WHENCLOSEDNFN [Property]

PLOT's may be copied, made into image objects, dumped onto files, sent in the mail, etc.

(COPYPLOT *plot*)

[Function]

Returns a copy of plot. The user defined properties require special handling. If there exists a plot prop COPYFN, which may be function or list of functions, the function (or functions) will be invoked with the arguments newplot plot and propname for each user defined property on plot. If the function returns a non-NIL value, it will be used as the value of propname on newplot. In the case of a list of functions, the first non-NIL value (traveling from the head to the tail of the list of functions) will be used as the new prop value. Otherwise the prop will be HCOPYALL'ed.

(COPYPLOT OBJECT *plotobject plot*)

[Function]

Returns a copy of plotobject. The protocol for copying objectprops is similar to plot props. The plotobject may have a COPYFN prop which may be a function or list of functions. The function (or functions) will be invoked with the arguments newplotobject plotobject plot propname. The first non-NIL value will be used as the prop value else the property will be HCOPYALL'ed.

(PRINTPLOT *plot stream*)

[Function]

Writes out an HREADable symbolic representation of plot on stream. Again, user defined properties require special handling. If there exists a plot prop PUTFN, which may be function or list of functions, the function (or functions) will be invoked with the arguments plot propname and stream for each user defined property on plot. If the function returns a non-NIL value, it is assumed an HREADable representation of the prop value has been written out on stream. In the case of a list of functions, the functions will be invoked one at a time, starting from the head of the list, until a non-NIL result is obtained. If there is no PUTFN, or the function (or none of the functions) returns a non-NIL value, the prop is HPRINT'ed.

Lists of the form ((FUNCTION function) arg) are recognized by the inverse of PRINTPLOT, READPLOT, to imply that function should be called with plot and arg as arguments at HREAD time, and the value returned to be the prop value.

(PRINTPLOT OBJECT *plotobject plot stream*)

[Function]

Writes out an HREADable symbolic representation of plotobject on stream. As in PRINTPLOT user defined object properties require special handling. The protocol is the same as in PRINTPLOT.

The following data types have HPRINT macros and need no special handling: FONTDESCRIPTOR, MENU, PLOT, and PLOT OBJECT.

A file package command has been defined to simplify dumping PLOT's on files.

(PLOTS . *plots*)

[FilePkgCom]

The syntax is identical to VARS.

A plot image object is fully supported.

(CREATEPLOTIMAGEOBJ *plot*)

[Function]

Returns an image object which contains a copy of plot. These image objects can also be created by copy-selecting from a plot window into a host window (e.g. TEdit or Sketch) that supports image objects. Such a selection will ask whether the plot should be inserted as a bitmap or a plot, the latter case constructing a plot image object. Buttoning on the image object provides the option of reshaping the plot or creating a separate plot window in which the plot can be modified. Closing the plot window will ask whether the new plot should be reinserted in the host.

---

---

**PLOTBOXPATCH**

---

---

By: Tad Hogg (Hogg.PA@Xerox.COM)

Uses: PLOT

This document last edited on 26-Jan-89.

**INTRODUCTION**

This file redefines the plot functions BOXREGION and DRAWPLOT so that each side of the box around the plot need not be drawn. Specifically, giving a plot a non-NIL value for the properties NOLEFT, NORIGHT, NOTOP or NOBOTTOM will eliminate the respective side from the drawing of the plot.

---

---

## PLOT EXAMPLES

---

---

By: Jan Pedersen (Pedersen.PA @ Xerox.com)

Uses: PLOT

This module contains two examples of how PLOT might be used to produce high level plotting facilities. The first example is a histogram primitive, and the second is a scatterplotter. The code is commented, and exploits most of the facilities in PLOT. The scatterplot example is the simpler of the two, and is suggested as a starting point.

(SCATPLOT *y x pointlabels ylabel xlabel title symbol*) [Function]

Generates of a scatterplot of y vs x which are numeric lists of equal length. If x is NIL, then y is plotted vs the integers from 1 to (LENGTH y). Pointlabels is a list of labels, one for each point plotted. Ylabel and xlabel are labels for the x and y axis respectively. Title is a title for the scatterplot. Symbol is the plotting symbol to use, must be a BITMAP; defaults to STAR.

Returns a PLOT.

(HISTPLOT *batch label shade*) [Function]

Batch is a list of numbers, or a list of pairs (number . frequency) whose histogram will be displayed. Label is an optional label for those numbers. Shade is a shade to use to fill the bars of the histogram (defaults to SHADE3). The case of all entries in batch being integers is treated specially.

Returns a PLOT.

---



---

## PLOT OBJECTS

---



---

By: Jan Pedersen (pedersen.PA @ Xerox.com)

Uses: PLOT and TWODGRAPHICS

Plot objects are the primitive quantities of the **PLOT** module. A plot object is abstracted as an instance of datatype PLOBJECT. A point plot object is an instance of PLOBJECT whose data component describes a point. That is, a point plot object is a subtype of PLOBJECT; all plot objects satisfy (type? PLOBJECT FOO), but only a point plot object satisfies in addition (PLOT OBJECTSUBTYPE? POINT FOO).

A PLOBJECT is both a datatype and a collection of functions that implements a set of generic operations on that plot object. A plot object must know how to draw itself, erase itself, highlight itself, etc. The PLOT module then deals only with generic operations, and allows the plot objects to implement them as is appropriate.

|                                                                                                                                |            |
|--------------------------------------------------------------------------------------------------------------------------------|------------|
| PLOBJECT                                                                                                                       | [Datatype] |
| OBJECTFNS                                                                                                                      | [Field]    |
| Must be an instance of PLOTFNS                                                                                                 |            |
| OBJECTSUBTYPE                                                                                                                  | [Field]    |
| Describes the plot objects subtype                                                                                             |            |
| OBJECTUSERDATA                                                                                                                 | [Field]    |
| Space for a property list                                                                                                      |            |
| OBJECTMENU                                                                                                                     | [Field]    |
| The object's MENU                                                                                                              |            |
| OBJECTLABEL                                                                                                                    | [Field]    |
| Something to print                                                                                                             |            |
| OBJECTDATA                                                                                                                     | [Field]    |
| Space for a datatype that describes the subtype of this PLOBJECT                                                               |            |
| The field OBJECTFNS must be an instance of PLOTFNS, essentially a vector of functions which implements the generic operations. |            |
| PLOTFNS                                                                                                                        | [Datatype] |
| DRAWFN                                                                                                                         | [Field]    |
| Implements the DRAWOBJECT generic operation                                                                                    |            |
| ERASEFN                                                                                                                        | [Field]    |
| etc.                                                                                                                           |            |

|             |         |
|-------------|---------|
| HIGHLIGHTFN | [Field] |
| LOWLIGHTFN  | [Field] |
| LABELFN     | [Field] |
| MOVEFN      | [Field] |
| EXTENTFN    | [Field] |
| DISTANCEFN  | [Field] |
| COPYFN      | [Field] |
| PUTFN       | [Field] |
| GETFN       | [Field] |

The generic operations are:

(DRAWPLOT OBJECT *object viewport plot*) [Function]

Draw the object within viewport. A VIEWPORT may be thought of as a sub imagestream. It will usually be associated with the plot's PLOTWINDOW, but might also be associated with some other image stream. Typically this generic operation will make use of functions from TWODGRAPHICS and the position of the object in world coordinates. The plot is also passed as an argument, so that the draw operation may make use of information cached on the property list of plot.

The only operation that is expected to draw on streams other than the PLOTWINDOW is drawobject, so the drawfn may have to behave differently depending on the imagestreamtype of the stream. All other generic operations are assumed to operate on the PLOTWINDOW. The idea here is that plot's may be drawn on any stream, but may be interacted with only through the PLOTWINDOW. It is also guaranteed that an object will be drawn before it is erased, highlighted, etc.

(ERASEPLOT OBJECT *object viewport plot*) [Function]

Erase the object from the viewport. The inverse of DRAWOBJECT. It is guaranteed that the viewport will be on the PLOTWINDOW

(HIGHLIGHTPLOT OBJECT *object plot*) [Function]

Highlight the object. Used in selection.

(LOWLIGHTPLOT OBJECT *object plot*) [Function]

The inverse of HIGHLIGHTOBJECT. With XOR drawing the HIGHLIGHTFN and the LOWLIGHTFN can often be the same.

(MOVEPLOT OBJECT *object dx dy plot*) [Function]

Destructively alter the object's OBJECTDATA, so that its position is moved dx, dy units (in world coordinates).

(LABELPLOT OBJECT *object plot*) [Function]

If it is desired to label the object, the LABELFN will be called. Often the function LABELGENERIC will do the trick.

(EXTENTOFPLOT OBJECT *object plot*) [Function]

Should return an EXTENT, which expresses the range of the object in world coordinates.

|                                                |            |
|------------------------------------------------|------------|
| EXTENT                                         | [Datatype] |
| MINX                                           | [Field]    |
| Minimun extent in the X (horizontal) direction |            |
| MAXX                                           | [Field]    |
| Maximun extent in the X (horizontal) direction |            |
| MINY                                           | [Field]    |
| Minimun extent in the Y (vertical) direction   |            |
| MAXY                                           | [Field]    |
| Maximun extent in the Y (vertical) direction   |            |
| All fields are type floating.                  |            |

(DISTANCETO PLOT OBJECT *object streamposition plot*) [Function]

Should return a number (more efficient if it returns a SMALLP), which is some measure of the distance from the REPRESENTATION of the object to the POSITION streamposition. Note that distance is calculated in stream coordinates, NOT world coordinates. This is done for efficiency and logical consistency. Selection makes most sense as an activity in stream coordinates.

A plot object will typically cache its stream coordinates when it is drawn. Although not strictly necessary (it is always possible to backsolve to stream coordinates from world coordinates), this improves efficiency many fold by avoiding generation of floating point boxes.

The following functions are provided to allow the plot object to customize how it is copied, printed on file, etc. The generic defaults will usually be satisfactory.

(COPY PLOT OBJECT *object plot*) [Function]

Returns a copy of object. COPY OBJECT will create a new instance of PLOT OBJECT and copy over all the fields of object except for OBJECT DATUM. The object's COPY FN is evoked with the arguments object and plot and is expected to return a new instance of OBJECT DATUM. The objects property list is handled as follows: If object has a prop COPY FN (which may be a function or list of functions), for each property it is called with the arguments newobject, oldobject, plot, propname. If the returned value is non-nil it is used as the value for that property on newobject; else the prop value is H COPY ALL'ed. If the value of COPY FN is a list of functions, they are invoked in order head to tail, and the first non-NIL value is used as the new value.

(PRINT PLOT OBJECT *object plot stream*) [Function]

Writes out to stream an H READable symbolic representation of object. As in COPY OBJECT, PRINT OBJECT takes care of all PLOT OBJECT fields except of OBJECT DATUM. The objects PUT FN will be invoked with the arguments object plot stream and is expected to write out a representation of OBJECT DATUM which is H READable. This will usually be in prop list format.

Again the prop list of object requires special handling. The special object prop PUT FN may be a function or list of functions. For each property it will be invoked with the arguments object plot propname and stream and if it returns a non-NIL value, it is assumed that property has been written out in a H READable format. Again, if the the PUT FN prop is a list of fns then if any one of them returns non-NIL then the property is assumed written out. If there is no PUT FN then the property is (H PRINT prop stream NIL T) 'ed.

PUTFNS may put out special lists of the form ((FUNCTION fnname) arg) in which case fnname will be invoked at HREAD time with args object plot propname arg and fnname will be expected to return the propvalue of propname.

(READPLOT OBJECT *stream*) [Function]

Reads in the product of PRINT OBJECT. Calls the objects GETFN to read in the OBJECTDATA field.

An instance of PLOT FNS may be created by the function:

(CREATEPLOT FNS *drawfn erasefn extentfn distancefn highlightfn  
lowlightfn labelfn movefn copyfn putfn getfn borrowfrom*) [Function]

Returns an instance of PLOT FNS. Drawfn, erasefn, and extentfn are required. If a distancefn is supplied then so must be a highlightfn. Lowlightfn defaults to highlightfn, labelfn defaults to LABELGENERIC. The other arguments also default to some safe, if not too efficient genericfn.

A primitive inheritance scheme is implemented via the optional argument borrowfrom. If supplied, borrowfrom must be an instance of PLOT FNS. Before creating the new instance of PLOT FNS, the NIL arguments passed are filled in from the fields of borrowfrom, with the following exception; lowlightfn is only inherited if highlightfn is also NIL.

The OBJECTDATA field will typically be a datatype which holds the data characterizing the PLOT OBJECT. For example a point plot object will have an OBJECTDATA field whose value is an instance of the datatype POINTDATA (has fields position, symbol, etc). So, a point PLOT OBJECT is a specialization of PLOT OBJECT. The field OBJECTSUBTYPE is supplied to make the subtype explicit. The following macro is provided to facilitate testing for plot object subtypes.

(PLOT OBJECTSUBTYPE? *subtype plotobject*) [Macro]

Essentially tests if (EQ subtype (fetch OBJECTSUBTYPE of plotobject))

(PLOT OBJECTSUBTYPE *plotobject*) [Function]

Returns the value of the OBJECTSUBTYPE field.

PLOT OBJECTS may be created via the function:

(CREATEPLOT OBJECT *objectfns objectlabel objectmenu objectdata*) [Function]

Returns an instance of PLOT OBJECT. Coerces objectmenu into a MENU if it is an item list.

The following subtypes of PLOT OBJECT are currently implemented.

pointPLOT OBJECT, curvePLOT OBJECT, polygonPLOT OBJECT, linePLOT OBJECT,  
graphPLOT OBJECT, texttPLOT OBJECT, filledrectanglePLOT OBJECT, compound PLOT OBJECT

The functions CREATEPOINT, etc. return an instance of PLOT OBJECT, with the appropriate OBJECTFNS and OBJECTDATA. In order for this to work, some initializations must be done at load time.

The function PLOT.SETUP performs the initializations at LOAD time.

(PLOT.SETUP *opstable*) [Function]

Opstable must be a list of lists of the form:

(



```
(subtype1 (opname1 function1) (opname2 function2)
(subtype2 (opname1 function1) (opname2 function2)
.....
(subtypenamen (opname1 function1)(opname2 function2)
)
```

Creates one instance of PLOTFNS for each subtype.

In summary, to add a new plot object you need to:

- Determine the data required to describe the new subtype. This may involve declaring a new datatype.
- Write functions similar to CREATEPOINT and PLOTPOINT for the new subtype.
- Write (or borrow) the functions which implement the generic ops described above.
- Invoke MAKEPLOTFNS to create an instance of PLOTFNS for the new plot object subtype, which all objects of that subtype will refer to.
- If continued use of the new plot object is contemplated, PLOT.SETUP should be evoked at load time to effect the proper initializations.

Look at the code for existing plot objects for more details. The point plot object is the simplest example.

---



---

## PLOTOBJECTS1

---



---

By: Tad Hogg (hogg.PA @ Xerox.com)

Uses: PLOT and PLOTOBJECTS

PLOTOBJECTS1 defines additional plot objects for use with PLOT.

### NEW PLOTOBJECTS

ERRORPOINT - a point with vertical and/or horizontal error bars.

SAMPLESET - a set of points drawn as line segments to a specified vertical or horizontal line.

### FUNCTIONS

The following functions provide an add facility for the new objects. They are similar to the corresponding functions for the standard plot objects, e.g. PLOTPOINT, etc. The allowed forms of the arguments *symbol*, *style*, *menu* and *nodrawflg* are the same as for the standard functions.

(PLOTERRORPOINT *plot position-range label symbol style menu nodrawflg*) [Function]

Position-range is a list of the form (POSITION XRANGE YRANGE). POSITION is the position of the point in world coordinates. XRANGE and YRANGE control the length of the horizontal and vertical error bars respectively. If the range is NIL, no error bars are drawn. If it is a number, it is a distance (in world coords) for the error bar to extend on each side of the point. Finally, if it is a pair of numbers (NegDist . PosDist) it specifies the extent of the error bar in the negative and positive directions, respectively. Symbol is used to plot the point. Symbol defaults to STAR. Style specifies the style to use for drawing the error bars.

Returns an ERRORPOINT PLOT OBJECT.

(PLOTERRORPOINTS *plot position-ranges labels symbol style menu nodrawflg*) [Function]

As above except that position-ranges is a list of POSITION-RANGES as described above and labels may also be a list. Reasonable things happen if positions and labels are of unequal length.

Returns a list of ERRORPOINT PLOT OBJECT's.

(PLOTSAMPLESET *plot positions constant vertical? side label style menu nodrawflg*) [Function]

The list of POSITION's defines a number of sample points. Constant specifies the location of a vertical or horizontal line, depending on whether vertical? is non-NIL. Line segments are drawn from the sample points to this line. Side determines which points are actually included. If side is NIL, only those points whose coord is greater than constant will be drawn (i.e. points above or to the right of the line). If side is T, only those with coord less than constant will be drawn. Otherwise, all points will be included. Style specifies the style to use for drawing the line segments.

Returns a SAMPLESET PLOT OBJECT.

All plot objects may be created independently of the previous functions. This is useful if it is desired to create a plot object without entering it on a PLOT's display list. The following functions create and return the new plot objects.

(CREATEERRORPOINT *position-range label symbol style menu*) [Function]

Returns an ERRORPOINT PLOT OBJECT.

(CREATESAMPLESET *positions constant vertical? side label style menu*) [Function]

Returns a SAMPLESET PLOT OBJECT.

In addition there are a number of functions to aid in creating position-ranges used with the error point plot objects:

(MAKE-POSITION-RANGE *position xrange yrange*) [Function]

Returns a position-range suitable for use for specifying an error point. The arguments are as described above for PLOTERRORPOINT.

(LOG-ERROR-RANGE *position-range axis base*) [Function]

Returns a position-range corresponding to *position-range* converted to a log scale. *base* is the log base to use (defaults to 10) and *axis* specifies which axis to convert: :X or :Y for a specific axis, NIL for both. Note that the position, with its error bars must be positive in order to be converted to a log scale.

(LOG-ERROR-RANGE-LIST *position-ranges axis base*) [Function]

Converts a list of *position-ranges* to log scale.

---



---

## PLOTOBJECTS2

---



---

By: Tad Hogg (hogg.PA @ Xerox.com)

Uses: PLOT and PLOTOBJECTS

PLOTOBJECTS2 defines an additional plot object for use with PLOT.

### NEW PLOT OBJECT

FILLEDPOLYGON - a polygon with optional shading in its interior.

### FUNCTIONS

The following functions provide an add facility for the new object. They are similar to the corresponding functions for the standard plot objects, e.g. PLOTPOINT, etc. The allowed forms of the arguments *texture*, *style*, *menu* and *nodrawflg* are the same as for the standard functions.

(PLOTFILLEDPOLYGON *plot positions label style texture menu nodrawflg*) [Function]

The points in *positions* define a closed polygon. The polygon is filled with *texture*, and the other arguments are the same as for PLOTPOLYGON. If the linewidth specified by *style* is 0, the polygon perimeter will not be drawn.

Returns a FILLEDPOLYGON PLOT OBJECT.

All plot objects may be created independently of the previous functions. This is useful if it is desired to create a plot object without entering it on a PLOT's display list. The following creates and returns the new plot object.

(CREATEFILLEDPOLYGON *positions label style texture menu*) [Function]

Returns aa FILLEDPOLYGON PLOT OBJECT.

The actual drawing is done with

(CLIPPED.FILLPOLYGON *clippingregion points texture stream operation windnumber draw? width drawoperation color dashing*) [Function]

which effectively does both of (FILLPOLYGON *points texture stream operation windnumber*) and (if *draw?* is non-NIL) a series of (DRAWLINE *x1 y1 x2 y2 width drawoperation stream color dashing*) for each edge of the polygon except they are clipped against *clippingregion* in stream coordinates.

---

---

**PREEMPTIVE**

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

This module turns on pre-emptive process scheduling. Using IL:\PERIODIC.INTERRUPT, it forces a block in whatever process is running.

(IL:PREEMPTIVE &OPTIONAL STATE)

[Function]

The function PREEMPTIVE turns preemptive process scheduling on and off. (IL:PREEMPTIVE 'ON) turns it on, (IL:PREEMPTIVE 'OFF) turns it off. (IL:PREEMPTIVE) with no argument returns the current state with no change.

**WARNING WARNING WARNING WARNING DANGER DANGER DANGER DANGER**

PREEMPTIVE is dangerous. Many places in the system do not have monitor locks and other mechanisms to prevent one process from overwriting the data of another in the face of preemptive interrupts. (Most do, of course.)

I've run with preemptive scheduling turned on for weeks, and about once a day, my screen gets trashed, windows and menus overwritten, etc. This version of PREEMPTIVE is a little more conservative than previous versions, e.g., it checks to see if the system is running in the MENU code and doesn't do a process switch. However:

**USE AT YOUR OWN RISK. CAUTION CAUTION.**

**NOTE: Using SPY turns preemptive scheduling OFF.**

---



---

## PRESSFROMNS

---



---

By: Tad Hogg (Hogg.pa@Xerox.com)

### INTRODUCTION

This module is a patch to allow Press printers to print NS characters by translating them to appropriate Press fonts. Before loading this file, make sure there are no open press streams (i.e. no hardcopy in progress to a Press printer).

### CONTROLLING CHARACTER SET TRANSLATIONS

The translations are controlled by a number of variables and functions described below. These variables can be modified to provide additional or different translations.

#### Global character set translations

NSTOASCIITRANSLATIONS [Variable]

an ASSOC list whose elements have the form (charset translationArrayName). This specifies which translation array is to be used when translating the specified character set.

Example: ((0 ASCIIFROM0ARRAY) (38 ASCIIFROM38ARRAY)) specifies that ASCIIFROM38ARRAY is bound to the array to be used for translating charset 38.

The translationArrayName is bound to an array whose index ranges from 0 to 255. Each element of the array specifies the translation to use for the corresponding charcode in this charset.

Translations are of one of the following forms:

1. NIL -- no translation specified which will use the font as is for charset 0 and otherwise print a black box to indicate the NS character could not be translated
2. an integer in the range 0 to 255 which indicates that this value is to be used as the translated charcode, but that no font translation is required [This is mainly useful for converting NS to ASCII in charset 0.]
3. a two element translation list of the form (fontFamily charcode) which indicates that this character should be translated to charcode in a font whose family is fontFamily. Note that charcode should be in the range 0 to 255. fontFamily can also be a font descriptor or a font specification list (e.g. (Gacha 10)) of a form acceptable to FONTCREATE.

PRESSFONTFAMILIES [Variable]

a list whose elements are of the form (FAMILY . specialTranslations). The optional list specialTranslations specifies translations to use for Press fonts in charset 0. Each element is (charcode translation) which specifies that translation is to be used for charcode in this family.

Example: ((GACHA (50 (HELVETICA 40)) (51 (TIMESROMAN 45))) (TIMESROMAN (SYMBOL))

Note: these translations are cached in the font descriptor when PRESS fonts are created so any changes to these variables will not change the translations in previously created fonts.

### Translations in individual fonts

The following two functions provided detailed control over the translations used in individual fonts.

(GETCHARPRESSTRANSlation *CHARCODE FONT*) [Function]

returns the translation (a two element list) used for CHARCODE in FONT

(PUTCHARPRESSTRANSlation *CHARCODE FONT NEWTRANSLATION*) [Function]

sets the translation to be used for CHARCODE in FONT. NEWTRANSLATION should be a translation in one of the forms described above.

### Additional functions

(PRESS.NSARRAY *CHARSET FAMILY ASCIIARRAY*) [Function]

This function returns a suitable translation array built as the inverse of a translation array from Press to NS characters. Such arrays are used to print Press fonts on Interpress printers and are listed in the variable ASCIIIONSTRANSATIONS. CHARSET is the character set for which to create a translation. FAMILY is the press font family for which ASCIIARRAY is the translation to NS characters. If ASCIIARRAY is not specified, the function looks through all arrays included on ASCIIIONSTRANSATIONS to fill in the translation array. The following two functions provided detailed control over the translations used in individual fonts.

### CONTROLLING PRESS FONT COERCIONS

There is also a mechanism for determining which press font is actually used for the translation. For example, an NS character in the Modern 8 font might translate to an ASCII character in Symbol 8. If this font does not exist on the printer, a (generally incorrect) font change will be done by the printer. The following procedure changes the actual font used, e.g. to Symbol 10 in this example.

The coercion is controlled by a coercion list which is an alist indexed by device. The font coercion scans down the element on the list for the requested device (e.g. PRESS) looking for the first entry that matches the user request. If a match is found, then the entry tells how to construct an appropriate new name from the requested specification. Fields of the newname not specified in the entry are simply copied over.

FONTCOERCIONS [Variable]

This list allows the user to coerce fonts that he knows don't exist on the printer even tho the fonts-widths files doesn't indicate that (e.g. the desired size doesn't exist). FONTCOERCIONS is initialized simply to take all SYMBOL fonts of size less than 10 into 10, and size greater than 12 into 12.

MISSINGFONTCOERCIONS [Variable]

If the initial coerced (or uncoerced) lookup fails, then MISSINGFONTCOERCIONS is used. This takes MODERN into HELVETICA etc--the standard press coercions.

The procedure for determining whether a user request matches a coercion entry is straightforward. If the match-part of the coercion entry is an atomic family name, it matches if it is eq to the requested family. Otherwise, the match-part must be a list of family, size, face in standard fontname order. If a component is NIL or missing, then it is assumed to match. The only funniness is in size matching. The size component can be NIL (matches anything), a particular size (EQ matches), or a list of the form (<

n) or (> n) where n is a size number. The first matches any requested size less than n, and the second matches any requested size greater than n.

FONTCOERCIONS for PRESS starts out as

```
((SYMBOL (< 10)) (SYMBOL 10)) (SYMBOL (> 12)) (SYMBOL 12))
```

MISSINGFONTCOERCIONS for PRESS starts out as

```
((MODERN HELVETICA) (CLASSIC TIMESROMAN) etc.)
```



---

---

## PRETTYFILEINDEX

---

---

By: Bill van Melle (vanMelle.PA@Xerox.com)

### INTRODUCTION

Prettyfileindex is a program for generating indexed listings for Lisp source files. Prettyfileindex operates by reading expressions from the file and reprettyprinting them to the output image stream, building up an index of the objects as it goes. The index is partitioned by type (e.g. FUNCTIONS, VARIABLES, MACROS, etc.); within each type, the objects are listed alphabetically by name along with the page number(s) on which their definitions appear in the listing.

Prettyfileindex also modifies the Exec's and the FileBrowser's SEE command to prettyprint the file being viewed, if it is a Lisp source file. It also modifies the PF and PF\* commands to prettyprint the requested function body. Together, these features mean you can use the NEW & FAST options to MAKEFILE to speed up file creation without sacrificing the ability to get pretty listings or see the files prettily inside Lisp.

Prettyfileindex performs some additional niceties in the listing: it prints bitmaps by "displaying" them, rather than dumping their bits; it translates underscore to left arrow (for the benefit of Interlisp listings); it prints quote and backquote in a font in which they are clearly distinguishable; and it suppresses some of the "noise" in source files, such as the filemap.

The module also contains a function MULTIFILEINDEX that can be used to generate a merged index of items from a whole set of files being listed.

Prettyfileindex subsumes, and is incompatible with, the modules SINGLEFILEINDEX and PP-CODE-FILE. You can, however, load Prettyfileindex on top of either one, and it will successfully wrest control of LISTFILES from them. Prettyfileindex has several advantages over SINGLEFILEINDEX: the prettyprinter has fine control over positioning of the output stream, so things that are supposed to line up do, despite font changes and variable-width fonts; the entire page is used, rather than sacrificing the bottom quarter or so due to lack of control over page breaks; and the use of an image stream allows bitmaps to be rendered directly.

### USING PRETTYFILEINDEX

For ordinary use, just load PRETTYFILEINDEX.LCOM. This redefines LISTFILES1 so that calling LISTFILES or using the File Browser's Hardcopy command invokes Prettyfileindex if the file is a Lisp source file. The listing is created by default in a single background process that handles all LISTFILES requests. The file being indexed needn't be loaded, or even noticed (in the File Manager sense) as long as the file's commands don't require customized prettyprinting defined by the file itself. The index is printed at the end of the listing; you are expected to manually transpose the index to the front of the collection of paper that emerges from the printer.

Prettyfileindex normally assumes that you are printing one-sided listings. However, if your global default is for two-sided (currently this means that EMPRESS#SIDES = 2) or you specified two-sided in the options you passed to LISTFILES, it will prepare the output as if for two-sided listing. For example, from an Interlisp exec,

```
(LISTFILES (SERVER "Perfactor:" %#SIDES 2) FOOBAR)
```

causes the file FOOBAR to be listed two-sided on the print server Perfactor: (the % is the Interlisp reader's escape character, needed to quote the special character #; in an XCL exec the escape character is \, and from other packages you also have to qualify the symbols LISTFILES, SERVER and #SIDES with the package prefix IL:).

For two-sided listings, the margins are symmetric, instead of being shifted a bit to the right, page numbers appear on the outside edge of the page, and a blank page is inserted at the end of the listing if necessary to ensure that the index starts on an odd page (and hence is transposable to the front).

Prettyfileindex prettyprints the file's contents and prints indexed names using the package and read table specified in the file's reader environment, which appears at the beginning of the file. It assumes, as does most of the file manager, that the reader environment is sufficient to read any expression on the file. If you have violated this assumption, for example, by referring in the file to a symbol in another package that is defined on a file that is indirectly loaded by the file somewhere in its coms, you will probably need to LOADFROM the file before you can list it.

## INDEXING MULTIPLE FILES

Ordinarily, you list files and get one index per file. If a module is made up of several files, you may want a master index of the whole set of files, so that you don't have to remember which file contains a function, macro, etc. that you are looking up. This job is handled by MULTIFILEINDEX:

```
(MULTIFILEINDEX files printoptions) [Function]
```

This function lists each of the files in the list *files* using PRETTYFILEINDEX and then produces a master index by merging all the individual indices. The master index is appended to the output of the last file listed. The argument *files* can be a list of file names and/or file patterns, such as "{FS:}<Carstairs>RED\*", or a single such pattern. In the pattern, unless explicitly specified, the extension defaults to null and the version to "highest". The argument *printoptions* is a property-list of options, the same as the *printoptions* argument to SEND.FILE.TO.PRINTER or PRETTYFILEINDEX, with the addition of some options recognized by MULTIFILEINDEX, described further below.

As each file is listed, its pages are numbered with an ordinal file number plus the page number within the file; e.g., in the first file the pages are numbered 1-1, 1-2, ..., in the second file 2-1, 2-2, etc. The master index then refers to page numbers in this form, although each individual file's own index shows only the file-relative page numbers. Alternatively, you can tell MULTIFILEINDEX to number all the pages consecutively, rather than using "part numbers", by giving the option :CONSECUTIVE, value T in *printoptions*.

In the event that some files in the set have different reader environments, the master index is printed in the environment used by the majority of the files. More specifically, MULTIFILEINDEX independently chooses the package used by the majority of the files and the readtable used by the majority; in the case of a tie, the file later in the set wins. If this default is not adequate, you can specify the environment yourself by giving the :ENVIRONMENT option. The value should either be a reader environment object, such as produced by MAKE-READER-ENVIRONMENT, or a property list of the form used by the MAKEFILE-ENVIRONMENT property.

For example,

```
(MULTIFILEINDEX "<Barney>Rub*"
 ' (:CONSECUTIVE T
 :ENVIRONMENT (:PACKAGE "JABBA" :READTABLE "XCL")))
```

would list each of the files matching "<Barney>Rub\*;", numbering the pages consecutively from the first file through the last, and printing the master index with respect to the package JABBA and read table XCL.

### INCREMENTALLY REPRINTING MULTIPLE FILES

If you have used MULTIFILEINDEX to list a group of files, and later one of the files changes, or maybe the printer just ate part of your listing, you might want to update your listing without reprinting the entire set of files. You have two options.

(1) You can have PRETTYFILEINDEX reprint the one file that changed (or was eaten). Specify the print option `:PART n` to have it treat the single file as the *n*th part of a multiple listing, or the option `:FIRSTPAGE n` to have it start numbering the pages at *n* instead of 1 (for the case where you used the `:CONSECUTIVE` option to MULTIFILEINDEX). For example,

```
(LISTFILES (:PART 3) "<Barney>Rubric")
```

would reprint <Barney>Rubric as the third file in a group. Of course, this doesn't reprint the master index, but it only has to process the one file, which may be adequate for your needs if things didn't move around too much.

(2) You can have MULTIFILEINDEX process the entire set of files again, but only print some of them. You specify this by parenthesizing the files you don't want printed. That is, each element of the *files* argument to MULTIFILEINDEX is a file name or a list of file name(s); those files inside sublists are processed but not printed. You cannot specify patterns. The master index is listed after the last file, as usual, except that if the last file was in a sublist, and hence not printed, the master index will appear as a separate listing. Calling MULTIFILEINDEX in this manner is nearly as computationally expensive as calling it to list the whole set for real (it omits only the transportation to the printer), but it does save paper and printer time.

### LISTING COMMON LISP FILES

Ordinarily, PRETTYFILEINDEX only processes files produced by the Lisp File Manager; it passes all others off to the default hardcopy routines. However, you can tell it to process a plain Common Lisp text file by passing the print option `:COMMON`; e.g.,

```
(LISTFILES (:COMMON T) "conjugate.lisp")
```

Prettyfileindex still processes the file by reading and prettyprinting, just as for Lisp files. It starts in the default Common Lisp reading environment (package USER and read table LISP), and evaluates top-level package expressions, such as `in-package` and `import`, in order to continue reading correctly. The index is printed in whatever the environment was at the end of the file.

Of course, this is of fairly limited utility, as all read-time conditional syntax is lost: comments, `#+`, `#o`, etc. The one exception is that top-level semi-colon comments are preserved—they are copied to the output directly, rather than being read.

## Customizing PRETTYFILEINDEX

The remainder of this document describes various ways in which PRETTYFILEINDEX can be customized.

## HOW TO SPECIFY INDEXING TYPES

Initially, PRETTYFILEINDEX knows about most of the standard file manager types. In addition, it handles all the types defined by DEFDEFINER. For definers with a :NAME option, it assumes that the function is free of side effects. PRETTYFILEINDEX also notices (but does not evaluate) DEFDEFINERs that appear on the file it is currently indexing, which should appear before any instances of the type so defined in order for correct indexing to occur. Of course, it can't know about definer types that are defined on some other file unless you load it.

You can augment the set of indexing types, or override the default handling of definers, by adding elements to the following variable:

\*PFI-TYPES\* [Variable]

A list of entries describing types to be indexed and a way of testing whether an expression on the file is of the desired type. Each entry is a list of up to 4 elements of the form (*type dumpfn namefn ambiguous*), the first two of which are required:

- type*    The name of the type, e.g., MACRO. This name will appear as the name of the index for this type, e.g., "MACRO INDEX". *type* is usually the name of a file package type, though it need not be. It must be a symbol.
- dumpfn*    The name of the function that appears as the CAR of the form that defines objects of type *type* on the file, or a list of such names. E.g., for type TEMPLATE it is SETTEMPLATE; for type VARIABLES it is (RPAQ RPAQQ RPAQ? ADDTOVAR).
- namefn*    A function that tests whether the expression that starts with *dumpfn* really is of the desired type, and returns the name of the object defined in the expression. The function takes as arguments (*expr entry*), where *expr* is the expression whose CAR matched the entry. The *testfn* should return one of the following:
  - NIL    the expression is not of the desired type.
  - name*    the expression defines a single object of this name and of the type given in the entry.
  - a list*    the value is either a single list or a list of lists, each of the form (*type . names*), meaning that the expression defines each of the names as having the specified type.

If the *namefn* is NIL or omitted, the name of the object is obtained from the second element of the expression. If that element is a list, the name is taken to be its CAR, or its CADR if the element is a quoted atom.

- ambiguous*    True if the expression is ambiguous, in the sense that even if *namefn* returns a non-NIL value, it is possible for this expression to also satisfy other entries in \*PFI-TYPES\*. E.g., the expression (RPAQ --) is ambiguous, because it could define either a variable or a constant. If *ambiguous* is true, you usually want a corresponding entry on \*PFI-FILTERS\* (below).

**\*PFI-PROPERTIES\***

[Variable]

A list used by the default handler for the PUTPROPS form. It associates property names with a type (something more specific than the type PROPERTY) under which objects having this property should be indexed. Each element is of the form (*propname type*). If *type* is NIL or omitted, then objects having this property are ignored. In addition, the default PUTPROPS handler treats all elements of the list MACROPROPS as implying type MACRO.

The initial value of \*PFI-PROPERTIES\* is

```
((COPYRIGHT)
 (READVICE ADVICE)),
```

meaning that the COPYRIGHT property should be ignored, and the READVICE property implies that the object should be indexed as type ADVICE.

**\*PFI-FILTERS\***

[Variable]

A list describing potential index entries that should be filtered out of the final index. Each element of \*PFI-FILTERS\* is a list (*type filterfn*), where *type* is one of the types in \*PFI-TYPES\* and *filterfn* is a function of one argument, an index entry. If *filterfn* returns true, then the index entry is discarded. An index entry is of the form (*name . pagenumbers*). For convenience, an element of \*PFI-FILTERS\* can also take the form (*type . subtype*), meaning that if an object is already indexed as a *subtype* then it should not also be indexed as a *type*.

The initial value of \*PFI-FILTERS\* is

```
((VARIABLES . CONSTANTS)),
```

meaning that "variables" that successfully index as constants should not also be listed in the VARIABLES index. This extra pass is needed because the CONSTANTS File Manager command causes expressions of the form (RPAQ *var value*) to be dumped on the file, and at the time this expression is read, it is not known whether there will later on appear a CONSTANTS form for the same variable.

Filter functions may want to call the following function:

```
(PFI.LOOKUP.NAME name type)
```

[Function]

Looks up *name* in the index being built for type *type*. If it finds an entry, it returns it. Index entries are of the form (*name . pagenumbers*). It is permissible for a filter function as a side effect to destructively change another index entry by adding page numbers to it. You might want to do so, for example, in the case where there is a kind of object that dumps two expressions on a file, each of which is a different type (according to \*PFI-TYPES\*), but you want both occurrences indexed as a single type.

## MORE EXPLICIT EXPRESSION HANDLING

The functions and variables described below allow you to completely control how certain expressions in the input file are handled. You can use these hooks to perform custom

prettyprinting, to suppress the printing of some expressions, or to perform indexing more complex than that supported by `*PFI-TYPES*`.

`*PFI-HANDLERS*`

[Variable]

An association list specifying explicit "handlers" for expressions that appear on the input file. Each element is a pair (*car-of-form* . *handler*), where *handler* is a function of one argument, an expression read from the file whose first element is *car-of-form*. The handler is completely in charge of indexing the expression and/or printing it to `*STANDARD-OUTPUT*`. Unless the handler chooses to suppress the printing altogether, it is expected to print at least one blank line first, so that expressions are attractively separated in the listing (see `PFI.MAYBE.NEW.PAGE`).

`*PFI-PREVIEWERS*`

[Variable]

This list is used when `Prettyfileindex` is used by the `SEE` command. During the `SEE` command, real-time performance is important, so it is undesirable to have long delays while reading a very large expression. For example, all the functions in an Interlisp FNS command appear on the file inside a single `DEFINEQ` expression. If handled in the obvious way, the user would have to wait for the entire expression to be read before any output appeared. A previewer has the opportunity to read the expression in pieces and prettyprint it as it goes.

Each element of `*PFI-PREVIEWERS*` is a pair (*car-of-form* . *previewer*), where *previewer* is a function of one argument, the *car-of-form*. The previewer is called when `Prettyfileindex` encounters an expression of the form "(*car-of-form* " on the file. Its job is to read expressions from `*STANDARD-INPUT*` (currently positioned after the *car of form*) until it encounters the closing right parenthesis, which it should consume, and prettyprint the elements appropriately to `*STANDARD-OUTPUT*`. `*PFI-PREVIEWERS*` is used only from the `SEE` command, so indexing is not necessary (but also not harmful, other than to waste some time).

If an expression does not have a previewer, `Prettyfileindex` reads the rest of the expression itself and handles it normally, i.e., performs `(PFI.HANDLE.EXPR (CONS car-of-form (CL:READ-DELIMITED-LIST #\)))`.

`(PFI.DEFAULT.HANDLER expr)`

[Function]

This is the function `Prettyfileindex` uses to process expressions that have no explicit handler. It indexes the expression according to `*PFI-TYPES*` and then prettyprints the expression. You can call this function from your handler if you decide you have an expression you didn't want to handle specially.

`(PFI.HANDLE.EXPR expr)`

[Function]

Performs `Prettyfileindex`'s normal handling of the expression *expr*, including looking on `*PFI-HANDLERS*`. Handlers and previewers of forms that encapsulate arbitrary expressions, such as `DECLARE:`, typically call this to process subexpressions.

(PFI.ADD.TO.INDEX *name type/entry*) [Function]

Adds an entry to the index for *type/entry* specifying that *name* occurs on the current page. *type/entry* is either a type or an entry from \*PFI-TYPES\* from which the type will be extracted.

(PFI.PRETTYPRINT *expr name formflg*) [Function]

Prettyprints *expr*. Optional *name* is the name of the object being printed; if a page crossing occurs in the middle of the prettyprinting, this name will be displayed in the page header. If *formflg* is true, print the expression as code; otherwise as data.

(PFI.MAYBE.NEW.PAGE *expr minlines*) [Function]

Starts a new page if the listing is currently near the bottom of the page and *expr* won't fit, else performs a single (TERPRI). If *minlines* is specified, it is an explicit estimate of how much space the expression will require, in which case *expr* can be NIL; otherwise, the function estimates the size. Handlers should call this *before* calling PFI.ADD.TO.INDEX, so that the page number in the index is correct. The typical handler calls PFI.MAYBE.NEW.PAGE, then PFI.ADD.TO.INDEX, then prints the expression, possibly via PFI.PRETTYPRINT.

## OTHER VARIABLES

\*PFI-INDEX-ORDER\* [Variable]

A list of types (as in \*PFI-TYPES\*) in the order in which the various types should appear in the index. Types not in this list are printed in an order of the program's choosing, currently a "best fit" algorithm (print the largest type index that will fit on the page). The initial value is (FUNCTIONS), meaning that the function index will appear first, with no constraints on the order of other types.

\*PFI-PRINTOPTIONS\* [Variable]

A plist of print options that PRETTYFILEINDEX appends to the list of print options passed to LISTFILES, thus supplying some printing defaults. The initial value is (REGION (72 54 504 702)), which on standard letter size paper in portrait mode results in left, bottom, top, and right margins of 1", 3/4", 1/2" and 1/2", respectively. If the print options passed to LISTFILES call for a two-sided listing, the default region is shifted 1/4" to the left. If the print options specify LANDSCAPE mode, the default region is ignored. Any REGION option specified in \*PFI-PRINTOPTIONS\* must be in points; it is scaled appropriately to the actual hardcopy device being used.

\*PFI-MAX-WASTED-LINES\* [Variable]

If an expression looks like it won't fit on the current page and there are no more than this many lines remaining on the page, PRETTYFILEINDEX starts a new page before printing the expression. A floating-point value indicates a fraction of the page; an integer indicates an absolute number of lines. The initial value is 12.

\*PFI-CHARACTER-TRANSLATIONS\* [Variable]

A list specifying how certain characters should be rendered on the output stream. This is used to get around the poor rendering of certain characters in the default font. Each

element is of the form (*imagetype* . *charpairs*), where *imagetype* is the type of image stream being printed to and each element of *charpairs* is an alist whose elements are of the form (*sourcecode* *destcode* . *looks-plist*), specifying the character code to use on the destination image stream for a specified character code in the input stream. If *looks-plist* is non-NIL, *destcode* is printed in a font obtained by applying FONTCOPY to the current font and *looks-plist*.

The initial value is

```
((INTERPRESS (95 172)
 (96 169 FAMILY CLASSIC)
 (39 185 FAMILY CLASSIC)))
```

meaning if the output stream is an Interpress stream the lister should turn character 95 (underscore) into 172 (left arrow), backquote into left single quote in the Classic font (of the same size and weight), and single quote into right single quote in Classic.

\*PRINT-PRETTY-FROM-FILES\* [Variable]

If true, the SEE (in the Exec and Filebrowser), PF and PF\* commands attempt to prettyprint to the display, rather than copying the file as it is currently formatted. The initial value is T.

\*PRINT-PRETTY-BITMAPS\* [Variable]

If true, then when \*PRINT-ARRAY\* is true and a bitmap is to be printed to an image stream, the bitmap itself is displayed as an image on the stream, rather than as the machine-readable representation of its bits (of the form #(16 16)H@@@L...). This variable has no effect on printing to files, such as in MAKEFILE, nor on PRETTYFILEINDEX, which binds it true; thus, changing the value mainly affects the display. The initial value is T.

\*PFI-DONT-SPAWN\* [Variable]

If NIL, LISTFILES arranges for a separate process to do the hardcopying (whether using PRETTYFILEINDEX or not) and returns immediately; if T, it makes the listing directly, not returning until it is finished. The initial value is NIL.

## LISTING ELSEWHERE THAN THE PRINTER

Ordinarily, you call LISTFILES (or uses the File Browser) to create listings. However, you can also call PRETTYFILEINDEX directly if you want to direct the output elsewhere, such as to an Interpress file:

(PRETTYFILEINDEX *filename printoptions outstream dontindex*) [Function]

Lists *filename*, the name of a Lisp source file or a stream open for input on such a file, printing it and its index to *outstream*. *outstream* is either an open image stream, or NIL, in which case the output goes to (OPENIMAGESTREAM) and the stream is closed afterwards, which results in it being sent to the default printer. If *filename* or *outstream* is open on entry, it is left open on exit. *printoptions* is a plist of options of interest to either LISTFILES or OPENIMAGESTREAM. If *dontindex* is true, no index is produced; this argument is used by the SEE command.



If the file is not a File manager file, PRETTYFILEINDEX takes no action and returns NIL; otherwise, it returns the full file name. However, if *filename* is an open stream, then PRETTYFILEINDEX copies the remainder of the stream to *outstream* (which must be given) using PFCOPYBYTES, and returns the full file name. This is so that the stream does not need to be backed up after discovering that the file is not a File Manager file, an operation not possible for a sequential-access stream.

## LIMITATIONS

Prettyfileindex assumes that the default font, which is used to print the index, is fixed-width.

Prettyfileindex uses the regular Interlisp prettyprinter. This means that if you have File Manager commands that produce their output in a customized way, e.g., by printing inside the E command, then the output will look different between MAKEFILE and PRETTYFILEINDEX. You can usually remedy this by supplying PRETTYPRINTMACROS for the types of expressions your command dumps (which may also let you replace the E with a simpler P command), or by defining handlers for the expressions (see \*PFI-HANDLERS\*). PRETTYFILEINDEX already supplies PRETTYPRINTMACROS for most of the customized printing done by the current File Manager: RPAQ, RPAQQ, RPAQ?, ADDTOVAR, PUTPROPS and COURIERPROGRAM.

With the exception of noticing the reader environment and DEFDEFINER expressions, PRETTYFILEINDEX does not interpret the contents of the file. If your file depends on itself for proper prettyprinting or indexing, you need to LOAD (or possibly just LOADFROM) the file first.

---

---

**PRINTERMENU**

---

---

By: ISLWS (Bloomberg.pa@Xerox.com)

**DESCRIPTION**

Creates a menu which displays all printers in the global list DEFAULTPRINTINGHOST, allows selection of a default printer, and permits addition and deletion of printers from DEFAULTPRINTINGHOST. Printers are displayed in the same order as they appear in DEFAULTPRINTINGHOST. Selecting an item from the menu will highlight by inversion and move it to the top of the menu, thus becoming the default printer. Selection in the title bar of the menu with the left or middle button will allow you to add or to delete a printer from the menu.

An auxiliary process, PRINTERMENU.WATCH, monitors the value of DEFAULTPRINTINGHOST and will update the menu if this variable is changed. If PRINTERMENU.WATCH is killed, the menu will be grayed out to indicate that it may no longer be valid. Clicking left or middle buttons inside the menu will restart PRINTERMENU.WATCH and update the menu.

**To use:**

Load the module with:

(LOAD 'PRINTERMENU.LCOM)

Start it with:

(PRINTERMENU)

**User Switches:**

1. Set DEFAULTPRINTINGHOST to contain all printers from which you wish to select.
2. Prior to calling the function (PRINTERMENU), the global variable PRINTERMENU.POSITION can be set to the position, in screen coordinates, where you want the menu to appear. If not set, you will be prompted for a position for the menu window.

---

---

## PROGRAMCHAT

---

---

By: ISLWS (bloomberg.pa@xerox.com)

### DESCRIPTION

PROGRAMCHAT is a Lisp function that invokes a windowless Chat process to execute a single command line on a remote host. PROGRAMCHAT requests a *login* if one has not been made recently to the remote host. After execution of the command, a normal *logout* is performed, and the Chat connection is closed.

PROGRAMCHAT was written by Eric Schoen to allow initiation of remote computation from Lisp workstations. It works with both VMS and Unix operating systems on the remote host.

### To use:

Load the module with:

```
(LOAD 'PROGRAMCHAT.LCOM)
```

Invoke the function with:

(PROGRAMCHAT *hostname* *commandString* *windowFlg*) [Function]

where

*hostname* is the network name of the remote host,

*commandString* is a string which is the exact format of the command to be run from the command line interpreter of a VAX/VMS host (or from the shell of a VAX/Unix host), and

*windowFlg* is a variable that, when T, opens a window and displays a log of data transferred between the PROGRAMCHAT process and the remote host. PROGRAMCHAT is normally invoked with *windowFlg* = NIL.

### Warnings:

1. When loaded, PROGRAMCHAT resets the variable NETWORKLOGINFO.
2. PROGRAMCHAT provides no error handling. If the connection to the remote host is broken, no error message is returned.

---



---

## PROMPTREMINDERS

---



---

To be periodically reminded of things

By: JonL White

Revised by: Larry Masinter (Masinter.pa@Xerox.com), subsequently by Becky Burwell  
(Burwell.PA@Xerox.COM)

### INTRODUCTION

PROMPTREMINDERS implements a facility which schedules events to be performed, or messages to be flashed in a prompt window. Events can be periodic or once-only. The showing of a message in a prompt window has the extra facility of flashing a message, and stopping only when there has been a recent response (mouse or keyboard movement) from the user.

If the MESSAGE given for the reminder (see description of the function SETREMINDER below) is a listp, then when the reminder "goes off", that listp will be EVAL'd rather than any of the "winking", "flashing", or "hassling" mentioned above.

The global variable REMINDERSTREAM holds the stream where the message is to be displayed; if not set by the user, it defaults to PROMPTWINDOW. After the message has been displayed, the window (if indeed REMINDERSTREAM holds a window) will be closed, depending on the value of CLOSEREMINDERSTREAMFLG.

REMINDERS is a file package type, so that they may be easily saved on files, and so that the general typed-definition facilities may be used. On any file which uses the REMINDERS filepkgcom, it is advisable to precede this command with a command

```
(FILES (SYSLOAD COMPILED FROM LISPUSERS) PROMPTREMINDERS)
```

since this package is not in the initial Lisp loadup. When initially defining a reminder, it is preferable for the user to call SETREMINDER rather than PUTDEF; but HASDEF is the accepted way to ask if some name currently defines a "reminder", and DELDEF is the accepted way to cancel an existing "reminder".

### EXAMPLES

```
(SETREMINDER NIL (ITIMES 30 60) "Have you done a CLEANUP recently?")
```

the user wants to be reminded every 30 minutes that he ought to save his work

```
(SETREMINDER 'WOOF NIL "Call home about dinner plans."
 "8-Jan-83 4:00PM")
```

he merely wants to be told once, at precisely 4:00PM to call home

```
(SETREMINDER NIL 600
 '(PROGN (AND (FIND.PROCESS 'LISTFILES) (add FREQ 1))
 (add TOTAL 1)))
```

checks every 10 minutes to see if there is a process called LISTFILES

## FUNCTIONS

(SETREMINDER NAME *PERIOD MESSAGE INITIALDELAY EXPIRATION*) [Function]

This will create and install a "reminder" with the name NAME (NIL given for a name will be replaced by a gensym), which will be executed every PERIOD number of seconds by winking the string MESSAGE into the prompt window; if MESSAGE is null, then NAME is winked; if MESSAGE is a listp, then it is EVAL'd and no "winking" takes place. "Winking" means alternately printing the message and clearing the window in which it was printed, at a rate designed to attract the eye's attention.

The first such execution will occur at PERIOD seconds after the call to SETREMINDER unless INITIALDELAY is non-NIL, in which case that time will be used; a numeric value for INITIALDELAY is interpreted as an offset in seconds from the time of the call to SETREMINDER, and a stringp value is an absolute date/time string.

If PERIOD is null, then the reminder is to be run precisely once. If EXPIRATION is non-null, then a fixp means that that number of seconds after the first execution, the timer will be deleted; a stringp means a precise date/time at which to delete the timer.

Optional 6th and 7th arguments -- called REMINDINGDURATION and WINKINGDURATION -- permit one to vary the amount of time spent in one cycle of the wink/flash loop, and the amount of time spent winking before initiating a "flash". The attention-attracting action will continue for REMINDINGDURATION seconds (default: the value of the global variable DEFAULT.REMINDER.DURATION which is initialized to 60), or until some keyboard action takes place.

Type-ahead does not release the winking. In case the user fails to notice the winking, then every WINKINGDURATION seconds (default: the value of the global variable DEFAULT.REMINDER.WINKINGDURATION which is initialized to 10) during the "reminding", the whole display videocolor will be wagged back and forth a few times, which effects a most obnoxious stimulus.

SETREMINDER returns the name (note above when NIL is supplied for the name).

(ACTIVEREMINDERNAMES) [Function]

ACTIVEREMINDERNAMES returns the list of active reminders.

(REMINDER.NEXTREMINDDATE NAME DATE) [Function]

REMINDER.NEXTREMINDDATE returns (and optionally sets) the date&time (in DATE format) at which the reminder is next to be executed.

(REMINDER.EXPIRATIONDATE NAME DATE) [Function]

REMINDER.EXPIRATIONDATE returns (and optionally sets) the date&time (in DATE format) at which the reminder will be automatically deleted.

(REMINDER.PERIOD *NAME SECONDS*)

[Function]

REMINDER.PERIOD returns (and optionally sets) the period (in seconds) at which the reminder gets rescheduled.

(SHOWDEF *name* 'REMINDERS)

[Function]

will show a reminder in a pretty format, etc.

---

---

## Proofreader

---

---

By: John Maxwell (Maxwell.pa@Xerox.com)

Use in conjunction with Analyzer, SpellingArray

### INTRODUCTION

The Proofreader interactively looks for and corrects spelling errors in a given TEdit document. To use it, go to the TEdit menu (click the middle button while in the title bar) and invoke the menu item labeled "Proofread". This will simultaneously attach a special menu to the side of the document and start proofreading from the caret. The proofreader scans the document from the caret location and stops at the first misspelling it finds, highlighting it with a pending delete selection. Successive misspellings can be found by clicking the "Proofread" menu item in the TEdit menu or the "Proofread" menu item in the new side menu.

At this point, you can either correct the misspelled word or skip to the next misspelling. If you are not sure what the correct spelling is, you can get a menu of possible corrections by invoking the "Correct" menu item. Selecting a correction from this menu will cause it to be automatically inserted into the document in place of the misspelling. (Note: The Proofreader occasionally suggests some very bizarre spelling corrections for your misspelled word. Do not be alarmed; this is a known but unavoidable artifact of the heuristic used for checking for misspellings. (see notes at the end)) If the word was erroneously flagged as misspelled, then you can insert the word into your personal word list of acceptable words by invoking the "Insert" menu item. At the end of the editing session you can save the word list on a file with the "StoreWordList" command (see below).

If when the Proofreader is invoked the current selection has more than one word in it, then the Proofreader will only correct the words in that selection. Otherwise, the Proofreader always proofreads the text from the caret to the end of the document.

### PROOFREADER SUB-COMMANDS

Under the TEdit Proofread menu item there are a number of sub-commands that the user can invoke. They are:

#### **Proofread**

The same as the top level Proofread command. Attaches a special menu to the side of the document and starts proofreading from the caret.

#### **CountWords**

Counts the number of words in the current selection. To count the number of words in the entire document, first click "All" in TEdit's expanded menu.

#### **SetProofreader**

Gives the user a menu of proofreaders to use for proofreading. If there is only one proofreader, no menu is generated. If the user selects a remote server, a second menu may be generated of proofreaders available on the server.

**StoreWordList**

Allows the user to save the words that he inserted into the proofreader onto a remote file. The words from an existing version of the file will be read in first and then a new file will be generated consisting of the old file plus the new words. After the file is written, the list of newly inserted words is set to NIL.

**LoadWordList**

The inverse of **StoreWordList**. If you have a file that you want to load every time you use the proofreader, you can add it to `Proofreader.AutoLoad`, and it will be loaded when the proofreader is first opened. `Proofreader.AutoLoad` can be either a file or a list of files.

**AutoCorrect**

Sets the variable `Proofreader.AutoCorrect` so that the proofreader *automatically* generates a list of corrections whenever it finds a misspelled word. In the AutoCorrect mode, the proofreader will also automatically scan for the next misspelled word whenever after a correction has been selected. If you want to stop the proofreading process, click outside of the correction menu. If you want to insert the flagged word into your word list, click the menu item labeled `"*INSERT*"`. If you want to continue proofreading without changing the flagged word, select the menu item labeled `"*SKIP*"`.

**ManualCorrect**

Sets the variable `Proofreader.AutoCorrect` so that the proofreader will not generate a list of corrections unless the user asks for it.

**PROOFREADER VARIABLES**

There are a couple of variables that the user can set in his init file to change how the proofreader works. They are:

`Proofreader.AutoLoad` [Variable]

A file or list of files to be loaded into the proofreader every time that the proofreader is initialized.

`Proofreader.AutoCorrect` [Variable]

A boolean that determines whether or not a list of corrections is automatically generated whenever the proofreader finds a misspelled word. The default value is NIL.

`Proofreader.AutoDelete` [Variable]

A boolean that determines whether or not to delete the old versions of a word list file when a new one is written out. The default value is T.

`Proofreader.MenuEdge` [Variable]

The side of the window that the proofreader menu appears on (can be either LEFT or RIGHT). The default value is LEFT.

`Proofreader.UserFns` [Variable]

A list of functions to be applied to misspelled words (as strings). If the function returns a non-NIL value, then the word is assumed to be correctly spelled. (If the function `ATOMHASH#PROBES` is added to `Proofreader.UserFns`, then any word defined as an atom becomes legal. `ATOMHASH#PROBES` tests whether or not a string is an atom without creating new atoms. If you want a more restricted test (i.e. "anything defined as a procedure is legal") first test that the string



exists at an atom before doing MKATOM. Otherwise, the atom space will fill up with misspelled words.)

#### NOTES

- The proofreader uses a heuristic to determine whether or not a word is in its word list that occasionally will produce false positives. This is most noticeable when the proofreader is generating corrections for misspelled words. I don't know of any way to eliminate this problem except to use a different algorithm, the fastest of which is at least twice as slow. Hopefully people will find it more of a nuisance than a real problem.
- There is no way to remove words once they have been inserted into the local dictionary. The only way to get rid of a bad word is to reload the dictionary. This can be done by reloading the SpellingArray file. If a bad word gets into one of the remote files, you can edit the file to get rid of it.

#### ACKNOWLEDGMENTS

The algorithm used in the Proofreader is based on the algorithm in the Cedar Spelling Tool by Bob Nix. For more information on the implementation, see the section "How it Works" in {Cyan}<CedarChest6.1>Documentation> SpellingToolDoc.tioga.

---

---

## PSEUDOHOSTS

---

---

By Ron Kaplan

This document was created in January 2022.

A pseudohost identifies the root of a file system that exists as a subdirectory of another pre-existing file system. This gives a shorthand way of operating on a file in the subdirectory of a particular project without having to specify in the name of that file the entire path to its location in a larger file system. For example, suppose that the variable `MEDLEYDIR` contains the path from `{DSK}` to the subdirectory that contains all Medley system files (e.g. `{DSK}<Users>kaplan>Local>medley>`). If not connected to that subdirectory, then the file `COREIO`, say, would have to be reference as

```
{DSK}<Users>kaplan>Local>medley>sources>COREIO.
```

If `MEDLEY` is defined as a pseudohost with `MEDLEYDIR` as its prefix, then that file can also be identified more succinctly as `{MEDLEY}<sources>COREIO`.

This package implements pseudohost file devices that allow files to be specified and manipulated in this way. The function `PSEUDOHOST` defines a new pseudohost whose files coincide with the files at the end of a prefix directory path:

```
(PSEUDOHOST HOST PREFIX)
```

[Function]

For the Medley example, executing

```
(PSEUDOHOST 'MEDLEY MEDLEYDIR)
```

will set up `MEDLEY` as a (pseudo) host name that can be used to reference Medley system files. As another example,

```
(PSEUDOHOST 'NC "{DSK}<Users>kaplan>Local>notecards>"
```

would provide a shorthand `{NC}` for accessing the files that make up the Notecards application. Note that the prefix is always interpreted as a directory path, whether or not it ends with an explicit `>` or `/`.

If `PSEUDOHOST` is called with the name of a previously defined pseudohost but with a different prefix, the new prefix replaces the old. If the prefix is `NIL`, the pseudohost is removed. If `HOST` is a list and `PREFIX` is `NIL`, `HOST` is interpreted as a (host prefix) pair. The target host defaults to `DSK` if `PREFIX` does not have an explicit host.

Of course, full filenames can continue to be used for file access--the pseudohost just provides a systematic abbreviated naming convention. Apart from this convenience, a further advantage is that pseudohost file designators will remain valid if directories are relocated or even moved to different machines, provided that the pseudohost is redefined for the new environment.

Pseudohosts can be used wherever hosts or directories are normally specified. Thus `(CNDIR '{MEDLEY})` or `cd {MEDLEY}` will connect to the underlying `MEDLEYDIR` directory, but the shorter pseudohost will appear in the file and path names returned by the usual system functions (`INFILEP`, `DIRECTORYNAME`, `FB`, `*`, `...`). When connected to `{MEDLEY}<sources>`, `(INFILEP 'FILEPKG)` will return `{MEDLEY}<sources>FILEPKG`.

When `PSEUDOHOSTS` is loaded, it executes

```
(PSEUDOHOST 'LI LOGINHOSTDIR)
```

so that files in the login directory can always be referenced succinctly with host `{LI}`, even while connected to another directory.

- (PSEUDOHOSTS) [Function]  
Returns the (host prefix) pairs of all currently defined pseudohosts.
- (PSEUDOHOSTP HOST) [Function]  
Returns the (host prefix) pair for a particular pseudohost, `NIL` if `HOST` is not a pseudohost.
- (TARGETHOST HOST) [Function]  
Returns the target host of a particular pseudohost, `NIL` if `HOST` is not a pseudohost.
- (TRUEFILENAME FILE) [Function]  
Returns the name of `FILE` in its true device, essentially replacing `FILE`'s `host` by its prefix if it is a pseudohost. Returns `FILE` (possibly extended with the prefix of the connected directory) if its host is not a pseudohost. `FILE` may be a stream as well as a name. It may also be a list, in which case the value is the result of applying `TRUEFILENAME` to each of its elements.
- (PSEUDOFilename FILE) [Function]  
Returns the name of `FILE` in its pseudo device, if any, essentially replacing `FILE`'s `prefix` by the hostname of a pseudodevice for that prefix. Returns `FILE` (possibly extended with the prefix of the connected directory) if it does not match a pseudohost prefix. `FILE` may be a stream as well as a name. It may also be a list, in which case the value is the result of applying `PSEUDOFilename` to each of its elements.

---

**PS-SEND**

---

By: Matt Heffron (mheffron@orion.cf.uci.edu)

Requires: POSTSCRIPTSTREAM, UNIXCOMM (only on Sun)

The module PS-SEND.LCOM is required by the PostScript ImageStream driver, and will be loaded automatically when POSTSCRIPTSTREAM.LCOM is loaded. It contains the function (POSTSCRIPT.SEND) which is called by SEND.FILE.TO.PRINTER to actually transmit the file to the printer. It is, by its nature, quite site specific, so it is in a separate file to make modifying it for any site relatively simple. For Sun Medley users, code has been added to send the completed Postscript files to the local printer (whatever the UNIX environment variable "PRINTER" is set to.)

POSTSCRIPT.SEND can handle the simple cases of copying a file to a spool directory or directly to a specific device (using COPYBYTES). The information about how to send a file to a specific host is expected to be on the SPOOLDIRECTORY, SPOOLFILE, SPOOLOPTIONS, HOST.CONTROL.STRING and HOST.CONTROL.AFTER.STRING properties of the host name. It checks first for the SPOOLFILE property on the host name, which must be a full filename that can be opened (by OPENSTREAM). If there is no SPOOLFILE property, then it checks for a SPOOLDIRECTORY property, if there, it will be concatenated together with a generated filename (by (GENSYM USERNAME)) and a ".PS" extension. If either the SPOOLFILE or SPOOLDIRECTORY properties exist, then an output stream will be opened onto the specified file. The value of the SPOOLOPTIONS property on the host name (if any) will be passed as the PARAMETERS argument to OPENSTREAM, and must be an appropriately formed list. (This is useful for cases where the specified destination is an 11xx device, such as {TTY} or {RS232} where you must set additional attributes of the stream like baud rate, etc.) If there is no SPOOLFILE or SPOOLDIRECTORY properties, then nothing will be sent and a message will appear in the PROMPTWINDOW ("[Unable to send FILE to HOST.]").

After the output stream is opened, if there is a HOST.CONTROL.STRING property of the hostname, then that string will be printed (IL:PRIN1) to the output stream first, then the first line of the file being sent (for a file generated by the PostScript ImageStream driver, this is the "%! ..." line), then the value of the POSTSCRIPT.CONTROL.STRING from the PRINTOPTIONS argument to POSTSCRIPT.SEND, finally the rest of the input file. (The idea of the HOST.CONTROL.STRING is that it should be a string to control the printing host itself, or perhaps a routing device that is mid-stream between the 11xx and the printer itself. For example, using a SPOOLFILE of "{TTY}FOO.PS" and having the PostScript printer shared by several additional computers (e.g. PC's) by use of a device like the Logical Connection from Fifth Generation Systems, it might be necessary to send a command to the Logical Connection to specify to route the output from this input to the output which is the PostScript printer.) Likewise, if there is a HOST.CONTROL.AFTER.STRING property of the hostname, then that string will be printed to the output stream last, just before closing the stream.

---

---

**PS-SKETCH-PATCH**

---

---

By: Will Snow (snow.envos@xerox.com)

Requires: SKETCH, POSTSCRIPTSTREAM

This module fixes some bugs in SKETCH when interacting with the PostScript driver. This will make the printing of sketches with text in them reasonable. It SYSLOADs SKETCH (from LISPUSERSDIRECTORIES) if it is not already loaded, it does not load POSTSCRIPTSTREAM.

---

---

## PS-TTY

---

---

By: Matt Heffron (mheffron@orion.cf.uci.edu)

Requires: POSTSCRIPTSTREAM, PS-SEND, DLTTY

The module PS-TTY defines a printing host named PS-TTY which sends PostScript output to a printer over the {TTY} port of the 11xx. It also puts a function onto AROUNDXITFNS which reinitializes the {TTY} after returning from LOGOUT. The BaudRate and other parameters of the {TTY} port are controlled by the following variables.

### VARIABLES

PS-TTY-BAUD [InitVariable]

This is the BaudRate for the {TTY} port output stream. Defaults to: 4800.

PS-TTY-DATABITS [InitVariable]

This is the BitsPerSerialChar for the {TTY} port output stream. Defaults to: 8.

PS-TTY-PARITY [InitVariable]

This is the Parity for the {TTY} port output stream. Defaults to: NONE.

PS-TTY-STOPBITS [InitVariable]

This is the NoOfStopBits for the {TTY} port output stream. Defaults to: 1.

PS-TTY-FLOWCONTROL [InitVariable]

This is the FlowControl for the {TTY} port output stream. Defaults to: XOnXOff.

---



---

## QEdit

---



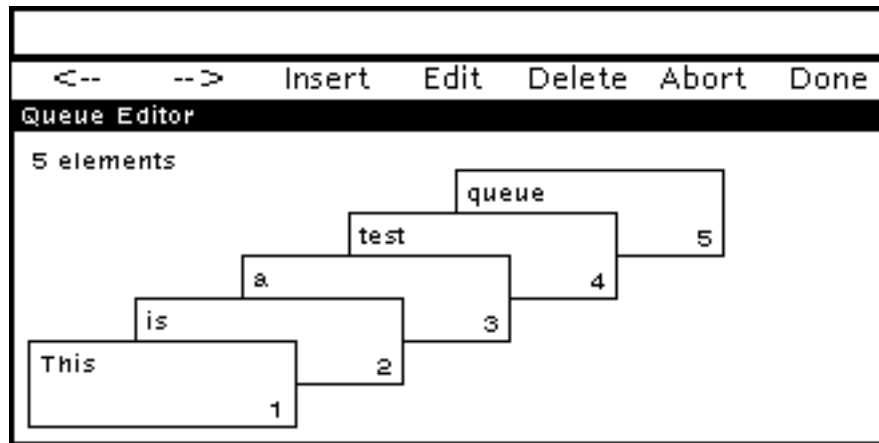
---

By: Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester.edu)

This document last edited on January 22, 1988

### INTRODUCTION

QEdit is a facility for editing queues, which are ordered lists containing arbitrary objects. QEdit provides facilities for reordering, editing and deleting current queue entries, and inserting new entries. A QEditor looks like this:



Clicking the left mouse button over a queue entry makes it the current selection. You can then select a QEdit menu operation. QEdit returns with the original list if aborted, or a new list upon normal exit. Selecting an entry that is not entirely visible causes the display to scroll until it is.

### DETAILS

(QEDIT *QUEUE PROPS*)

[Function]

Invokes a QEditor on *QUEUE*. Returns *QUEUE* if aborted, a new list otherwise. The *PROPS* argument is a property list which serves to customize the behavior of QEDIT for this particular invocation. Currently defined props are listed below.

<—

[QEdit command]

Moves the current selection forward, i.e., switches the current selection and the queue entry just before it.

—>

[QEdit command]

Moves the current selection backward, i.e., switches the current selection and the queue entry just after it.

|                                                                                                                                                                                                                                                                                                |                  |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Insert                                                                                                                                                                                                                                                                                         | [QEdit command]  |
| Inserts a new entry in front of the current selection, provided <i>PROPS</i> contained an INSERTFN.                                                                                                                                                                                            |                  |
| Edit                                                                                                                                                                                                                                                                                           | [QEdit command]  |
| Edits the current selection, provided <i>PROPS</i> contained an EDITFN.                                                                                                                                                                                                                        |                  |
| Delete                                                                                                                                                                                                                                                                                         | [QEdit command]  |
| Deletes the current selection, provided <i>PROPS</i> contained a DELETFN.                                                                                                                                                                                                                      |                  |
| Abort                                                                                                                                                                                                                                                                                          | [QEdit command]  |
| Aborts the current QEdit session, returning the original queue.                                                                                                                                                                                                                                |                  |
| Done                                                                                                                                                                                                                                                                                           | [QEdit command]  |
| Ends the current QEdit session, returning a new list containing the current queue entries.                                                                                                                                                                                                     |                  |
| TITLE                                                                                                                                                                                                                                                                                          | [QEdit property] |
| If supplied, its value becomes the title for the QEdit window.                                                                                                                                                                                                                                 |                  |
| CONTEXT                                                                                                                                                                                                                                                                                        | [QEdit property] |
| The value of the CONTEXT property is passed to the functions below as an extra argument. It does not affect QEdit directly. The functions below can also obtain the current queue by calling the function QEDIT.CURRENT.QUEUE (see below).                                                     |                  |
| LABELFN                                                                                                                                                                                                                                                                                        | [QEdit property] |
| If supplied, its value is a function which, when invoked on a queue entry and the user context, returns a label to use for displaying the queue entry. If not supplied, QEdit displays the queue entry itself.                                                                                 |                  |
| LABELFONT                                                                                                                                                                                                                                                                                      | [QEdit property] |
| If supplied, its value is a font specification for displaying the queue entry.                                                                                                                                                                                                                 |                  |
| INSERTFN                                                                                                                                                                                                                                                                                       | [QEdit property] |
| If supplied, its value is a function which, when invoked on the user context, returns either NIL or a new element to be inserted in front of the current selection (at the front of the queue, if there is no current selection). If not supplied, no elements can be inserted into the queue. |                  |
| EDITFN                                                                                                                                                                                                                                                                                         | [QEdit property] |
| If supplied, its value is a function which, when invoked on a queue entry and the user context, returns either NIL or a (possibly new) entry to be used instead of the current selection.                                                                                                      |                  |
| DELETFN                                                                                                                                                                                                                                                                                        | [QEdit property] |
| If supplied, its value is a function which, when invoked on a queue entry and the user context, returns NIL if the entry should not be deleted. If not supplied, no elements can be deleted. Hint: the function TRUE always returns T.                                                         |                  |
| (QEDIT.CURRENT.QUEUE)                                                                                                                                                                                                                                                                          | [Function]       |
| Invoked from one of the above mentioned functions, returns the queue being edited in its current form. The INSERTFN might use this, for example, if duplicates are not allowed.                                                                                                                |                  |



(QEDIT.RESET)

[Function]

QEdit will reuse QEditors upon reinvocation. This function will throw away any known but currently inactive QEditors. Useful if you wish to change the default QEdit props.

\*QEDITPROPS\*

[QEdit variable]

Any props not explicitly supplied in the call to the function QEDIT are taken from this free variable. Its initial value is (TITLE "Queue Editor" LABELFONT (HELVETICA 8))

### Examples

(QEDIT '(This is a test queue))

[Function]

Brings up a QEditor as shown above. Queue elements can be rearranged, but not added to, edited or deleted.

(QEDIT '(This is a test queue) '(INSERTFN READ DELETEFN TRUE))

[Function]

Brings up a QEditor as shown above. Queue elements can be rearranged, inserted or deleted, but not edited.

---



---

## READAIS

---



---

By: Nick Briggs (Briggs.pa@Xerox.com)

### INTRODUCTION

AIS (array of intensity samples) is a format for color and gray- level images. The following functions allow reading and writing of AIS files from Lisp.

*(AISBLT FILE SOURCELEFT SOURCEBOTTOM DESTINATION DESTLEFT DESTBOTTOM WIDTH HEIGHT HOW FILTER NBITS LOBITADDRESS)* [Function]

Puts the image in an AIS file into a bitmap. AISBLT checks the sample size of the AIS file and the number of bits per pixel of the DESTINATION and performs the required reduction (if any). SOURCELEFT and SOURCEBOTTOM give the left and bottom coordinates in the source file of the image to be read (default to (0,0)). DESTINATION can be a bitmap, a color bitmap, or a window.

HOW indicates what method of reduction is to be used if the sample size of the AIS file is larger than the number of bits per pixel in the destination bitmap. The recognized methods are TRUNCATE (use the high-order bits) and FSA (use the Floyd-Steinberg dithering algorithm). The default when going to a 1 bpp bit map is FSA; otherwise it is TRUNCATE.

FILTER if non-NIL should be an array that will be used to filter the samples read from the AIS file. If FILTER is given and a sample point of intensity N is read from the file, (ELT FILTER N) is used to determine the bits for the destination. The function SMOOTH HIST described below is one way of getting a filter that balances the contrast in an image.

NBITS and LOBITADDRESS allow an image to be read into one or more “planes” of a color bitmap. NBITS tells how many bits are to be taken from each image sample, and LOBITADDRESS indicates the lowest bit within each pixel that the NBITS bits are to go. (Bit address zero is the leftmost or highest-order bit. For a four-bit-per-pixel bit map, three would be the lowest-order bit.) This is used by SHOWCOLORAIS to put the different planes of a color image into the bit map.

*(SHOWCOLORAIS BASEFILE COLORMAPINFO HOW SOURCELEFT SOURCEBOTTOM DESTINATION DESTLEFT DESTBOTTOM WIDTH HEIGHT)* [Function]

Reads a color image from three AIS files into a color bit map. The three color files are obtained by concatenating the strings “-RED.AIS”, “-GREEN.AIS”, and “-BLUE.AIS” onto the end of BASEFILE. If COLORMAPINFO is a list of three small integers, it indicates how many of the bits in the destination are allocated to each color. For example, if DESTINATION is a four-bit-per-pixel color bit map and COLORMAPINFO is (1 2 1), one bit (bit zero) will be allocated to the red image, two bits (bits one and two) will be allocated to the green image, and one bit (bit three) will be allocated to the blue image.

DESTINATION is the color bitmap the image will be stored into.

HOW, SOURCELEFT, SOURCEBOTTOM, DESTLEFT, DESTBOTTOM, WIDTH, and HEIGHT are as described in AISBLT.

An experimental feature that is available only when going to 8 bpp color bit map: if COLORMAPINFO is a color map, each pixel will be determined by finding the color in the color map that is closest to the 24 bits of color information read from the three image files. (This takes a long time.) The function

COLOR.DISTANCE (red green blue redentry greenentry blueentry) is called to calculate the distance by which “closest” color is determined.

(CMYCOLORMAP *CYANBITS MAGENTABITS YELLOWBITS BITSPERPIXEL*) [Function]

Returns a color map that assumes the BITSPERPIXEL bits are to be treated as three separate color planes with CYANBITS bits being in the cyan plane, MAGENTABITS bits being in the magenta plane, and YELLOWBITS bits being in the yellow plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 0 and black is 255.

(RGBCOLORMAP *REDBITS GREENBITS BLUEBITS BITSPERPIXEL*) [Function]

Returns a color map that assumes the BITSPERPIXEL bits are to be treated as three separate color planes with REDBITS bits being in the red plane, GREENBITS bits being in the green plane, and BLUEBITS bits being in the blue plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 255 and black is 0.

(GRAYCOLORMAP *BITSPERPIXEL*) [Function]

Returns a color map containing shades of gray. White is 0 and black is 255.

(WRITEAIS BITMAP *FILE REGION*) [Function]

Writes the region REGION of the color bit map BITMAP onto the file FILE in AIS format. This provides an efficient way of saving color or gray- level images.

(AISHISTOGRAM *FILE REGION*) [Function]

Returns a histogram array of the region REGION in the AIS file FILE. The histogram array has as its Nth element the number of pixels in the region that have intensity N.

(GRAPHASHISTOGRAM *HISTOGRAM W*) [Function]

Draws a graph of a histogram array in the window W. If W is NIL, a window is created.

(SMOOTHEDFILTER *HISTOGRAM*) [Function]

Returns a “filter” array that maximally distributes the intensities values contained in HISTOGRAM. The filter array can be passed to AISBLT to change the contrast of the image being read.

## READAPPLEFONT

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: READDISPLAYFONT

READAPPLEFONT defines a new display font type which allows Envos Lisp to use (commercially available) Macintosh™ display fonts. Although this module is primarily intended for extracting font width information for conversion programs (eg. MacPaint™ to Sketch and vice versa) it can also be used to extend the number of display fonts available within the Envos Lisp environment.

There are *no* user functions in the module. In addition to this module, you need a directory of extracted Macintosh™ font files (as described below) and you must add the name of that directory to the system DISPLAYFONTDIRECTORIES list. The following function is added under the type APPLE to the list DISPLAYFONTTYPES (defined by the READDISPLAYFONT module).

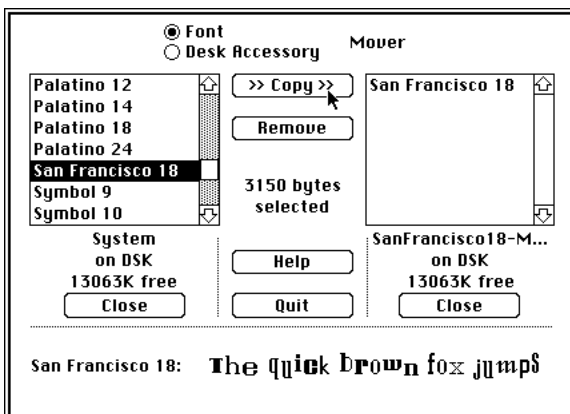
(READAPPLEFONT *STREAM FAMILY SIZE FACE*)

[Function]

The module also adds the extension *APPLE* to the system list DISPLAYFONTEXTENSIONS.

### FONT FILES

This module only uses the FontRec portion of the font files (see *Inside Macintosh™*). The font resources must be extracted into individual files with appropriate names, eg. SANFRANCISCO18-MRR-C0.APPLE. One method of doing this is to use the *Font/DA Mover™* utility:



The individual font files should be moved to Unix AUFS/CAP server (or the equivalent) and the resource forks (in the .resource directory) should be copied to the directory you added to DISPLAYFONTDIRECTORIES above.

If another method for extracting the FontRec data structure into individual files is used, the following variable will probably need to be reset:

APPLEFONTREC.OFFSET

[Variable]

The offset in bytes into the font file of where the FontRec data structure begins (initially 264).

### NOTES

- This module only handles proportionally spaced fonts and ignores fractional character widths.
- The user is responsible for determining the legality of extracting the fonts in question.

---

---

## READBRUSH

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

uses: BITMAPFNS

This document last edited on September 8, 1988.

### INTRODUCTION

This module implements two things:

(IDLE.GLIDING.BRUSH W box wait) [Function]

Like the default IDLE.BOUNCING.BOX Idle function but glides the bitmap around the screen instead of bouncing it.

(READBRUSHFILE file) [Function]

Reads files in the ".brush" format used by Mesa/Viewpoint. Returns a pair of bitmap/mask (or just (bitmap) if there is no mask. Brush file names use defaults from BRUSHDIRECTORY, initially {goofy:osbu north:xerox}<hacks>data>brushes> (and of use only inside Xerox.)

(READBRUSH file) [Function]

Calls READBRUSHFILE and then creates a window with that brush in it.

BRUSHDIRECTORY [Variable]

Default location to get brushes from.

Adds an entry to IDLE.FUNCTIONS for "Gliding box", which will use IDLE.GLIDING.BOX on the brush selected from a menu created by enumerating all of the .brush files on BRUSHDIRECTORY.

---



---

## READDATATYPE

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

READDATATYPE gives @ a read macro definition (in the **INTERLISP** readtable) so that it can be used to type in datatype pointers directly. For example, suppose you have lost your pointer to a window (or menu, etc.) but you have the printed representation around (eg. {WINDOW}#56,17470) then you can do things like:

```
90_ (INVERTW @{WINDOW}#56,17470)
{WINDOW}#56,17470
```

The read macro is only intended to be used at the *read-eval-print* loop. If the character following the @ is not a { then the read macro returns the @ character just as if you had typed it in so that other expressions that use @, like '(A B ,@FILELST C D)', will still work correctly.

Although the read macro does not need the data type name in the brackets (eg. {MENU}) to get the pointer, it does require it in order to check the pointer to make sure it is of the correct *type*. If the pointer is not of the type specified, then the read macro returns NIL.

The following form is used in the COMS of the file to set the syntax of @ in the **INTERLISP** readtable and can be used to add the capability to other readtables and/or characters:

```
(SETSYNTAX '%@ ' (MACRO FIRST READDATATYPE) (FIND-READTABLE "INTERLISP"))
```

---

---

## READDISPLAYFONT

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

READDISPLAYFONT modifies the display font functions to make it possible to define new display font types.

The functions \READDISPLAYFONTFILE and FONTFILEFORMAT are modified to use the list:

DISPLAYFONTTYPES [Variable]

An ALST containing font file extensions and the functions that can read those types from a file. Its initial value is:

```
((AC \READACFONTFILE)
 (STRIKE \READSTRIKEFONTFILE))
```

The functions take (STREAM FAMILY SIZE FACE) as arguments and return a CHARSETINFO datum. You will (probably) need the Xerox (internal) documentation about fonts and character sets (not supplied with the standard documentation) to define a new font file reading function.

The AC and STRIKE font types are handled specially to be compatible with the existing font code, so files with extension DISPLAYFONT still work and FONTFILEFORMAT moves the file pointer to the appropriately for those two types. For all other (new) types, the type is determined solely from the file extension and FONTFILEFORMAT has no side effects.

When defining a new display font types, you will need to add the new extension to the system list DISPLAYFONTEXTENSIONS.

---



---

## REGION

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

REGION facilitates having multiple complex cursor behaviors in a single window without having the CURSORMOVEDFNs, CURSORINFNs, CURSOROUTFNs, and BUTTONEVENTFNs of the behaviors know about each other. In its simplest form it can be used to implement active regions.

To use, set the various window functions of the window to the REGION window functions and put a list of REGIONEVENT records on the REGIONEVENTLST property of the window. When the cursor moves over the window, REGION checks which region it is in, calls the CURSOROUTFN of the previous region and the CURSORINFN of the new region. If regions overlap, then the appropriate functions will be called on all regions affected by the mouse event.

The REGIONEVENTLST property of the window should contain a list of REGIONEVENT records which have the fields:

|                 |                                                                                                             |
|-----------------|-------------------------------------------------------------------------------------------------------------|
| EVENTREGION     | A REGION record which is the region of the window over which the region specific functions will be invoked. |
| REGIONBUTTONFN  | (WINDOW POSITION REGION REGIONEVENT)                                                                        |
| REGIONMOVEDFN   | (WINDOW POSITION REGION REGIONEVENT)                                                                        |
| REGIONINFN      | (WINDOW REGION REGIONEVENT)                                                                                 |
| REGIONOUTFN     | (WINDOW REGION REGIONEVENT)                                                                                 |
| REGIONREPAINTFN | (WINDOW REGION REGIONEVENT)                                                                                 |
| ACTIVEREGION    | Boolean indicating if the region is active or not.                                                          |
| REGIONFLAGS     | User defined identification flags.                                                                          |
| REGIONUSERDATA  | User defined field.                                                                                         |

All of the fields in the REGIONEVENT record are optional. If a REGIONEVENT record has a NIL EVENTREGION, then it is considered the default REGIONEVENT and will be invoked whenever a mouse event occurs outside of any other region.

The REGION window functions are:

```
(WINDOWPROP WINDOW 'CURSORINFN (FUNCTION REGIONINFN))
(WINDOWPROP WINDOW 'CURSOROUTFN (FUNCTION REGIONOUTFN))
(WINDOWPROP WINDOW 'REPAINTFN (FUNCTION REGIONREPAINTFN))
(WINDOWPROP WINDOW 'CURSORMOVEDFN (FUNCTION REGIONMOVEDFN))
(WINDOWPROP WINDOW 'BUTTONEVENTFN (FUNCTION REGIONEVENTFN))
```



The above window properties can be set using the function:

(REGION.INIT WINDOW [*REGIONEVENTLST* *SAVE?*]) [Function]

The *REGIONEVENTLST* is a list of *REGIONEVENT* records to put on the window. If *SAVE?* is non-NIL, the *CURSORINFN*, *CURSOROUTFN*, etc. of the window are put into a default region event record (one with an *EVENTREGION* = NIL) and added to the *REGIONEVENTLST*. The macro:

(ADDREGIONEVENT *REGIONEVENT* WINDOW) [Macro]

can be used to add a *REGIONEVENT* record onto the current *REGIONEVENTLST* of *WINDOW*.

The *REGIONFLAGS* field of the *REGIONEVENT* record consists of whatever atoms the user wishes to identify regions with. These allow the user to issue commands such as “turn off all regions marked *GRAPH*”, “activate all the *MENU* regions”, etc.

(ACTIVATEREGIONS *FLAGS* WINDOW) [Function]

(DEACTIVATEREGIONS *FLAGS* WINDOW) [Function]

Activate and deactivate all the *REGIONEVENT* records on *WINDOW* whose *REGIONFLAGS* have a flag in common with *FLAGS*. If *FLAGS* is T, activate or deactivates all *REGIONEVENT* records.

DISABLEFLG [Variable]

If set to T, disables all of the region functions for all windows using the *REGION* module. Alternatively, setting *DISABLEFLG* to a window, or list of windows, disables all the windows using the *REGION* package except for those windows. This allows selectively turning off cursor actions on parts of the screen.

---

---

## REGIONMANAGER

---

---

By Ron Kaplan

This document created in December 2021, last edited September 2023.

Medley comes equipped with a core set of functions for specifying regions and creating the windows that occupy those regions on the screen. But it can be disruptive if not irritating to have to draw out a new ghost region for every invocation of a particular application. Thus the common applications (e.g. TEDIT, SEDIT, DINFO...) implement particular strategies to reduce the number of times that a user has to sweep out a new region. They instead default to regions that were allocated for earlier invocations that are no longer active. TEDIT for example recycles the region of a session that was recently shut down, SEDIT allocates from a list of previous regions, DINFO always uses the same region, but FILEBROWSER always prompts for a new one. Applications that do recycle their regions tend to do so indiscriminately, without regard to the current arrangement of other windows on the screen or the role that those windows may play in higher-level applications.

The REGIONMANAGER package provides simple extensions to the core region and window functions. These are aimed at giving users and application implementors more flexible and systematic control over the specification and reuse of screen regions. It introduces three new notions:

A "typed region" allows the regions of particular applications to be specified, classified, and recycled according to their types.

The size, location, and orientation of a "relative region" is specified with respect to particular screen points and the location of other windows.

A "constellation region" encloses the collection of satellite windows (prompts, menus, etc) that surround the central window of an application.

REGIONMANAGER is innocuous in that explicit user action is required to change the default behavior of any system components.

### Typed regions

REGIONMANAGER adds overlay veneers to the core `CREATEW`, `CLOSEW`, and `GETREGION` functions to make it easier to predict and control how different applications arrange their windows on the screen without always needing to respond to a ghost-region prompt.

The `REGION/INITREGION` arguments may now be region-type atoms in addition to either `NIL` or particular regions as `CREATEW` and `GETREGION` otherwise allow. The type-atom will resolve to a region drawn from a predefined pool of regions associated with that type, if the pool has at least one that is not currently allocated to another window. If the pool has no available regions, then the pool will be enlarged with a region that the user produces from a normal ghost-region prompt, and the type-atom will then resolve to the newly installed region.

A typed-region is marked as "inuse" and therefore unavailable when `CREATEW` assigns it to a window, and the extended `CLOSEW` marks it as again available when the window is closed. The region of the most recently closed window will be offered the next time a region of its type is requested.

An example of how an application can take advantage of this facility is the TEDIT-PF-SEE package. This provides lightweight alternatives to the `PF` and `SEE` commands that print their output to scrollable read-only Tedit windows, specifying `PF-TEDIT` and `SEE-TEDIT` as their region types. The user can predefine a preference-ordered sequence of recyclable regions that bring up multiple output windows in a predictable tiled arrangement, without region-prompting for each invocation.

The global variable `TYPED-REGIONS` is an alist that maintains the relationship between atomic type-names and the list of regions that belong to each type. The list is ordered according to preferences set by the user, and a type-atom is always resolved to the first unused region in its list. If the user is asked to sweep out a new region, that region is added at the end, as the least preferable. The function `SET-TYPED-REGIONS` is provided to add or replace `TYPED-REGION` entries.

```
(SET-TYPED-REGIONS TYPELISTS REPLACE)
```

[Function]

`TYPELISTS` is an alist of the form

```
((type . regions) (type . regions) ...)
```

where each `regions` is a possibly empty list of regions. For convenience, if `TYPELISTS` is just a literal type-atom, it is interpreted as `((type))`, and if it is a list `(type . regions)` beginning with an atom, it is interpreted as `((type . regions))`. The new regions replace preexisting regions if `REPLACE`, otherwise they are added at the front.

Typically, a call to `SET-TYPED-REGIONS` would be placed in a user's INIT file to set up the preference order for the regions that the user wants to participate in this reallocation scheme. If an application uses a type that is not on `TYPED-REGIONS`, then that type-atom is treated as `NIL` and always gives rise to the normal ghost-region prompting. Thus a user will observe no change in system behavior if `TYPED-REGIONS` is left with its initial value `NIL`. A type that is added with an empty region list (as opposed to not being on the list at all) will allow new regions to accumulate for recycling.

The function `REGION-TYPE` returns `NIL` if `X` is not a typed-region or not a region of type `TYPE`.

```
(REGION-TYPE X TYPE)
```

[Function]

In most scenarios the interpretation of a typed region specification is handled automatically by the extended `CREATEW` and `GETREGION` functions. Sometimes it may be useful to perform to for the regions dimensions to be entered into other calculations before it is installed in a window. The function `GRAB-TYPED-REGION` recycles an existing `REGION-TYPE` window if one meets the optional minimum width and height requirements, otherwise a new region is returned.

```
(GRAB-TYPED-REGION REGION-TYPE MINWIDTH MINHEIGHT)
```

[Function]

A type can be assigned to an untyped region and installed in a window by the function `REGISTER-TYPED-REGION`. That region will then be recycled when the window is closed.

```
(REGISTER-TYPED-REGION REGION REGION-TYPE WINDOW)
```

[Function]

If `REGION` is `NIL`, the (presumably) untyped region of `WINDOW` will be registered. An entry in `TYPED-REGIONS` will be created for `REGION-TYPE` if it is not already present.

## Relative regions

Two functions are provided to make it easy to create regions relative and oriented with respect to a specified reference point. These may be useful for constructing an application that includes a constellation of windows arranged in a particular relative way.

```
(RELCREATERECTION WIDTH HEIGHT CORNERX CORNERY REFX REFY ONSCREEN)
```

[Function]

`RELCREATEREGION` creates a region of dimensions `WIDTH` and `HEIGHT`. One of its corners is identified by `CORNERX` and `CORNERY` and that corner will be aligned with a reference screen-point determined by `REFX` and `REFY`. If `ONSCREEN`, the `WIDTH` or `HEIGHT` will be adjusted with respect to that alignment so that the resulting region is entirely within the screen.

`WIDTH` and `HEIGHT` can be given as absolute (natural) numbers or specified relative to the `WIDTH` and `HEIGHT` of another region or of the screen. The possibilities are interpreted as follows:

natural number: the number of screen points

list of the form (anchor fraction adjustment), where anchor is a region, window, or an atom `SCREEN` or `TTY`. The corresponding dimension of the anchor is multiplied by `fraction` and `adjustment` is added to the result. For example, specifying `(<window> .5 -1)` results in a `WIDTH` that is one point smaller than half the width of window's region. `Fraction` and `adjustment` default to 1 and 0 respectively.

region/window/SCREEN/TTY: equivalent to (region/window/SCREEN/TTY 1 0).

`CORNERX` can be `LEFT`, `RIGHT`, or `NIL=LEFT`, `CORNERY` can be `BOTTOM`, `TOP`, or `NIL=BOTTOM`. If `LEFT/TOP` are specified, for example, the region will be displayed down and to the right of the reference point. If `RIGHT/BOTTOM`, then up and to the left.

The reference-point arguments `REFX` and `REFY` are interpreted as follows:

`NIL`: `LASTMOUSEX/LASTMOUSEY`

natural number: an absolute screen coordinate

(anchor fraction adjustment) or just region/window/SCREEN/TTY: the quantity determined relative to the size of anchor (as above) is added to the anchors left/bottom produce the `REFX/REFY` coordinate. In this case, fractions specified as `LEFT/BOTTOM/NIL` are interpreted as 0 and `RIGHT/TOP` are interpreted as 1. For example, a specification `(<window> .4 -2)` for `REFY` will produce a coordinate 2 points below the level that is 40% of the distance between the bottom and top of the window's region.

For convenience, if `REFX` is a position and `REFY` is `NIL`, then the `XCOORD` and `YCOORD` of `REFX` are taken as absolute values for `REFX` and `REFY`.

Also for convenience, if `WIDTH` is a potentially a list of `RELCREATEREGION` arguments, then the elements of that list are spread out in a recursive call.

(RELGETREGION WIDTH HEIGHT CORNERX CORNERY REFX REFY MINSIZE)

[Function]

Calls `GETREGION` with an initial ghost region as created by `RELCREATEREGION`. `CORNERX` and `CORNERY` determine the ghost region's fixed corner, and the cursor starts at the region's diagonally opposite corner. If `MINSIZE` is true, then `WIDTH` and `HEIGHT` are taken as the minimum sizes of the region, except for adjustments that may be needed to ensure that all corners of the ghost region are initially visible on the screen.

(RELCREATEPOSITION REFX REFY)

[Function]

Creates a position with X and Y coordinates specified by `REFX` and `REFY` references as above.

## Constellation regions

Applications are often set up as a constellation of windows, a central or primary window surrounded by some number of satellites for menus, headers, prompts, and secondary outputs. The main panel of a file browser, for example, displays the list of files, but above it are carefully arranged windows for the column headers, summary information, and prompts, and off to the side is the menu of file browser commands. FILEBROWSER interprets the screen region that the user sweeps out for a new browser as the region for the whole constellation, the smallest region that will enclose the central window and all of its satellites. Similarly, the screen region given to TEDIT and SEDIT is divided between the prompt window and the central editing window, again so that the whole constellation (a pair in these cases) fit within the provided region.

Each of these applications is constructed by anticipating the subregions that the satellite windows will occupy after they are attached, decreasing the constellation region by their estimated (using WIDTHIFWINDOW HEIGHTIFWINDOW) or actual sizes, and then using remainder as the region for the central window.

An alternative approach is to construct the central window first, giving it the entire constellation region, and then to have ATTACHWINDOW reshape that window to accomodate the satellite windows as they are attached in sequence. This leads to the same final configuration, but there is no need for separate calculations to pre-adjust the region of the central window.

REGIONMANAGER provides an overlay veneer for ATTACHWINDOW that implements this strategy. If the new argument TAKEFROMCENTRAL is true, then the region of the WINDOWTOATTACH will be subtracted from the region of the existing central window according to the EDGE parameter of the attachment.

```
(ATTACHWINDOW WINDOWTOATTACH MAINWINDOW EDGE POSITIONONEDGE WINDOWCOMACTION TAKEFROMCENTRAL) [Function]
```

This behavior is also triggered if the UNDERCONSTRUCTION property of the central window is true. Thus, a constellation can be set up by creating all of the satellites and the central window, marking the central window as under construction, and then doing the sequence of attachments. The property can be reset to NIL when the construction is complete, so the central window does not shrink if other windows are attached (e.g. expanded menus) by later user actions.

A somewhat weaker form of a constellation is a collection of windows that are not attached around a central window but stand in a parent-child relationship at least with respect to closing and moving. A parent windows spawns children that respond independently to ordinary window commands (move, shape, close). But the children close when the parent closes, and the children move when the parent moves so that they continue to appear in the same relative positions. These primitives allow the construction of a tree of windows that are dependent in this way.

```
(CLOSEWITH CHILDREN PARENT) [Function]
```

Establishes a link between the PARENT window and any number of CHILDREN windows such that all CHILDREN will close when PARENT closes. The closing is accomplished by CLOSEWITH.DOIT:

```
(CLOSEWITH.DOIT PARENT) [Function]
```

Closes the close-with children of PARENT.

```
(MOVEWITH CHILDREN PARENT) [Function]
```

Establishes a link between the PARENT window and any number of CHILDREN windows such that all CHILDREN will move when PARENT moves. The closing is accomplished by MOVEWITH.DOIT:

```
(MOVEWITH.DOIT PARENT NEWPOS) [Function]
```

If `NEWPOS` is the new position of `PARENT`, moves each of the move-children so that they stand in the same relation to `PARENT` after it moves as before.

---



---

## REMOTEPSW

---



---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: COURIERSERVE, COURIERDEFS

REMOTEPSW defines a *remote* process status window facility that runs on top of Courier. The remote process status window is identical to the local one except that it contains **UPDATE** (to get the current process status) instead of **BREAK**, and **INFO** is not implemented. Both the client and server code are contained in the module which must be loaded on both hosts. The Courier server must be running on the host you wish to monitor.

The only user function is:

(REMOTE.PROCESS.STATUS.WINDOW *HOST*)

[Function]

which opens a remote process status window onto *HOST*, where *HOST* is any NS host specification that COURIER.OPEN accepts.

```

221 # 0.52612.100127 # 0
COURIER221#0.52612.100127#137624
COURIER.LISTENER+124745
SPP#221#0.52612.100127#5
COURIER.LISTENER
LAFITEMAILWATCH
ERIS#LEAF
\3MBWATCHER
MOUSE
TEdit#2
TEdit
EXEC
\NSGATELISTENER
\PUPGATELISTENER
\TIMER.PROCESS
BACKGROUND

```

|      |        |         |
|------|--------|---------|
| BT   | WHO?   | KILL    |
| BTV  | KBD←   | RESTART |
| BTV* | INFO   | WAKE    |
| BTV! | UPDATE | SUSPEND |

---



---

## RPC

### SUN REMOTE PROCEDURE CALLS

---



---

By: JFinger

Supported by Atty Mullins (Mullins.pa@Xerox.com) and Bill Van Melle (vanMelle.pa@Xerox.com).

This document last edited on August 1, 1988.

#### INTRODUCTION

This module implements SUN remote procedure calls as specified in the Remote Procedure Call Protocol Specification. The syntax is oriented toward Lisp users, differing greatly from Sun's C-like syntax.

#### RPC2 Package

All functions and variables mentioned in this document are defined as external variables in the package RPC2, unless otherwise stated.

#### REMOTE PROCEDURE DEFINITION

Remote programs are defined via calls to define-remote-program.

|                       |                                                                                                                                                                                                                                                                               |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| define-remote-program | name number version protocol &key :constants [Function]<br>:types :inherits :procedures                                                                                                                                                                                       |
|                       | Defines parameters and result types of the procedures of remote program (number, version, protocol) . If successful, returns name, otherwise nil.                                                                                                                             |
| name                  | a string or symbol that may be used by other procedures (for example, remote-procedure-call) to uniquely specify this remote program.                                                                                                                                         |
| number                | is the program number of this program on the remote machine. As specified in Sun's Remote Procedure Call Programming Guide, programs 0 - #x1ffffff are defined by Sun, #x20000000 - #x3ffffff are reserved for users, and #x40000000 - #x5ffffff are designated as transient. |
| version               | a number, is the desired version of remote program.                                                                                                                                                                                                                           |
| protocol              | an atom,UDP or TCP.( At the moment TCP is not supported under Medley 1.0-S).                                                                                                                                                                                                  |
| constants             | a list of pairs (<constant-name> <constant-def>), where <constant-name> is a symbol and a <constant-def> is an XDR constant (See XDR Constant Definitions below) .                                                                                                            |
| inherits              | a list of name 's of other remote programs from which types and constants are inherited. Inherited types and constants are resolved by searching this list in order.                                                                                                          |
| types                 | a list of pairs (<type-name> <typedef>) , where a <type-name> is a symbol and a <typedef> is an XDR type definition (defined below).                                                                                                                                          |



procedures

a list of 4-tuples of the form (<procname> <procnumber> <arg-types> <result-types>), where <procname> is a symbol or string naming the procedure, <procnumber> is the procedure number on the remote machine, <arg-types> is a (possibly empty) list of XDR type definitions (see below) of the arguments to be sent to the remote procedure, and <result-types> is a (possibly empty) list of XDR type definitions of data to be returned from this remote procedure.

## XDR (EXTERNAL DATA REPRESENTATION) TYPE DEFINITIONS

Because the client and server machines may represent data in different ways, a data representation common to both machines is necessary. Remote procedure calls pass data between machines in 'External Data Representation' (XDR). The XDR language implemented here is oriented toward Lisp in its syntax and is not identical to the language spelled out in the Sun XDR Protocol Specification.

XDR data types may be defined in the `:types` keyword argument for later reference in the `:types` or `:procedures` of this or later remote programs. When a remote program is defined (usually at load time), the needed reading and writing functions are compiled for each constructed type referenced. Note that all XDR calls are eventually resolved to a composition of Primitive and Constructed XDR Type Definitions (see below).

## SYNTAX

The keywords of the XDR language may be specified as symbols of the `Keyword` package.

All XDR Data Types Definitions (notated here as a <typedef>), used in Remote Procedure Calls are from the following language:

- |                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1) <b>Primitive Definition:</b>   | <p>One of the types in *xdr-primitive-types*,</p> <pre> :integer :boolean :unsigned :hyperinteger :hyperunsigned :string :float (not yet implemented) :double (not yet implemented).</pre>                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 2) <b>Constructed Definition:</b> | <p>One of the types in *xdr-constructed-types*,</p> <pre> (:enumeration          (&lt;symbol-1&gt; &lt;constant-1&gt;) ...                         (&lt;symbol-n&gt; &lt;constant-n&gt;)) (:union &lt;enumeration-type&gt; &lt;typedef-1&gt; ... &lt;typedef-n&gt;) (:fixed-array &lt;typedef&gt; &lt;constant&gt;) (:counted-array &lt;typedef&gt;) (:opaque &lt;constant&gt;) (:struct &lt;defstruct-type&gt; (&lt;field-name-1&gt; &lt;typedef-1&gt;) ...                         (&lt;field-name n&gt; &lt;typedef-n&gt;)) (:sequence &lt;typedef&gt;) (:list &lt;typedef-1&gt; ... &lt;typedef-n&gt;).</pre> |
| 3) <b>Local Definition:</b>       | <p>A symbol defined previously in the same remote program definition.</p> <p>Example:           :types ((nrec :unsigned)...) says that type 'nrec' is really only of type ':unsigned'.</p>                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 4) <b>Qualified Definition:</b>   | <p>A dotted pair of the form (&lt;RPC program name&gt; . &lt;type&gt;), where &lt;type&gt; is an XDR type local to &lt;RPC program name&gt;.</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

Example: `:types ((count (myprog . nrec))...)` says that a 'count' is really whatever myprog defines a 'nrec' to be.

#### 5) Inherited Definition:

A symbol defined in the `:types` argument of a remote program R such that R is on the list of remote programs passed as the `:inherits` argument to the current remote program definition. The first such type definition found is used, that is, the list of inherited programs is scanned from left to right.

### XDR CONSTANT DEFINITIONS

Constants in XDR are defined by the following grammar:

**<constant-def> ::= <integer> | <defined-constant>**

**<defined constant> ::=**

**<locally defined constant>**

; Defined in the Remote Program currently being defined.

**| <inherited constant>**

; Defined in a remote program inherited by the current Remote Program (searched from left to right).

**| <qualified constant>**

; A dotted pair (`<rp> . <constant>`), where `<constant>` is defined in remote program `<rp>`.

### SEMANTICS

An XDR type can be defined by a bidirectional filter mapping a subset of Lisp onto a byte stream and vice-versa.

For the XDR primitive type's filter, a description is given of its argument on the Lisp and XDR sides.

|                              |                                                                                                                                                                                                                                                      |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>:integer</code>        | Lisp: an integer in range -2,147,483,648 to 2,147,483,648 inclusive.<br>XDR: a 4 byte two's complement integer, high order to low order.                                                                                                             |
| <code>:unsigned</code>       | Lisp: an integer in range 0 to 4,294,967,295 inclusive.<br>XDR: a 4 byte non-negative integer, high order to low order.                                                                                                                              |
| <code>:boolean</code>        | Lisp: NIL for false, non-NIL for true. (The Lisp symbol T is returned when decoding a 1 from the XDR side.)<br>XDR: 0 for false, 1 for true.                                                                                                         |
| <code>:hyperinteger</code>   | Lisp: an integer in range -(263) to 263 -1 inclusive.<br>XDR: a 8 byte two's complement integer, high order to low order.                                                                                                                            |
| <code>:hyperunsigned</code>  | Lisp: an integer in range 0 to 264-1 inclusive.<br>XDR: a 8 byte non-negative integer, high order to low order.                                                                                                                                      |
| <code>:string</code>         | Lisp: a string of any length.<br>XDR: Suppose the string is of length n. The XDR representation is an <code>:unsigned</code> (the string length n), followed by the n bytes of the string, followed by enough 0 bytes to make a multiple of 4 bytes. |
| <code>:string-pointer</code> | (UDP only)<br>Lisp: a dotted pair ( <code>addr . nbytes</code> ), where <code>addr</code> is a buffer's address and <code>nbytes</code> is the number of bytes in the buffer. (Should I add an offset                                                |

argument?). This is a speed hack to avoid having to copy VMEMPAGEP's twice.

XDR: An XDR :string, as above.

:float

Lisp: A floating point number. (NOT YET IMPLEMENTED).

XDR: A 4 byte floating point number in IEEE format.

:double

Lisp: A floating point number. (NOT YET IMPLEMENTED).

XDR: A double precision floating point number in IEEE format.

:void

Lisp: null

XDR: no bytes.

For each constructed XDR type, the declaration syntax is given along with its corresponding mapping.

(:enumeration (<symbol> <integer>) ... (<symbol> <integer>))

Lisp: a symbol

XDR: an XDR :integer.

The Lisp symbol (Each symbol is the "discriminant" for that value of the enumeration) and the XDR integer will be from a corresponding pair in the declaration. It is an error to try to encode a symbol not in the declaration or to try to decode an XDR integer for which there is not a corresponding symbol in the declaration.

(:union <enumeration-type> (<symbol-1> <typedef-1>) ... (<symbol-n> <typedef-n>))

Lisp: A list of two elements, the first being a discriminant for the enumeration type, and the second the appropriate Lisp input/output for the typedef corresponding to that discriminant's type..

XDR: An :integer discriminant followed by the XDR input/output for the typedef corresponding to that discriminant's type.

(:fixed-array <typedef> <constant>)

Lisp: An array of length <constant>, each element of which is an object of type <typedefLisp>. Note that since the function elt is used in encoding, any Lisp sequence could be used in place of an array.

XDR: A sequence of <constant> objects of type <typedefXDR>.

(:counted-array <typedef>)

Lisp: A list of two elements, the first of which is an integer (the number of objects to be encoded/decoded), and the second of which is an array of objects of type <typedefLisp>.

XDR: An integer (the number of objects to be encoded/decoded) followed by that number of objects of type <typedefXDR>.

(:opaque <constant>)

Lisp: A string of length <constant>.

XDR: A sequence of <constant> bytes followed by enough null bytes to round <constant> up to a multiple of four.

(:struct <defstruct-type> (<field-name-1> <typedef-1>) ...(<field-name n> <typedef-n>))

Lisp: A struct of type <defstruct-type> such that each field mentioned in the this XDR declaration has a value. Note that a separate defstruct must be executed. The fields need not be named here in the same order as those in the defstruct, nor must all the fields named in the defstruct be used here.

XDR: A sequence of objects of types <typedef1 XDR>...<typedefn XDR>.

(:sequence <typedef>) This is fashioned after Courier's method for encoding/decoding linked lists. This type can often be used to get around clumsy recursive definitions involving :union's of enumeration type :boolean.

Lisp: A list of objects of type <typedefLisp>.

XDR: A sequence of objects, each preceded by an XDR :boolean encoding of true. The last object in the sequence is followed by the XDR :boolean encoding of false.

Note: (:sequence <typedef>) produces the same encoding (but not the same decoding) as  
(defstruct astructure this-element the-rest)  
along with the declaration

```
(:recursive (:union :boolean
 (T (:struct astructure (this-
 element <typedef>) (the-rest
 astructure)))
 (NIL :void))))
```

(:list <typedef-1> ... <typedef-n>)

Lisp: A list, the ith element of which is of type <typedefi Lisp>.

XDR: A sequence of objects, the ith of which is of type <typedefi XDR>.

(:skip <unsigned>)

(For decoding only)

Lisp: Nothing

XDR: Any n bytes of data, where <unsigned> = n.

Note: This is a kludge for not having to decode the fattr's that NFS returns with every single cotton-pickin' memory read.

## EXAMPLE OF A REMOTE PROGRAM DEFINITION

The following call to define-remote-program defines the portmapper remote procedures described in Sun's Remote Procedure Call Specification. Note that there are two definitions of procedure 4 given. Since remote procedures may be invoked by name, it is reasonable for there to be more than one definition for how to decode and encode the arguments to a given routine. In this case, both a recursive and non-recursive definition is given for the values returned from procedure 4. Note also that mapstruct and mapsequence must be defstruct'ed before this call to define-remote-procedure.

```
(define-remote-program 'portmapper 100000 2 'udp
 :types '((mapstruct (:union :boolean
 (nil :void)
 (t (:struct mapstruct
 (program :unsigned)
 (vers :unsigned)
 (prot :unsigned)
 (port :unsigned)
 (therest mapstruct))))))
 (mapsequence (:sequence (:struct mapsequence
 (program :unsigned)
```

```

:procedures
 '((null 0 nil nil)
 (lookup 3 (:unsigned :unsigned :unsigned :unsigned)
 (:unsigned))
 (gooddump 4 nil (mapsequence))
 (dump 4 nil (mapstruct))
 (indirect 5 (:unsigned :unsigned :unsigned
 :string)
 (:unsigned :string))))

```

## UNDEFINING REMOTE PROGRAMS

undefine-remote-program    name number version    [Function]

## MAKING REMOTE PROCEDURE CALLS

remote-procedure-call    destination program procid arglist    [Function]  
                              &key destsocket version credentials protocol  
                              dynamic-prognum dynamic-version  
                              msec-until-timeout msec-between-tries noerrorflg

Performs a remote procedure call to program on destination. Returns a list of the returned values.

**destination**    Designates the host to which the procedure call is made. If Destination is a number it is interpreted to be the il:iphostaddress of the host; if a symbol or string, it is a name from which the net address of the host may be found.

**program**    Designates the remote program to be called. If Program is a number, it is interpreted to be the remote program number. If a symbol, in which case it is assumed to be the name of the remote procedure (as defined in define-remote-procedure. If :version is non-nil, then program is treated as a number rather than as a name. If version is nil and program is a number, then the latest version of that program is used.

**procid**    Designates the procedure number from program to be called. If Procid is a number it is interpreted to be the remote procedure number; if a symbol, it is the name given that procedure in define-remote-procedure.

**arglist**    A list of the arguments to be serialized into XDR representation and passed as the arguments of the remote procedure call.

**:destsocket**    Normally, the remote socket must be looked up in the local caches (See \*rpc-socket-cache\* and \*rpc-well-known-sockets\*) or else found by making a call to the Portmapper on the remote machine. If :destsocket is non-nil, its value is used as the remote socket.

**:version**    If non-nil designated the desired version of program as well as causing program to be interpreted as a number rather than a name. See program above.

|                     |                                                                                                                                                                                                                                        |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| :credentials        | An object of type authentication to be passed as the credentials of the remote procedure call. (See create-unix-authentication).                                                                                                       |
| :protocol           | A symbol specifying the transport protocol. Currently only UDP is implemented. Defaults to UDP. The only reason for using this parameter is to specify (along with the program and version), which known remote program is to be used. |
| :dynamic-prognum    | If you really can't live without it, dynamic-prognum is used as the remote program number in spite of treating the arglist and returned values exactly as in program. Don't ask why.                                                   |
| :dynamic-version    | If you really can't live without it, dynamic-version is used as the remote program version in spite of treating the arglist and returned values exactly as specified in program (and possibly version). Don't ask why. Defaults to 1.  |
| :msec-until-timeout | Total number of milliseconds of waiting for a reply packet before giving up on this remote procedure call. Defaults to value of *rpc-msec-until-timeout*.                                                                              |
| :msec-between-tries | Number of milliseconds between outgoing UDP packets. Defaults to *rpc-msec-between-tries*.                                                                                                                                             |
| :errorflg           | If :noerrors, ignores remote procedure call errors. If :returnerrors, returns the error as an s-expression. Otherwise, signals a Lisp error. Default t.                                                                                |

## LOW-LEVEL REMOTE PROCEDURE CALL FUNCTIONS

|                  |                                                                                                                                                                                                                                                                                                                    |            |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| setup-rpc        | destination program procid<br>&optional destsocket version protocol dynamic-prognum dynamic-version                                                                                                                                                                                                                | [Function] |
|                  | Returns four values destaddr, socket, program and procedure (Yes, this is real, live multiple value return requiring a multiple-value-bind or something similar.) for consumption by perform-rpc. The arguments to setup-rpc are identical in meaning to the identically named arguments to remote-procedure-call. |            |
| open-rpc-stream  | protocol destaddr destsocket                                                                                                                                                                                                                                                                                       | [Function] |
|                  | Returns an rpcstream for use by perform-rpc. Destaddr and destsocket are as returned by setup-rpc and protocol is identical to the protocol argument to remote-procedure-call.                                                                                                                                     |            |
| close-rpc-stream | rpcstream protocol                                                                                                                                                                                                                                                                                                 | [Function] |
|                  | Closes rpcstream, an rpc-stream of protocol protocol created by open-rpc-stream.                                                                                                                                                                                                                                   |            |
| perform-rpc      | destaddr destsocket program procedure rpcstream<br>arglist credentials protocol &key errorflg leave-stream-open msec-until-timeout msec-between-tries                                                                                                                                                              | [Function] |
|                  | Performs a remote procedure call returning a list of the values retruned by the remote procedure.                                                                                                                                                                                                                  |            |

**LISTING REMOTE PROGRAMS CURRENTLY DEFINED**

list-remote-programs [Function]

Returns a list of 4-tuples (name number version protocol) for each remote program currently defined.

**CREATION OF CREDENTIALS**

create-unix-authentication stamp machine-name uid gid gids [Function]

Returns a Unix-type authentication suitable for use as the credentials of a call to remote-procedure-call or perform-rpc.

stamp An arbitrary unsigned integer.

machine-name A string containing the name of the calling machine.

uid User id number on the remote machine.

gid Group id number on the machine.

gids A list or array of group id numbers (on the remote machine) that contain the caller as a member.

**GLOBAL VARIABLES**

\*xdr-primitive-types\* An a-list of keywords and the corresponding function that implements that XDR primitive type.

\*xdr-constructed-types\* An a-list of keywords and the corresponding function that generates code to implement that XDR constructed type.

\*msec-until-timeout\* Number of milliseconds before giving up on receiving a reply packet. Default 1000.

\*msec-between-tries\* Number of milliseconds to wait before resending UDP packet. Default 100.

\*rpc-ok-to-cache\* If non-nil, uses \*rpc-socket-cache\* as a cache of socket numbers found to date.

\*rpc-well-known-sockets\* A list of well-known sockets. Format is  
( <host address>  
  <remote program number>  
  <remote program version>  
  <protocol>  
  <socket> )

\*rpc-socket-cache\* A list of non-well-known sockets. Format is same as \*rpc-well-known-sockets\*.

\*debug\* If non-nil prints out debugging information. If a number, the higher the number, the more information is printed. Default nil.

**RPC FILES**

|                      |                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------|
| <b>RPC</b>           | Sets up the RPC2 Package and loads other RPC files. Loads Portmapper remote program definition and executes it.   |
| <b>RPCLOWLEVEL</b>   | Super low-level UDP/TCP functions added to Eric Schoen's TCPUDP code.                                             |
| <b>RPCOS</b>         | Low-level interface to Sun OS networking code .                                                                   |
| <b>RPCSTRUCT</b>     | Structure definitions used by the other files. These are in a separate file because they take so long to compile. |
| <b>RPCCOMMON</b>     | Common lookup functions and stream i/o functions used by the other files.                                         |
| <b>RPCXDR</b>        | External Data Representation (XDR). Code Generation for XDR constructed types and XDR primitive functions.        |
| <b>RPCRPC</b>        | Remote program definition and remote procedure calls.                                                             |
| <b>RPCPORTMAPPER</b> | Definition of portmapper in UDP and TCP.                                                                          |

**KNOWN DEFICIENCIES**

Floating point XDR types are not implemented.

The view-packet utility is not documented and needs to be smarter about authentications.

Fall through cases of XDR types UNION and ENUMERATE should be added.

TCP is not supported under Medley 1.0-S, this should be in the next release.

**COPYRIGHT INFORMATION**

Copyright (c) 1987,1988 Leland Stanford Junior University and Envos Corporation.

Written by Jeff Finger under support of National Institutes of Health Grant NIH 5P41 RR00785

to the SUMEX-AIM Computing Resource at Stanford University.

Modified to work under Medley 1.0-S by Atty Mullins.



---

---

**RS232CNetwork**

---

---

By: Nick Briggs (Briggs.pa@xerox.com)

Uses: DLRS232C

This document last edited on August 10, 1988.

## INTRODUCTION

The RS232C port on a Daybreak can be configured in such a way that it can be used to communicate with a Pup Gateway over a phone line to provide a network communications path equivalent in all but speed to an Ethernet connection.

## USE

Load the RS232CNetwork module. It redefines a number of functions, the one of general interest being

(RS232C.INIT *BAUDRATE BITSPERSERIALCHAR PARITY NOOFSTOPBITS*  
*FLOWCONTROL LINETYPE*) [Function]

RS232C.INIT performs as documented in the Lisp Library modules manual, except for the addition of the *LINETYPE* argument. *LINETYPE* should be SYNC or ASYNC (also accepted are the alternate spellings SYNCH and ASYNCH).

In order to use the RS232C port for communicating with a Pup Gateway it must be configured (using RS232C.INIT) in SYNC mode. In addition, the variable

\*RS232C-NETWORK\* [Variable]

should be set to T (which is the default setting on loading the RS232CNetwork module.

The variable

\*RS232C-NETWORK-AUTODIAL\* [Variable]

controls whether the system will attempt to start the network connection automatically when Lisp returns from a LOGOUT. If \*RS232C-NETWORK-AUTODIAL\* is set to NIL, you can cause the system to attempt to establish the connection manually by calling the function RESTART.ETHER.

The modem should be configured to dial on detecting an off to on transition of DTR. On a Codex 2260 modem this is DTR mode 108.1. Since the dialing and answering process can take a non-trivial amount of time, the variable

\*RS232C-NETWORK-DIALING-TIMEOUT\* [Variable]

indicates how many seconds the code will wait for the modem to signal that the connection has been established (by raising DSR) before timing out and continuing without the network connection. The default value is 30 seconds.

## EXAMPLE

The very first time the code has been loaded the initialization sequence, when using Codex 2260 modems which can communicate at 9600 baud, would be

```
(RS232C.INIT 9600 8 'NONE 1 NIL 'SYNC)

(SETQ *RS232C-NETWORK-AUTODIAL* T)

(RESTART.ETHER)
```

Subsequently, whenever the Daybreak boots into Lisp, the network code will attempt to establish a phone connection.

To shutdown the phone connection you can call the function TURN.OFF.ETHER. The connection will also be broken when you log out of Lisp (assuming that the modem is configured to disconnect on loss of the DTR signal, which the Codex 2260 modems will do)

---



---

## SCREENPAPER

---



---

By: Larry Masinter (Masinter.pa@Xerox.com)

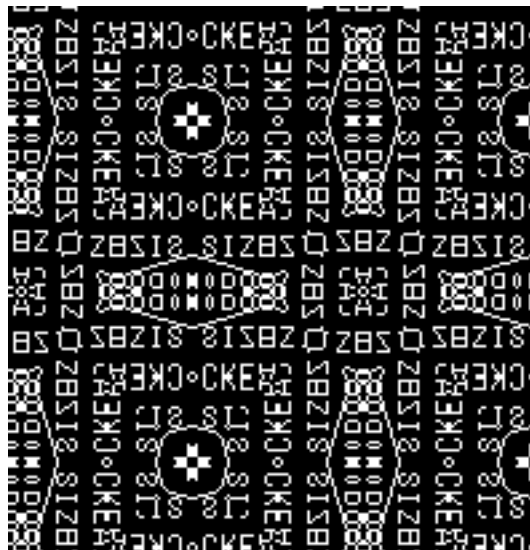
SCREENPAPER is an Idle hack ("Screen wallpaper"). Old fashioned wallpaper/wrapping paper from your screen.

### Global parameters:

SCREENPAPERSIZE size of viewport, initially 64.

SCREENPERIOD how often to go into reflective move

SCREENREPEAT how long to stay in reflective mode (initially 0. e.g., disable reflective mode).



---

---

## SEARCHMENU

---

---

By: John Maxwell (Maxwell.pa)

Uses: DICTTOOL, DICTCLIENT, ANALYZER

INTERNAL

This document last edited on October 16, 1987

### INTRODUCTION

The SearchMenu package implements a user interface to the relevance search capabilities of the dictionary server. Relevance search is a technique for finding items by giving examples of what you are looking for and having the search algorithm look for "similar" items. These items are then displayed in a FreeMenu which allows you to select those items which are actually relevant. You can then iterate on the new set of examples until no new relevant items show up.

The Dictionary Server currently supports three databases that can be searched using this technique: the WordNerd (based on the American Heritage Dictionary), the EtymologyNerd (based on the etymological portion of the American Heritage Dictionary), and the IRMNerd (based on the Interlisp Reference Manual). For example, if you were looking for different types of gases, setting the database to the WordNerd and giving it example gases such as hydrogen and helium would produce oxygen, xenon, krypton, argon, neon, radon, fluorine, chlorine, and nitrogen. Or, if you were looking for Interlisp functions that tested the equality of something, setting the database to the IRMNerd and giving example procedures like EQ and EQUAL would produce EQP, IEQP, FEQP, STREQUAL, EQMEMB and EQUALALL.

### HOW TO USE THE SEARCH MENU

When you load the SearchMenu package, a Search Menu will appear in the lower left corner of the screen. (The Search Menu expands to display the results of a search, so you might leave it there until you see how big it can get.) The Search Menu has a number of commands on the top line, followed by a place to type in examples.

To try your first search, click the "Examples:" field and type "lion tiger". Then click the MATCH WORDS! command just above it. A little menu of databases will appear, asking you which database you want to search in. Select "DictServer: WordNerd". A message will then appear in the prompt window saying: "Searching in DictServer: WordNerd for words like: (tiger lion)." After about 30

seconds a new Search Menu will be created with a list of items that the Dictionary Server thinks is similar to a lion and a tiger. At the top of the list lion and tiger will already be selected.

To continue the search, scan the list of items for things that are relevant to you. If you are not sure what a word means, click the "def" button to its left, and the definition will be printed out in a separate window. If you want an item to be included in the next search, simply click at it and it will become inverted, like lion and tiger. If you click at an item twice, a line will be drawn through it to indicate that any keywords that it uses should have their weights reduced. Words that are neither highlighted nor struck out are ignored. When you are through examining the items, invoke the MATCH WORDS! command. You can iterate like this as often as you like.

### **Clearing The Search Menu**

Before you start a new search, you should invoke the "CLEAR!" command to clear the Search Menu. The Search Menu caches some information about the search you are conducting which may interfere with your next search. Therefore, to be sure that you get a clean search, you should clear the menu.

### **Changing Databases**

When you want to search for items in a new database, all that you need to do is click the "SET DATABASE!" command. This will cause a menu of databases to appear. Clicking one of the items in the menu will cause the Search Menu to search in that database from then on. Clicking outside the menu leaves the Search Menu in its current state.

### **The Key Menu**

To the left of the commands in the first line is a command labelled "KEY MENU!". Invoking this command will turn the Search Menu into a key menu. The key menu will have all of the keys that were used in the last search, along with their weights. The weights are editable. If you want to see what would happen if you searched using different weights, simply edit the weights and then invoke the "MATCH KEYS!" command. Weights that are set to 0 or have no value are ignored.

If you want to see all of the uses of a particular key click the "uses" button to the left of that key. To get back to the example menu, invoke the "SAMPLE MENU!" command in the upper left of the Search Menu.

### **Logging a Search**

If you wish to keep a log of each search, set the SearchMenu.LogData variable to T. From then on, the Search Menu will write the results of its search into a private stream. When you click the "CLEAR!" command, a TEdit window will be opened on the log and the log cleared in preparation for the next search.

## **THE PROGRAMMER'S INTERFACE**

The Dictionary Server can be accessed directly through the following procedures:

(DICTCLIENT.MATCHWORDS *POSWORDS NEGWORDS MINWORD MAXWORD DICTIONARY*)[Function]

Takes a list of words to match, plus a list of words to ignore, plus the range of words that you are interested in, plus the database. The range of words lets you look at a search one chunk at a time: the first fifty (MINWORD = 1, MAXWORD = 50), then the next fifty (MINWORD = 51, MAXWORD = 100), and so on. The DICTIONARY is the name of the database to be searched in. Currently there are three possibilities: 'WordNerd, 'EtymologyNerd, and 'IRMNerd. If no data base is specified, the default is 'WordNerd.

(DICTCLIENT.WEIGHTEDSEARCH *WEIGHTEDKEYS MINWORD MAXWORD DICTIONARY*)[Function]

Takes a list of weighted keys to match, plus the range of words that you are interested in, plus the database. The weighted keys should be of the form '((key1 weight1)(key2 weight2)...). The range of words lets you look at a search one chunk at a time: the first fifty (MINWORD = 1, MAXWORD = 50), then the next fifty (MINWORD = 51, MAXWORD = 100), and so on. The DICTIONARY is the name of the database to be searched in. Currently there are three possibilities: 'WordNerd, 'EtymologyNerd, and 'IRMNerd. If no data base is specified, the default is 'WordNerd.

---

---

**SEdit-Menu-Always**

---

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

This package advises the SEdit Editor so that when an SEdit window is opened, the SEdit Attached Command Menu is automatically opened as well (depending on the setting of the global variable IL:SEditMenuAlwaysFlg). The value of IL:SEditMenuAlwaysFlg is initialized to T as an INITVAR when the file is loaded.

---



---

## SETDEFAULTPRINTER

---



---

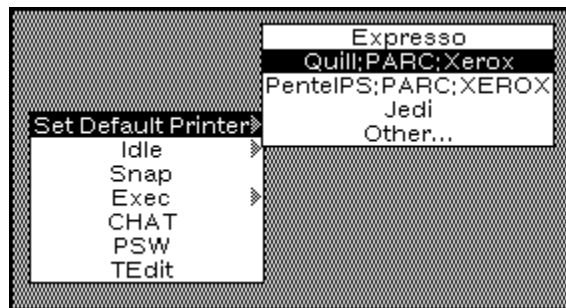
By: Nick Briggs (Briggs.pa@Xerox.com)

The SETDEFAULTPRINTER module provides a (cleaner) mechanism for moving printer names around on your DEFAULTPRINTINGHOST list. There are no user callable functions. Access to the features of the module are through the Background menu. This module uses the DEFAULTSUBITEMFN module which redefines the DEFAULTSUBITEMFN used in menus to accept an expanded form for menu subitems.

### Set Default Printer

[Background Menu Entry]

Selecting the "Set Default Printer" item off the background menu will prompt you for a new default printer, which will be added at the beginning of the DEFAULTPRINTINGHOST list. If you roll-out into the subitems for Set Default Printer it will present a submenu with the entries on DEFAULTPRINTINGHOST, and an "Other..." item. Selecting one of the printer name entries will cause it to be moved to the front of DEFAULTPRINTINGHOST, selecting "Other..." will prompt for the name of a printer in the same manner as selecting the "Set Default Printer" top level item off the background menu. If any commentary information has been supplied (see below) holding the mouse over the printer name will display the information in the prompt window.



### SDP.PRINTERINFO

[Variable]

The variable SDP.PRINTERINFO is an A-list which will be used to lookup commentary information about a printer to be included as the "help" in the menu subitems. The UPPERCASE name of the printer is used as a key. An example SDP.PRINTERINFO setting might be

```
((QUAKE . "Press, Rm 1532") (PENTELPS:PARC:XEROX . "Interpress, Rm 1532"))
```

### LOCATION

[Property]

The code that looks up the commentary information about a printer will also check for a LOCATION property on the UPPERCASE atom which is the printrname if no entry is found on SDP.PRINTERINFO. For example

```
(PUTPROP 'JEDI 'LOCATION "FullPress, Pod 5, 2nd floor")
```

Would describe the location of printer Jedi.



---



---

## SHOWTIME

---



---

By: Timothy Bigham (TBigham.henr@Xerox.com)  
 Medley mods by: Ron Fischer (Fischer.PA@Xerox.com)

Uses: BITMAPFNS, SCALEBITMAP, READBRUSH

This document last edited on May 13, 1988.

### INTRODUCTION

SHOWTIME provides a user interface to read, write, and edit bitmaps in several different formats. Among the supported formats is RES, used in VIEWPOINT Freehand Graphics. Other supported formats include Brush (Mesa Doodle format); and Lisp.

SHOWTIME has been written to readily accomodate new formats. Users may add new formats to Showtime by writing format-specific read and/or write functions and adding them to those Showtime knows about (described below).

Selecting SHOWTIME from the background will provide the user the opportunity to specify an area to use as the SHOWTIME window. After the user creates a SHOWTIME window, a left mouse button within the window will popup a menu of available options. There may only be one bitmap displayed in a SHOWTIME window at a time, but any number of SHOWTIME windows may be opened. Shrinking a SHOWTIME window will create an icon with the name of the bitmap that is displayed in the window.

### Functions, Variables, and Lisp Code Examples

SHOWTIME.FORMAT.FNS [Variable]

A global association list that maintains a list of all the formats Showtime knows about and the read and write functions to use with those formats. This variable should be updated by calling the function SHOWTIME.ADD.FORMAT to ensure successful integration of any new bitmap formats.

SHOWTIME.DEFAULT.FORMAT [Variable]

A variable that is initially set to 'LISP. This format uses the binary storage routines found in the lispusers module BITMAPFNS.

(SHOWTIME.ADD.FORMAT *FORMAT* *READFN* *SAVEFN*) [Function]

A function that should be called when the user wants Showtime to know about new bitmap formats. *FORMAT* may be any descriptive atom, such as RES or LISP. *READFN* and *SAVEFN* must be functions that have as the first two arguments FILENAME and BITMAP. In addition, the READFN must return a bitmap.

For example, the READFN code for LISP format is:

```
(LAMBDA (FILENAME)
```

```
(* this function must <1> read a bitmap from a file and <2> return the value
of the bitmap)
```

```
(READBM (OPENFILE FILENAME (QUOTE INPUT))))
```

For example, the WRITEN code for LISP format is:

```
(LAMBDA (FILENAME BITMAP) (* TBigham "30-Dec-86 13:09")
```

```
(* this function must write a bitmap to a file)
```

```
(WRITEBM (OPENFILE FILENAME (QUOTE OUTPUT)) BITMAP))
```

## ACKNOWLEDGEMENTS

SHOWTIME was originally developed to provide a user interface in the exchange of bitmaps between the VIEWPOINT and INTERLISP-D environments. Tom Wall initiated the idea to exchange bitmaps between these environments and was instrumental in developing the code to write RES files. Gary Gocek originally wrote the code to read RES files; Mitch Garnaat wrote the code to write BRUSH files; reading brush files is supported by the lispusers module READBRUSH written by Larry Masinter. The SHOWTIME icon was designed and created by Mary Baecher-Cocca.

---

---

**SIMPLECHAT**

---

---

By: Larry Masinter (Masinter.PA@Xerox.COM)

Uses: TEDIT

This document last edited on Sept. 8, 1988.

**INTRODUCTION**

Like CHAT except that it works in the current window/exec instead of spawning a new window. To exit from TTYCHAT there is an escape character, control-right-bracket (^]). If you type ^], you get prompted for a Chat command. This can be one of Binary, Text, or Close. Normally TTYCHAT translates incoming characters and converts EOL; setting Binary mode disables this. Close will close the connection.

**MODULE EXPLANATIONS**

The CHATSERVER module advises CHAT to use TTYCHAT when the main "terminal" is not the display. This allows one to use the Lisp system as a "protocol translation gateway"; for example, on a Sun with CHATSERVER-NS loaded, you can Chat to the Sun using NS and then use UNIXCHAT to CHAT(SHELL).

(TTYCHAT *&optional host logoption*)

[Function]

---

---

## SNAPW-ICON

---

---

By: Randy Gobbel (Gobbel.pa)

Uses: nothing but basic window functionality

This document last edited on September 8, 1988.

### INTRODUCTION

SNAPW-ICON creates an icon for shrunken screen snap windows. It looks like this:  
Everything else about screen snaps is as before.



---

---

**SOLID-MOVEW**

---

---

By: Lennart Lövstrand  
(Lovstrand.EuroPARC@Xerox.COM)

This document last edited on May 13, 1988.

**INTRODUCTION**

This module changes the behaviour of MOVEW when no destination is given to let the whole image of a window track the mouse instead of just its outline. To avoid flickering and to give an illusion of a smooth animation, all rendering operations are done off the screen, ending with a single bitblt to the frame buffer for each cycle. This can easily be done on small windows such as icons, but the more bits there are to be moved, the longer it takes to do the animation updates and the slower it becomes to solidly move windows. Therefore, the user can control when solid vs. outline moving is to be done by setting *\*SOLID-MOVEW-FLAG\** to an appropriate value. By default, only windows containing less than 15,000 pixels will be moved solidly; all other windows are moved using the original MOVEW method.

SOLID-MOVEW interfaces nicely with both ICONW and ATTACHEDWINDOWS by being able to move images of arbitrary shape — not just pure rectangles. It also knows about GRID-ICONS and can be made to force the icons to snap to grid positions while being moved, thus producing a kind of jagged feeling. Finally, a shadow has been added emphasize the 2½-D property of window systems and to give a clear indication of when the window is in the process of being moved.

**PROGRAMMER'S INTERFACE**

When loaded, the module replaces the system MOVEW function with its own version and moves the original code to ORIGINAL-MOVEW. The control and interaction is then comes through the following variables:

*\*SOLID-MOVEW-FLAG\** [Variable]

This variable controls whether the new MOVEW should use solid or outline moving. It should have one of the following types of values:

- |             |                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------|
| a NUMBERP   | Only use solid moving on windows that have a total size (width x height) less than or equal to the given number of pixels. |
| a POSITIONP | Only use solid moving on windows that have a width and height less than or equal to the two numbers.                       |
| ICON        | Only move icons solidly. A window is considered to be an icon if it either has an ICONFOR or an ICONIMAGE property.        |
| T           | Move all windows solidly.                                                                                                  |
| NIL         | Move all windows using outlines.                                                                                           |

The default value for *\*SOLID-MOVEW-FLAG\** is 15000.

\*SOLID-MOVEW-SHADOW\* [Variable]  
 \*SOLID-MOVEW-SHADOW-SHADE\* [Variable]

These two variables define whether or not a shadow should accompany the moving image. The shadow is always directed towards south-east and the first variable, \*SOLID-MOVEW-SHADOW\*, determines its position by taking on any of the following types of values:

|             |                                                              |
|-------------|--------------------------------------------------------------|
| a NUMBERP   | The x and y offsets of the shadow (same)                     |
| a POSITIONP | The x and y offsets of the shadow (different)                |
| T           | Use the default shadow offset — 3 pixels in both directions. |
| NIL         | Don't show a shadow.                                         |

The second variable, \*SOLID-MOVEW-SHADOW-SHADE\*, sets the darkness of the shadow, ie. the texture to be added to the background where the shadow is visible.

The default values for the two variables are T and 42405, a 50% gray shadow offset by 3 pixels.

\*SOLID-MOVEW-GRIDDING\* [Variable]

When used together with the ICON-GRIDS module, SOLID-MOVEW can be made to only move solid window images on grid positions, thus creating a kind of "jagged" feeling when interactively moving icons on the screen. If this is disabled, the icon will "snap" to the closest grid position only after the move has been completed.

The default value for \*SOLID-MOVEW-GRIDDING\* is NIL, thus disabling early gridding.

\*SOLID-MOVEW-CASHING\* [Variable]

SOLID-MOVEW uses separate bitmaps for rendering purposes so as to produce a smooth animated move and avoid unnecessary flickering on the screen. To speed up the initial phase of the move operation, the rendering bitmaps can be cached from one invocation to another. This will use up some bitmap space, but can be freed using (GAINSPACE) if need arises.

The default value for \*SOLID-MOVEW-CASHING\* is T, thus enabling cached rendering bitmaps.

(SOLID-MOVEW *POSorX* *Y*) [Function]

Because only those windows meeting the requirements of \*SOLID-MOVEW-FLAG\* will be moved solidly, the user has the option of calling SOLID-MOVEW. It takes the same arguments as MOVEW, but if either *POSorX* or *Y* is specified, control is again turned over to the old MOVEW.

If you get tired of all this, you can undo the behaviour of SOLID-MOVEW by typing the following form into an Interlisp Exec:

```
(MOVD 'ORIGINAL-MOVEW 'MOVEW)
```

## BUGS

No provision has been made to make SOLID-MOVEW work with color.

If the window is closed as a side effect of the its MOVEFN or AFTERMOVEFN, it will be reopened before SOLID-MOVEW returns

---

---

## SOLITAIRE

---

---

By: Beau Sheil. Upgraded for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

The SOLITAIRE package is a simple graphics demonstration program that plays and animates the solitaire card game (known as ~Patience~ in English speaking countries). Solitaire is a game for one, so there is no way to play ~against~ the machine. SOLITAIRE is most effective as a background activity when the machine is doing nothing else, so it is frequently used as an IDLE hack.

### TO USE

#### To play once

(SOLITAIRE *SOLOW REPLAY*)

[Function]

Plays one hand of solitaire, which it will animate in the window *SOLOW* (which should be at least 700 by 700, although the program will do its best to adapt). If *REPLAY* is T, SOLITAIRE will use the deck from the previous shuffle, else it will deal a new hand.

#### To play repeatedly

(SOLO *SOLOW*)

[Function]

Calls (SOLITAIRE *SOLOW*) repeatedly.

#### The results

SOLO keeps a record of the frequency of each of its results in the array SOLORESULTS [0..52] which it plots at the end of each hand.

#### As an IDLE hack

Loading SOLITAIRE automatically adds SOLITAIRE as an option to the IDLE menu. If chosen, it will be given the ~whole screen~ covering window of IDLE and will use a black background, rather than its usual shaded one, to preserve the screen phosphor. Otherwise, its operation is completely normal.

---

---

**STARBG**

---

---

By: Gregg Foster (Foster.PA@Xerox.COM)

Upgraded for Medley by Larry Masinter (Masinter.PA@Xerox.COM)

STARBG creates a random star field for your screen background and a little flying saucer to follow your cursor when it's in space (so it doesn't get lost). It also supplies an alternate IDLE function, Cosmos.

The star field will look something like this:



The saucer will look like this:

**USAGE**

(STARBG) [Function]

STARBG fills a screensized bitmap with random stars, turns the saucer on, and calls CHANGEBACKGROUND. If you don't like the star pattern you get, try it again.

(Cosmos window) [Function]

Cosmos is puts an evolving universe in a window. It's intended as an IDLE function, but will entertain you for hours in any decently sized window.

(SaucerOn) [Function]

SaucerOn turns the saucer on by changing the CURSORBACKGROUND\*FNs.

(SaucerOff) [Function]

SaucerOff turns the saucer off and sets the BACKGROUNDCURSOR\*FNs to NIL.



**CUSTOMIZATION**

There are lots of user-settable parameters, all of which have reasonable defaults. Here are some of the interesting ones:

STARBGParameters [Variable]

is a list of settable parameters. Most are dotted pairs specifying ranges (e.g. stars3 defaults to (6 . 70) meaning that STARBG will make 6 to 70 type-3 stars). The others are bitmaps.

BM1, ..., BM5 [Variables]

The star bitmaps used to BLT the stars. BM1 must be a single bit.

SBM [Variable]

The starry screen bitmap. This is reused in subsequent calls to STARBG.

stars1, ..., stars5 [Variables]

Ranges for the 5 kinds of stars.

constellations [Variable]

Range for number of constellations. A constellation is a group of bright stars.

clusters [Variable]

Range for number of clusters. Clusters are tightly globular.

superClusters [Variable]

Range for number of superClusters. SuperClusters are clusters of clusters.

eventPause [Variable]

Number of milliseconds to block between events. Larger numbers have the effect of slowing down the rate of evolution..

changeStars [Variable]

Will use the IDLE-ing star field as your new background.

---

---

**STEP-COMMAND-MENU**

---

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

This package changes the function CL::STEP-COMMAND (used by CL:STEP) to call a new function (instead of IL:ASKUSER) to get its commands from a menu attached to the stepping window (depending on the setting of the CL:SPECIAL variable IL:\*STEP-COMMAND-MENU\*). The value of IL:\*STEP-COMMAND-MENU\* is initialized to T as an INITVAR when the file is loaded. The variable USER:\*STEP-COMMAND-INVERT-MENU-SHADE\* is the shade used to *grey-out* the attached menu when the stepping is not awaiting a command. The menu is attached to the Right edge (at the Bottom) of the stepping window. (If there isn't enough room on the Right, it will be attached to the Left edge.) The menu is detached and closed when the stepping level which first attached it is exited.

## STORAGE

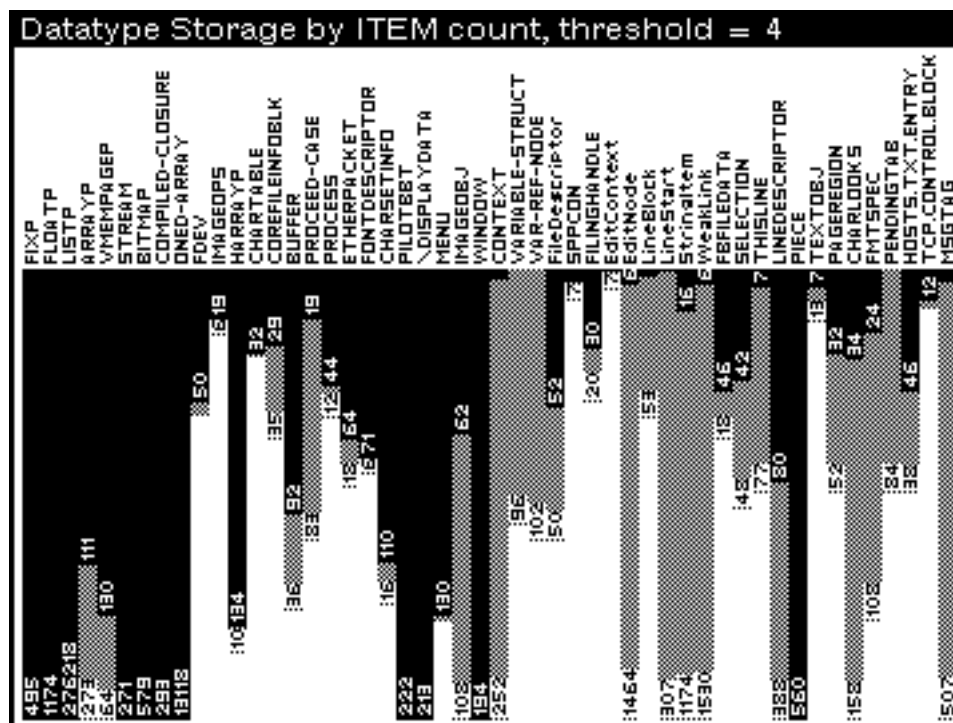
By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

STORAGE implements a bar-graph version of the Lisp STORAGE function, providing a visual summary of the amount of storage allocated to each data type.

(SHOWSTORAGE [*PAGETHRESHOLD* *MODE* *ROTATION*])

[Function]

Displays the storage allocation of Lisp data types in bar graph format:



All the arguments are optional. *PAGETHRESHOLD* is the same as for the STORAGE function and defaults to 1. *MODE* determines what to display and can be one of the following:

- ITEM** The number of items of each type that have been allocated (the default mode).
- PAGE** The number of pages allocated for each type.
- BOX** The number of times each type has been allocated (see BOXCOUNT in the IRM).

The mode can be changed when the window is open by clicking with the *middle* mouse button. Clicking in the window with the *left* mouse button will update the window. When the window is redisplayed (using the standard window menu or REDISPLAYW) it will add new data types that have been defined since the window was last redisplayed.

For the **ITEM** and **PAGE** modes, the black part of the bar represents the number of items or pages currently *in use*. The gray part of the bar represents the number of *free* items or pages. The total length of the bar represents the *total* number of items or pages.

The *ROTATION* argument can be one of **NIL** (use the rotation of the *SHOWSTORAGEFONT*), **0** (labels from bottom to top on the right, bars grow to the left) or **90** (labels from left to right and bars grow down).

The display is controlled by the following global variables:

*SHOWSTORAGEWINDOWSIZE* [Variable]

The width or height (depending on the rotation) of the window, initially 275 (pixels). The bars truncate at the edge of the window; the window can be reshaped to put the longer bars in perspective.

*SHOWSTORAGEIGNORE* [Variable]

A list of data types to ignore. The information for the data types initially on this list is incorrect and/or their inclusion breaks the program.

*SHOWSTORAGEDEFAULTTHRESHOLD* [Variable]

The default threshold used when *PAGETHRESHOLD* is NIL, initially 1 (page).

*SHOWSTORAGEPRIN2FLG* [Variable]

Flag that causes PRIN2 to be used instead of PRIN1 when printing data type names (PRIN2 will include package names), initially NIL.

*SHOWSTORAGEFONT* [Variable]

The window font, initially one of Helvetica 5 through 10, i.e. the smallest that can be found when the file is loaded. The default font has a rotation of 90 degrees.

---

---

**STYLESHEET**

---

---

By: Tayloe Stansbury

Unsupported

## INTRODUCTION

Stylesheets are collections of menus. These collections pop up all at once in a group. This group does not disappear until all menus in it have been dealt with, and the user signals that he is done.

Stylesheets are intended to be used in situations wherein the computer wants an answer to several related questions all at once. One example is font selection. To select a font, the user needs to specify font family (Classic, Modern, etc.), font size (8 point, 10 point, etc.), and font style (bold, italic, etc.). Rather than prompt for each of these parameters in succession, one could use a stylesheet to prompt for it all at once.

When the stylesheet pops up, it will shade (preselect) default selections (if provided) in each of the menus. The user can either decide that the default selections are OK, or change them to suit his taste. (The default selection mechanism can be used to convey the current state of something the user is trying to change with the stylesheet: for example, the current looks of the text with which the user is dissatisfied.)

When the user is finished, he hits the DONE button and the stylesheet disappears, and the final selections are returned. There is also a RESET button. This is useful if the user has mucked up his selections and would like to reinstate the default selections. Finally, there is an ABORT button that if selected returns NIL from STYLESHEET and is intended to provide the user with a convenient way of aborting the selection. Note: This means that NIL can be returned from a call to STYLESHEET.

Menus in a stylesheet can be set up to accept exactly one selection (like a normal menu), less than two selections, or any number of selections. Menus that need not be filled in (i.e., can accept zero selections) have an attached CLEAR button, which can be used to remove selections made in that menu. Menus that can have more than one selection have an attached ALL button, which can be used to select all the items in the menu.

## HOW TO MAKE A STYLESHEET

To create a stylesheet, call

(CREATE.STYLE *Prop1 Value1 Prop2 Value2 ... PropN ValueN*)

[Function]

CREATE.STYLE accepts an arbitrary number of property-value pairs. Properties currently recognised are

ITEMS [Style Property]

A list of menus. Most menu format parameters contained in menu records are honored by the stylesheet package. WHENSELECTEDFNs are, of course, ignored.

SELECTIONS [Style Property]

A list of menu items, each one corresponding to a menu in ITEMS. The specified selections will be shaded in the appropriate menu, and will be the default selections. If not specified or too short, it will be filled out with NILs (no selection).

NEED.NOT.FILL.IN [Style Property]

A list of T or NIL or MULTI, each one corresponding to a menu in ITEMS. T indicates that the corresponding menu need not be filled in and will be given a CLEAR button. MULTI indicates that the corresponding menu can have any number of selections and will be given both a CLEAR button and an ALL button. If the list is too short, it will be filled out with NILs. If a single T or NIL or MULTI is given instead of a list, it will be replaced by a list of Ts or NILs or MULTIs, respectively.

TITLE [Style Property]

The title that will be given to the stylesheet. If no title is specified, the stylesheet will not have a title bar.

ITEM.TITLES [Style Property]

A list of strings or atoms to serve as titles over the menus. Items without titles specified will not have titles.

ITEM.TITLE.FONT [Style Property]

A fontdescriptor or other font specification which determines the font item titles will be printed in. If NIL, titles will be printed in DEFAULTFONT.

POSITION [Style Property]

The screen position (of type POSITION) of the lower left-hand corner of the stylesheet. If position is not specified, the function STYLESHEET will prompt for the position (using GETBOXPOSITION). STYLESHEET will modify positions as necessary to ensure that the entire stylesheet will be on the screen.

Stylesheets can be modified by calling

(STYLE.PROP *Stylesheet Prop Newvalue*)

[Function]

STYLE.PROP always returns the old value of the specified property of the specified stylesheet. If Newvalue is provided (even if NIL), it replaces the old value. If not provided, the old value remains. (Just like WINDOWPROP.)

To use the stylesheet thus created, call

(STYLESHEET *Stylesheet*)

[Function]

This returns a list of selections the user made from the stylesheet. (If a selection is returned as NIL, that indicates that no selection was made.)

One can determine in advance of displaying a stylesheet how big it will be. (This may help in determining a reasonable screen position for the stylesheet.) The relevant functions are

(STYLESHEET.IMAGEWIDTH *Stylesheet*)

[Function]

and

(STYLESHEET.IMAGEHEIGHT *Stylesheet*)

[Function]

They return the width and height, respectively, of the stylesheet in pixels.

## AN EXAMPLE

The package is located in STYLESHEET and STYLESHEET.DCOM. To familiarize yourself with its workings, you might want to load it and try the following example:

```

(SETQ FONT.STYLE
 (CREATE.STYLE
 'TITLE "Please select a font:"
 'ITEM.TITLES '(Family Size Face)
 'ITEM.TITLE.FONT '(Modern 12)
 'ITEMS
 (LIST
 (CREATE MENU ITEMS ← '(Classic Modern Terminal))
 (CREATE MENU ITEMS ← '(8 9 10 11 12 14))
 (CREATE MENU ITEMS ← '(Regular Bold Italic BoldItalic)))
 'SELECTIONS '(Modern 11 Regular)
 'NEED.NOT.FILL.IN 'T]

(STYLESHEET FONT.STYLE)

```

---

---

## SuperParentheses

---

---

By: Andrew J. Cameron, III (Cameron.pa@Xerox.com or cameron@cs.wisc.edu)

Most useful when used with: WHO-LINE (LispUsers)

This document last edited on Oct 19, 1987.

### INTRODUCTION

This file, when loaded, creates a readtable (named "LISP[]") for use with CommonLisp which contain SuperParentheses, that is, the left square bracket (LEFTBRACKET syntax class) and right square bracket (RIGHTBRACKET syntax class) available in InterLisp. CommonLisp does not give these two characters their "usual" definitions, so as to allow users to easily give these character any macro/syntax definition they might desire.

This readtable will appear on, and can be selected via, the "Rdtbl" menu provided by the WHO-LINE LispUsers utility.

One can also access this new readtable with:

```
(IL: FIND-READTABLE "LISP [] ")
```

These facilities obviate the need to store the readtable on a variable, as was done in an earlier version of this module.

### INTERNALS

- The reading and writing of files using this readtable has not been tested or explored.
- SEdit is not too friendly to SuperParentheses.
- For more information, see Section 25.8.2 in both the IRM (InterLisp Reference Manual) and the Lyric Release Notes, and Section 22.1 in Steele (CommonLisp - The Language).



---



---

## SYSTATS

---



---

By: Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester)

This document last edited on: October 28, 1987

### SUMMARY

SYSTATS provides a functional interface to system statistics such as PageFaults, DiskIOTime, etc. Statistics are maintained in objects of type SYSTATS. Functions are provided to fetch values from these objects, and to update the objects to reflect the current system state or to compute differences. This facility provides a Lyric alternative to the (undocumented) MISCSTATS functions in Koto.

### DESCRIPTION

SYSTATSPROPS [Variable]

A list of statistics maintained by SYSTATS. Changing it does **not** alter SYSTATS behavior.

(SYSTATSPROP *prop fromstats*) [Function]

If *fromstats* is NIL, the internal SYSTATS object is updated and used. Returns the value of the statistic named by *prop*, which must be a member of the variable SYSTATSPROPS.

**Caveat:** The value returned is a FIXP which is an element of the *fromstats* object and which, for the sake of performance, is reused during a SYSTATSREAD on the *fromstats* object. Note that there is an implicit SYSTATSREAD on the internal SYSTATS object if *fromstats* is NIL.

(SYSTATSREAD *intostats fromstats*) [Function]

If *intostats* is NIL, it is set to a newly created SYSTATS object. If *fromstats* is NIL, the internal SYSTATS object is updated and used. Copies system statistics from *fromstats* into *intostats*. Returns *intostats*.

(SYSTATSDIFF *oldstats newstats difstats*) [Function]

If *oldstats* is NIL, the internal SYSTATS object is updated and used in its place. If *newstats* is NIL, the internal SYSTATS object is updated and used in its place. If *difstats* is NIL, it is set to a newly created SYSTATS object. Computes the statistics differences between *oldstats* and *newstats*, and places the results in *difstats*. Returns *difstats*.

(CLOCKTICKS *interval timerunits*) [Function]

Returns the (machine dependent!) number of internal clock ticks over the interval. For instance, on the D'Lion, (CLOCKTICKS 2.5 'MINUTES) = 5211900.

---



---

## TALK

---

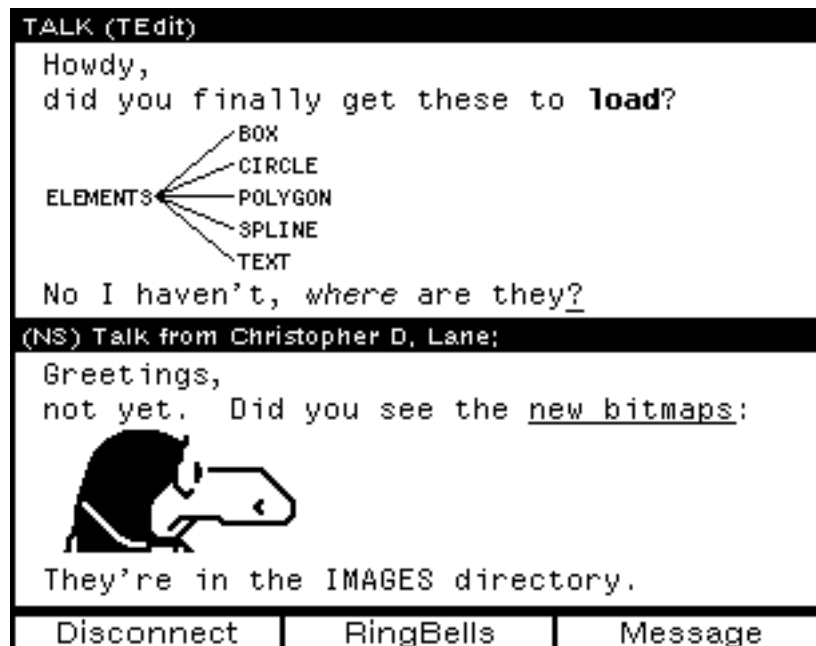


---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: Various editor and network protocol modules.

TALK allows users to hold conversations between machines across the Ethernet. TALK uses various *services* (TTY, TEdit and Sketch) and network *protocols* (NS and IP).



### TALK FILES

Talk's services and protocols are now in separate files which may be loaded independently:

TALK            The main Talk module.

#### Services

TTYTALK        Simple text conversation between machines running Lisp, XDE and Viewpoint.

TEDITTALK     Uses TEDIT; allows the full capabilities of the TEdit editor in a conversation.

SKETCHTALK    Uses SKETCH; allows a conversation using the Sketch graphics editor.

#### Protocols

NSTALK         Uses COURIERSERVE (and optionally NSTALKGAP); allows XNS protocols.

NSTALKGAP     Used by NSTALK if the GAP Courier program has not been defined (by NSCHAT).

IPTALK         Uses TCP and TCPUDP; allows conversations using IP protocols.

Any Talk service can be used with any Talk protocol. The preferred order of loading is:

```
(FILESLOAD TALK TEDITTALK TTYTALK SKETCHTALK NSTALK IPTALK)
```

dropping out those services/protocols you do not use. Order of loading determines which services/protocols are tried first; the files may be loaded in any order to force different priorities.

## USING TALK

(TALK [USER.OR.HOSTNAME SERVICE PROTOCOL]) [Function]

Starts a TALK session; *USER.OR.HOSTNAME*, *SERVICE* and *PROTOCOL* are optional. If not supplied, *USER.OR.HOSTNAME* is prompted for (either by menu or typein or both). If *SERVICE* and *PROTOCOL* are not supplied (the usual case), TALK will figure out which to use based on what is available on the local and remote machines. The supported services and protocols are described below. The service and protocol used are indicated in the title bars of the TALK window.

TALK returns a process handle if the connection is successfully opened; it returns NIL if the user aborts out of the host/user menu and it returns an error message (as a string or list instead of breaking) if it cannot contact the remote host (for whatever reason). TALK can also be invoked from the background menu.

## TALK MENU

The menu at the bottom of the TALK window (which is only active while the connection is open) contains the following items:

- Disconnect** Closes the TALK connection. This is equivalent to closing the TALK window, but leaves the window open in case you want to save and/or hardcopy part or all of the session.
- RingBells** Rings the bell on the remote workstation (if possible) and flashes the TALK window on the local one to indicate it has done so. This is useful if you have asked a person to hold and want to let them know you have returned.
- Message** Prompts for and inserts a *canned* message into the TALK stream. Useful if the phone rings and you want to ask the other person to hold with a minimum of time/effort. Messages can be added to the list, see the TALK.USER.MESSAGES variable below. The TALK window must have the keyboard in order to use this item.

## ERROR MESSAGES

The TALK function will return one of the following error messages when it fails to start a session:

- Host not found!** It could not find host address for the host or user name specified.
- Can't connect to host!** The remote workstation does not have the appropriate server loaded and/or running or does not have TALK loaded.
- No answer from TALK service!** A connection was made, but no one responded (intentionally or otherwise). A darkened TALK icon is left on the remote screen to log the connection attempt (unless TALK.GAG is non-NIL).
- Unknown service type!** An unknown type was given as the *SERVICE* argument.
- No services available!** The *SERVICE* argument was not supplied and it cannot find one.
- Unknown protocol!** An unknown protocol was given as the *PROTOCOL* argument.

**No protocols available!** The *PROTOCOL* argument was not supplied and it cannot find one.

Service and protocol errors may indicate additional files need to be loaded.

## RECEIVING TALK

When your machine is contacted by another via TALK, the following icon will appear on your screen, ringing bells, flashing and showing the time, mode (service and protocol) and (when possible) the caller's identity:



If you button the icon with the left or middle buttons, a TALK session will begin. If you either close the icon or do not button it (it will go *dark* (invert) in about 15 seconds if not buttoned) the TALK connection will be refused. TALK connections are automatically refused if the TALK.GAG flag is non-NIL (settable using the subitem(s) of the TALK item in the background menu). If the machine is in IDLE, TALK will wait twice the normal time out for the user to respond.

If you button a darkened (unanswered) TALK icon, it will try to reconnect you to the caller (after a mouse confirm). If a TALK connection comes in from someone who has already left an unanswered TALK icon on your screen, the icon will be reused.

## TALK SERVICES

### TEdit

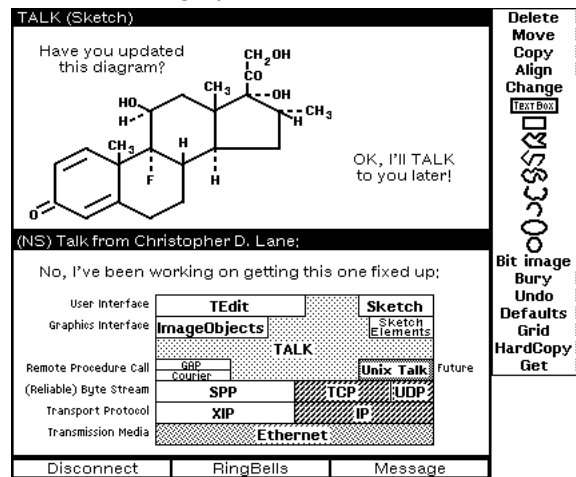
This service allows you to use the full capabilities of TEdit in your conversation, including: correcting mistakes anywhere in the document, changing character and paragraph looks, inserting ImageObjects, etc. Along with keyboard input, mouse selections and the caret are also visible to the remote user. The **GET** and **INCLUDE** commands in the TEdit command menu will load files into both the local and remote TEdit windows, so make sure the files are accessible to both. Similarly for fonts, if your workstation has to load a font from a server, the remote workstation must also have access to the font. Since the remote workstation may also need to load the font, you may experience communication delays. The TEdit service supports NS character codes and most of the 1108 and 1186 function keys.

### TTY

This service is similar to the TEdit service except that the only supported feature is *backspace* (but not across lines). TTY is the only service that can talk with the Talk.bcd program in XDE or the TALK application (VPTalk.bcd) in Viewpoint. You *do not* need to know what type of workstation you are contacting when using any of the TALK programs.

## Sketch

The Sketch service is built on the Sketch graphics editor:



## TALK PROTOCOLS

### NS

When NSTALK is loaded, TALK will accept as a host name anything that COURIER.OPEN will accept including an NS address or the name of a workstation registered in a Clearinghouse. Additionally, user names can be used if the address of the user's workstation is registered under the user's name in the Clearinghouse. The following function can be used to register a user and workstation correspondence in the Clearinghouse:

(CH.USER.WORKSTATION *USER WORKSTATION*) [Function]

Sets (or changes) the AddressList Clearinghouse property of *USER* (which must already be a name or alias in the Clearinghouse) to be the address of *WORKSTATION* (an NS address or name). If *WORKSTATION* is NIL, the function removes the AddressList property from *USER*. To use this function, you must be logged in (via (LOGIN)) as a System Administrator for *USER*'s domain.

One way to register users would be to go to the individual's workstation, login as the System Administrator and evaluate: (CH.USER.WORKSTATION 'UserName \MY.NSADDRESS)

Note that you cannot use the USERNAME function in this example since the (LOGIN) will change it.

NSTALK *does not* require or use NSCHAT, but they do share the Courier program GAP. If both NSTALK and the NS CHATSERVER modules are to be loaded, the CHATSERVER should be loaded *first* if possible. NSTALK is designed to allow other types of NSCHAT/GAP servers. The GAP server function determines which function to call using the service type requested (TTY = 5, TEdit = 6, Sketch = 7) and the entries on the association list GAP.SERVICETYPES which has entries of the form (ServiceNumber ServiceName ServerFunction). It is possible to have both NSTALK running and an EXEC server by adding appropriate entries to GAP.SERVICETYPES. If a GAP server already exists when NSTALK is loaded, it is made the default for all unrecognized service types.

Although NSTALK loads the COURIERSERVE LispUsers module you do not have to have a Courier server running to initiate an NS TALK connection, but you *must* have one running in order to receive an NS TALK connection.

**IP (Interim)**

When IPTALK is loaded, TALK will accept as a host name anything that DODIP.HOSTP will accept, including symbolic and numeric IP addresses. User names can be used by adding them as synonyms for local workstation hosts in the HOSTS.TXT file.

The current TALK IP interface is *only temporary* and will eventually be replaced by one which is compatible (for TTY service) with the TALK program which runs under BSD Unix; at that time, the allowable username format may be expanded to handle user@host. The current IP interface will probably not be compatible with the eventual, Unix-compatible one.

**TALK VARIABLES**

The following variables can be used to affect TALK's default behavior:

TALK.DEFAULT.REGION = (0 0 500 500) [Variable]

The LEFT and BOTTOM of this region determine where the (initial) TALK icon appears on the screen; the HEIGHT and WIDTH are the combined dimensions of the TALK windows (each uses half the HEIGHT). If this variable is set to NIL, then the icons start at (0 . 0) and the TALK window region is prompted for as needed.

TALK.USER.MESSAGES [Variable]

A list of menu items to put up when the MESSAGES item on the TALK menu is selected. Items on the list should return strings to be put into the TALK stream. If there is an entry of the form (GREETING "message") on this list, it will be printed automatically when a connection is opened.

TALK.GAG = NIL [Variable]

If non-NIL, causes the TALK server to automatically reject any TALK connections.

TALK.ANSWER.WAIT = 15 [Variable]

The number of seconds the TALK icon remains up before closing and aborting the connection.

TALK.HOSTNAMES = NIL [Variable]

A list structure containing hosts TALK has connected to along with the address used.

TALK.SERVICETYPES [Variable]

This list determines which services are tried and in what order. You only need to modify this if you wish to force an order other than the one determined by the order files were loaded or you wish to add or drop a service.

TALK.PROTOCOLTYPES [Variable]

This list determines which protocols are tried and in what order. You only need to modify this if you wish to force an order other than the one determined by the order files were loaded or you wish to add or drop a protocol.

**KNOWN PROBLEMS****Talk**

- Since TALK uses the Dove/Dandelion sound generator to help announce a connection, on other machines it is difficult for the user to detect connections being made during IDLE.

### TTY Talk

- The TTY service cannot backspace beyond the left margin (unlike other implementations).

### TEdit Talk

- Sometimes the local and remote TEdit windows will get out of sync as to what the current *looks* are; usually this is not serious.
- Page layout commands have not been implemented for the remote TEdit window; there are probably other commands that do not work either.
- ImageObject specific manipulations to ImageObjects already in the window do not get transmitted to the remote Tedit window.
- Inserting (other than keyboard input) into a pending delete does not echo correctly on the remote Tedit window.
- A large ImageObject inserted into the TALK window may not be seen by the remote user until some text is typed to force the remote window to scroll. The remote user may not see the ImageObject at all if it is larger than his window. These are both true of any TEdit window.
- User scrolling of the TEdit window will not cause scrolling of the remote TEdit window. System scrolling of the window (due to insertions and deletions) will be tracked in the remote window.

### Sketch Talk

- When the TALK window is opened, some sketch menus will be created and then replaced. This is due to Sketch not allowing a user to specify both an existing window and an initial menu.
- When text (or a text box) is entered, only the initial character is seen in the remote window until the text is completed and the user buttons some other point in the window.
- Arrow heads do not show up at all on the remote sketch window.
- **Put** of a SKETCHTALK sketch gets into an infinite loop so temporarily you must copy the sketch items to another sketch if you wish to save them on a file.
- If you *sweep* a control point on a box past the other one (like sweeping one corner of a region past the other in RESHAPE), the remote box will not move identically.
- Since there are no functions to programmatically manipulate grouped elements the **Group** and **UnGroup** items have been disabled in the Sketch Talk window.
- For a small number of changes (text fonts, text box brushes and closed wire dashing), the entire remote sketch window is redisplayed to make the change visible.
- Setting the SKETCHINCOLOR flag to a non-NIL value will cause some operations to break.



---

---

## TCPTIME

---

---

By: Christopher Lane (Lane@Sumex-Aim.Stanford.Edu)

Uses: TCP, TCPUDP

TCPTIME implements time client and server routines under TCP/IP and UDP/IP based on RFC868. The following are the user functions; the *PROTOCOL* argument refers to one of **TCP** or **UDP** and defaults to the value of RFC868.DEFAULT.PROTOCOL, initially **TCP**. All arguments are optional:

(RFC868.SETTIME [*RETFLG PROTOCOL*]) [Function]

Obtains the time from the network, similar to the \PUP.SETTIME and \NS.SETTIME functions. If *RETFLG* is non-NIL, the time is returned as an integer (as specified in RFC868), otherwise SETTIME is called and the new time is printed in the prompt window. Either TCP.TIME.HOSTS and/or UDP.TIME.HOSTS (see below) must be set before calling this function.

(RFC868.START.SERVER [*PROTOCOL ASCIIIFLG*]) [Function]

Starts a network time server process for the specified (or default) *PROTOCOL* if one is not already running. The *ASCIIIFLG* is discussed below.

(RFC868.STOP.SERVER [*PROTOCOL*]) [Function]

Deletes the network time server process for the specified (or default) *PROTOCOL* if one is running.

The following variables are used by the functions above:

RFC868.TIME.PORT = 37 [Variable]

Used to set the initial value of the protocol specific port variables when the file is loaded. Once the file is loaded, changing this variable has no effect, so it must be reset (if necessary) before loading the file, otherwise the protocol specific port variables should be reset directly. See TCP.TIME.PORT and UDP.TIME.PORT below.

RFC868.DEFAULT.PROTOCOL = TCP [Variable]

The default protocol to use when one is not specified.

### **BINARY & ASCII TIME FORMAT**

Some network software implements the RFC868 standard by returning the printed (ASCII) representation of the time, rather than the binary representation as specified in the RFC. To work around this, the ASCIIIFLG can be specified when starting a server to indicate that it should output the printed representation of the number. Similarly, when getting the time from the network, the following is used:

RFC868.ASCII.OSTYPES = (VMS) [Variable]

to decide based on the host's operating system whether to read the time as a binary or ASCII number. If this variable is set to NIL, the ASCII format is never used.

The ASCII format is currently only supported in the TCP protocol.



**PROTOCOL SPECIFIC FUNCTIONS**

(TCP.SETTIME *[RETFLG]*) [Function]

(UDP.SETTIME *[RETFLG]*) [Function]

Functions called by RFC868.SETTIME which can be called directly. The variables TCP.TIME.HOSTS and UDP.TIME.HOSTS must be set to use these functions.

(TCP.TIMESERVER *[ASCIIFLG]*) [Function]

(UDP.TIMESERVER) [Function]

Functions used by RFC868.START.SERVER. Can be used directly using ADD.PROCESS.

TCP.TIME.PORT = RFC868.TIME.PORT [Variable]

UDP.TIME.PORT = RFC868.TIME.PORT [Variable]

The ports to use in both the client and server functions.

TCP.TIME.HOSTS [Variable]

UDP.TIME.HOSTS [Variable]

Lists of host names and/or addresses (including broadcast addresses) to try to get the time from. Host are tried until one responds.

TCP.SETTIME.TIMEOUT = 10000 [Variable]

UDP.SETTIME.TIMEOUT = 10000 [Variable]

Length of time (in milliseconds) to wait for a host to respond to TCP.OPEN or UDP.EXCHANGE before trying the next one on the list.

---

---

**TEdit-Close-On-Shrink**

---

---

By: Nick Briggs (Briggs.pa@Xerox.com)

Uses: TEdit

This document last edited on August 4, 1987.

**INTRODUCTION**

TEdit has the unfortunate habit of keeping the file you are editing open when you shrink the TEdit window. For users of the FileCache this has the unfortunate sideeffect of preventing a newly saved file from being written out. In places where many people are editing files on file servers can lead to an excessive number of open files. TEdit-Close-On-Shrink attempts to handle some of the problem by persuading TEdit to close unmodified files when the edit window is shrunken.

**USE**

Load the module. It will install itself in the appropriate places (for the curious: that's `TEDIT.CREATEW`, `\TEDIT.CREATEW.FROM.REGION`, and `\TEDIT.REOPEN.STREAM`; and the shrink function `\TEDIT-CLOSE-ON-SHRINK`).

# TEdit Dorado Keys

*A set of convenience keys for the Dorado and TEdit.*

## INTERNAL

This package defines a number of meta-keystrokes as TEdit commands, providing much of the functionality of the 1186's expanded keyboard. An effort was made to keep the command set compatible with SEdit's for similar functions.

In a number of cases, the meta-lower-case and meta-upper-case commands are different--typically, the lower-case command turns some attribute (like boldness) on, and the upper-case command turns it off.

Here's the set of defined functions:

|                   |                                                                       |
|-------------------|-----------------------------------------------------------------------|
| <b>meta-u</b>     |                                                                       |
| <b>meta-U</b>     | UNDO                                                                  |
| <b>meta-f</b>     |                                                                       |
| <b>meta-F</b>     | FIND                                                                  |
| <b>metaa</b>      |                                                                       |
| <b>meta-A</b>     | ABORT (i.e., do a GET)                                                |
| <b>meta-s</b>     |                                                                       |
| <b>meta-S</b>     | SUBSTITUTE                                                            |
| <b>ESC</b>        | REDO (note that this changes the default definition, which is EXPAND) |
| <b>meta-x</b>     |                                                                       |
| <b>meta-X</b>     | EXPAND (also on control-X)                                            |
| <b>meta-n</b>     |                                                                       |
| <b>meta-N</b>     | Move to NEXT fill-in blank                                            |
| <b>meta-c</b>     |                                                                       |
| <b>meta-C</b>     | Change margins, rotating thru centered, left, right, just.            |
| <b>meta-b</b>     | Turn BOLD on                                                          |
| <b>meta-B</b>     | Turn BOLD off                                                         |
| <b>meta-i</b>     | Turn ITALIC on                                                        |
| <b>meta-I</b>     | Turn ITALIC off                                                       |
| <b>meta-=</b>     | Turn strike-thru on                                                   |
| <b>meta-+</b>     | Turn strike-thru off                                                  |
| <b>meta--</b>     | Turn underline on                                                     |
| <b>meta-__</b>    | Turn underline off                                                    |
| <b>meta-^</b>     | Superscript (or de-superscript)                                       |
| <b>meta- </b>     | Subscript (or de-superscript)                                         |
| <b>meta-space</b> | Return to default font/weight/slope/etc.                              |
| <b>meta-?</b>     | Show current font in prompt window.                                   |
| <b>meta-(</b>     | Insert parentheses around the current selection                       |
| <b>meta-"</b>     | Insert neutral double quotes (ASCII ") around the current selection   |
| <b>meta-'</b>     | Insert real double quotes (" and ") around the current selection      |

---

---

## TEDITKEY

---

---

By: Greg Nuyens

Supported by: Jan Pedersen (Pedersen.pa@Xerox.com)

Uses: KEYOBJ, DLIONFNKEYS

TEditKey is a module that provides a keyboard interface to TEdit. On a Dandelion, the interface takes advantage of the special keys to the left, top, and right of the main keyboard. On a Dorado or Dolphin, a window mimicking the Dandelion function keys provides some of the same abilities.

The abilities provided include allowing the user to alter the *caret looks* (the looks of characters typed in) or the selection looks. These commands are given using the Dandelion function keys and/or metacodes. (Metacodes are keys typed while a meta key is held down. The default meta key is the tab key; to alter this see "User Switches" below.) Other metacodes and control codes move the cursor within the document (beginning/end of line, back/forward a character, up/down a line, etc.).

Thus, many of the special Dandelion keys are made to function in TEdit the way they are labeled. The following keys change their behavior once TEditKey is loaded.

**CENTER** modifies the justification of the paragraph(s) containing the current selection. If the selection was left justified, then hitting the CENTER key makes it centered. Hitting it again produces right and left justification.

**BOLD** boldfaces the selection. All other properties remain unchanged. Holding the shift key down while hitting BOLD will make the selection become un-bold.

**ITALICS** italicizes the selection. Shift-ITALICS is the opposite.

**UNDERLINE** underlines the selection. Shift-UNDERLINE is the opposite.

**SUPERSCRIP** superscripts the selection by a constant amount. Any relative superscripts (or subscripts) are maintained. Thus if "X" is selected in "the set X is empty" then pressing the SUPERSCRIP button produces "the set <sup>X</sup> is empty." See "User Switches" below for how to set the increment. Shift-SUPERSCRIP is the same as SUBSCRIPT.

**SUBSCRIPT** is analogous to SUPERSCRIP.

**SMALLER** decreases the font size of the selection. All relative size differences are maintained. E.g., "this is bigger than that" produces "this is bigger than that." Shift-SMALLER (labeled LARGER) does the opposite.

**DEFAULTS** makes the selection have default looks. N.B.: The default looks can be set to the current caret looks by typing shift-DEFAULTS.

The above keys all affect the caret looks if the keyboard key is held down when they are hit. Thus holding down KEYBOARD and then hitting UNDERLINE makes the caret looks be underlined.

**FONT** changes the font of the selection or caret looks according to the following table (to alter this table see "User Switches" below):

|   |             |
|---|-------------|
| 1 | Times Roman |
| 2 | Helvetica   |
| 3 | Gacha       |
| 4 | Modern      |
| 5 | Classic     |
| 6 | Terminal    |
| 7 | Symbol      |
| 8 | Hippo       |

Thus, to change the font of the selection to Classic, hold down FONT and hit 5. To change the caret font to Classic, hold down FONT (to signal the font change) and KEYBOARD (to direct the change to the caret looks) then hit 5. Note that this table is part of the menu displayed when the HELP button is pressed.

On a Dorado, middle-blank is the FONT key.

**KEYBOARD** applies any changes that occur while this key is down to the caret looks instead of the selection. On a Dorado, bottom-blank is the KEYBOARD key.

**AGAIN** invokes the redo facility in TEdit. A wide variety of operations can be repeated very simply by making a selection, performing an operation (for instance, an insertion), then picking a new selection and hitting the AGAIN key. The AGAIN key is an ESCape key, which acts as the TEdit REDO syntax class. (See page 20.22 of the *Interlisp Reference Manual*.)

**OPEN** opens a blank line at the current cursor position. OPEN is also used to type a linefeed outside of TEdit (for example to the function FILES?).

**FIND** prompts the user for a target string, then searches from the selection forward.

**NEXT** acts as the TEdit NEXT syntax class. (It goes to the next field to be filled in. These fields are marked as follows: >>text to be substituted<< .)

**shift-NEXT** transfers the TTY (which window will receive typed characters) to the next window which can accept typein. Thus one can cycle through the open text windows (mail windows, top level lisp windows, TEdit windows, etc.) without using the mouse.

**EXPAND** expands TEdit abbreviations. (See page 20.31 of the *Interlisp Reference Manual*.)

**HELP** displays a menu of the keybindings until a mouse key is clicked.

**UNDO** acts as the TEdit UNDO syntax class. Note that it still retains its TELERAID function as does STOP. There are TEditKey operations (such as Transpose Characters) that are implemented with multiple TEdit operations. Since TEdit will UNDO only single operations, it does not fully UNDO these operations.

**RightArrow** enters \, and | when shifted. (Recall that AGAIN is an escape key.)

**MARGINS** indents the margins of the paragraph selected. Shift-MARGINS exdents the margins. If the right margin is a floating margin, it is left unchanged. To control the amount by which the margins are moved, see "User Switches."

As well as the previous functions available on the Dandelion's special keys, the following functions are available on the standard keyboard (thus usable on the Dandelion, Dolphin, and Dorado). Each function is shown with the key that invokes it (in conjunction with the control (denoted ^) or meta (denoted #) key). Thus, for the sixth entry, holding down the metakey and hitting f (or "F") would move the caret one word forward. (To find out how to get a metakey see "User Switches" below.)

|    |                                                           |
|----|-----------------------------------------------------------|
| #/ | defaults the caret looks                                  |
| #= | queries caret looks                                       |
| #9 | smaller caret font                                        |
| #0 | larger caret font                                         |
| ^b | back character                                            |
| ^f | forward character                                         |
| #b | back word                                                 |
| #f | forward word                                              |
| ^p | previous line                                             |
| ^n | next line                                                 |
| ^a | beginning of line                                         |
| ^e | end of line                                               |
| #< | beginning of document                                     |
| #> | end of document                                           |
| #s | select whole document                                     |
| ^k | kills line (delete from caret to end of line)             |
| ^o | opens line                                                |
| ^d | deletes character forward (also on shift backspace)       |
| #d | deletes word forward (as always ^w deletes word backward) |
| ^t | transposes characters                                     |
| #[ | indents paraloops. Also available on the MARGINS key      |
| #] | exdents paraloops. Also available as shift-MARGINS        |
| #j | justification change (same as CENTER key)                 |
| #u | uppercases selection                                      |
| #c | capitalizes selection                                     |
| #l | lowercases selection                                      |
| #o | inserts object into document                              |

#?        shows keybindings (same as HELP)

#r        restores the display

Note that the positions of any of these functions can be individually changed using TEDIT.SETFUNCTION (see page 20.30 of the *Interlisp Reference Manual*). For wholesale customization see "User Switches" below.

## INTERRUPTS

Any operation can be aborted by typing the STOP key. This can be used to abort font changes, GETs, PUTs, etc. A stronger form of interrupt is available as shift-STOP, which prompts the user for a menu of processes to interrupt.

^G is available as a synonym for hitting the STOP key within TEditKey. Outside of TEdit, however, ^G will continue to have the meaning specified in the user's init file. This is often the HELP interrupt, which acts as shift-STOP.

Users who are accustomed to typing ^E as a soft interrupt should note that ^E moves to the end of the line. As discussed above, hitting the STOP key (or equivalently, typing ^G) accomplishes what ^E did.

Since ^H is defined to be the Backspace action in TEditKey, users cannot type ^A to erase characters even outside of TEditKey (Interlisp-D currently does not allow multiple backspace characters).

In addition to the changed functionality mentioned above (provided courtesy of TEditKey), the user should be aware of the following standard Interlisp-D/TEdit behavior:

**SAME** operates as a LooksCopy mode key. First make a selection. Now to copy the looks from some other text simply hold down the SAME key, then select the source for the looks. (Paragraph looks can be copied the same way, but by making the final selection while in the left margin. This is the standard way to select a whole paragraph in TEdit.)

**MOVE** and **COPY** act as mode keys for the selection mechanism. Thus the user can select the destination, then hold down the MOVE key and make a second selection. This selection will be moved (or COPY'd depending on the mode key used) to the (original) caret position.

**CONTROL** operates as a mode key to signal deletion. This means that holding down the CONTROL key and selecting some text will delete that text when the CONTROL key is released.

**DELETE** deletes the current selection when pressed.

## DORADO EQUIVALENTS

|                |                                    |
|----------------|------------------------------------|
| Dandelion Key: | Equivalent key on Dorado:          |
| <b>OPEN</b>    | ^o ( or ^O)                        |
| <b>SAME</b>    | META                               |
| <b>FIND</b>    | finds item in TEdit menu           |
| <b>AGAIN</b>   | ESC                                |
| <b>DELETE</b>  | DEL                                |
| <b>COPY</b>    | SHIFT                              |
| <b>MOVE</b>    | CTRL-SHIFT                         |
| <b>PROP'S</b>  | META or LOCK depending on switches |

|                   |                                                              |
|-------------------|--------------------------------------------------------------|
| <b>NEXT</b>       | #n ( or #N)                                                  |
| <b>EXPAND</b>     | ^x (or ^X)                                                   |
| <b>HELP</b>       | #?                                                           |
| <b>MARGINS</b>    | #[ (unnest (which is shift-MARGINS on the Dandelion) is #) ) |
| <b>FONT</b>       | top blank                                                    |
| <b>KEYBOARD</b>   | middle blank                                                 |
| <b>UNDO</b>       | bottom blank                                                 |
| <b>STOP</b>       | ^G                                                           |
| <b>shift-STOP</b> | #^S (intentionally difficult to type accidentally)           |

The function keys (CENTER, BOLD, etc.) are all available on the function key window brought up when TEditKey is loaded on a Dorado.

Note that the function key window can be rebuilt on a Dorado by selecting "Function Keys" in the default TEdit menu (obtained by buttoning in the title bar of a TEdit window).

## USER SWITCHES

**TEDITKEY.METAKEY** The user must choose a metakey to make use of TEditKey. The value of the variable **TEDITKEY.METAKEY** is the name of the key that will be your metakey. For instance to make TAB (the default) your metakey, (SETQ TEDITKEY 'TAB) before loading TEditKey. (Note that even in the standard system, TAB is available as Control-I).

NOTE: METASHIFT (see page 18.9 of the *Interlisp Reference Manual*) is redefined to operate on TEDITKEY.METAKEY instead of on the bottom-blank key of the Dorado.

The operation of TEditKey is controlled by the following (INITVAREd) variables:

**TEDITKEY.LOCKTOGGLEKEY** is the key that will be turned into a lock-toggle. If it is non-NIL, that key is set to act as a lock-toggle. Thus hitting this switches the case of the type-in. For those users who have removed the spring from their lock key, TEDITKEY.LOCKTOGGLEKEY is usually PROP'S. The action of LOCK is then made to be '(CTRLDOWN. CTRLUP) providing the user with a control key where LOCK is located and a lock toggle where PROP'S is located.

**TEDITKEY.FONTS** is an eight-element list of the fonts that are invoked by meta-1 through meta-8. The defaults are listed above.

**TEDIT.DEFAULT.CHARLOOKS** defines the looks that result when the DEFAULTS key is pressed or when default caret looks are requested. It is an instance of the CHARLOOKS datatype. To preset it, for instance, to TIMESROMAN 10 type the following to the Lisp top level.

```
(SETQ TEDIT.DEFAULT.CHARLOOKS (CHARLOOKS.FROM.FONT (FONTCREATE 'TIMESROMAN 10)))
```

However, a much simpler method is to select an instance of the desired looks and type shift-DEFAULTS.

**TEDITKEY.VERBOSE** if T (the default), the functions that modify the caret looks print feedback in the (TEdit) prompt window.



**TEDITKEY.NESTWIDTH** is the distance (in points) that the indent and exdent functions move the margins. Initially 36 points (0.5 inches).

**ITK.SIZEINCREMENT** is the amount (in points) which the LARGER function increases the selection (and conversely for SMALLER). Initially 2 points.

**TEDITKEY.OFFSETINCREMENT** is the amount (in points) which the SUBSCRIPT function raises the selection (and conversely for SUPERScript). Initially 3 points.

**TEDITKEY.KEYBINDINGS** is the list that controls the mapping of keys to functions for the functions that are common to the Dandelion, Dorado, and Dolphin. It consists of triples of function name, list of CHARCODE-style character specifications, and a comment describing what the function does. (The comments are used by the automated menu-building tools and their inclusion is encouraged.)

**TEDITKEY.DLION.KEYACTIONS** is the list that specifies the key actions of the non-Alto keys (to the left and right) on the Dandelion. It is the format acceptable to MODIFY.KEYACTIONS (see page 18.9 of the *Interlisp Reference Manual*).

**TEDITKEY.DLION.KEYBINDINGS** is the list specifying the functions to be tied to the characters generated from above. The keynames in the CAR of each element are comments. Note that TEDIT.DLION.KEYACTIONS and TEDIT.DLION.KEYBINDINGS must be coordinated (similarly for TEDITKEY.FNKEYACTIONS and TEDITKEY.FNKEYBINDINGS).

**TEDIT.DLION.KEYSYNTAX** is the list of syntax classes to be applied to the Dandelion keys.

**TEDITKEY.FNKEYACTIONS** is the list that specifies the keyactions of the function keys (center, bold, etc.).

**TEDITKEY.FNKEYBINDINGS** is analogous to TEDIT.DLION.KEYBINDINGS but for the function keys.

**TEDITKEY.DORADO.KEYACTIONS** are the keyactions unique to the Dorado (and Dolphin).

**TEDITKEY.DORADO.KEYSYNTAX** is analogous to TEDIT.DLION.KEYSYNTAX.

The previous variables in conjunction with the following functions specify the effect of TEditKey.

**(TEDITKEY.INSTALL *readtable*)** invokes the keyactions and bindings as specified by the above variables on *readtable*. (*Readtable* defaults to TEDIT.READTABLE).

**(ITK.BUILD.MENU)** is a function that automagically builds the help menu from the values of the above variables.

---

---

## TEdit-Line-Numbering

---

---

By: Randy Trigg (Trigg.pa)

This document last edited on May 9, 1988

### INTRODUCTION

TEDIT-LINE-NUMBERING enables the automatic conversion of a TEdit selection into multiple lines of specified width each ending in a carriage return and prefixed by a line number. For example, TEDIT-LINE-NUMBERING converted the following piece of transcript:

-----

C:    *We have to be able to check that, within the memory I, I claim.*

(1.0)

M:    *Check whether Tore is a graduate student? I think we can do that (.) I mean*

C:    *Yea I know but more we have to be able to, within the memory somehow, recognize that (.) because of this constraint being in the memory, we have to check that the time matches Wednesday morning. We have to add this constraint to the time and see if (.) the time is not overconstrained.*

-----

into:

-----

021   C:    *We have to be able to check that, within the memory I, I claim.*

022   (1.0)

023   M:    *Check whether Tore is a graduate student? I think we can do that*  
024       *(.) I mean*

025   C:    *Yea I know but more we have to be able to, within the memory*  
026       *somehow, recognize that (.) because of this constraint being in*  
027       *the memory, we have to check that the time matches Wednesday*  
028       *morning. We have to add this constraint to the time and see if (.)*  
029       *the time is not overconstrained.*

-----

### INTERFACE

(MAKE-LINE-BREAKS    *TextStream*    *WidthInInches*    *Device*    *StartLineNum*    *LineNumDigits*  
*FirstLineParaLooks*    *OtherLineParaLooks*    *InsertExtraTabFlg*    *InsertExtraCRFlg*)

TextStream should be a TEXTSTREAM of an open TEdit. Line breaks and line numbers will be inserted for the text in the current selection. WidthInInches is the number of inches wide the resulting lines will be when printed to Device. Device should be one of the litatoms DISPLAY, INTERPRESS, or PRESS. StartLineNum will be the line number used for the first line formed. LineNumDigits is the number of digits used to print the line numbers. FirstLineParaLooks should be a standard TEdit paralooks proplist for the first line formed from every paragraph. OtherLineParaLooks will be used for the remaining lines of each paragraph. InsertExtraTabFlg, if non-nil, will cause an extra tab to be inserted after the line number. InsertExtraCRFlg, if non-nil, causes an extra carriage return to be inserted between paragraphs. (If this last Flg is on, then FirstLineParaLooks and OtherLineParaLooks are probably equal.)

For example, the conversion depicted above was done with the following call (where SS is bound to an open textstream):

```
(MAKE-LINE-BREAKS SS 5 'INTERPRESS 21 3 '(LINELEADING 2 PARALEADING 5)
'(LINELEADING 2 PARALEADING 0) T)
```

The algorithm used is (pretty much) as follows: First, place a temporary marker at the end of the selection. For each paragraph in the selection, insert a line number LineNumDigits wide followed by a tab. Move one character at a time through the paragraph adding up STRINGWIDTHs until reaching WidthInInches (converted to appropriate units for Device). Then move back to last whitespace and insert a carriage return. Change the paralooks of the paragraph just formed according to FirstLineParaLooks. Insert the next line number followed by one or two tabs (depending on value of InsertExtraTabFlg) and continue adding up STRINGWIDTHs. When WidthInInches is reached, insert a carriage return and change the paralooks to OtherLineParaLooks. Continue in this manner till the end of the paragraph. If InsertExtraCRFlg is non-nil, then insert an extra carriage return. Continue with the rest of the paragraphs until encountering the marker at the end of the selection. Delete the marker and quit.

## NOTES

It's a good idea to have a couple of tabs set for the selected text, though MAKE-LINE-BREAKS will use the default tab setting if you don't.

It tries to do proper measurement of embedded tabs, but this hasn't been extensively tested.

## BUGS

IT'S INCREDIBLY SLOW! This is because we use only calls available from the TEdit programmers interface. Things could be significantly sped up by walking the piece table like TEdit does when printing, but it's alot of work to figure out how to do that and anyway I'd prefer that this tool only call advertised functions.

Doesn't handle imageobjs.

---

---

## TEDIT-PF-SEE

---

---

By:

Ron Kaplan

This document created in January 2022.

This tiny package adds alternatives to the PF and SEE commands that produce their output in scrollable read-only TEDIT windows rather than the unscrollable EXEC window. The new commands are tf (for t(edit)f(unction)) and ts for t(edit)s(ee)

`tf FUNCTION (FILELIST) (REPRINT)` [command]

prints the definitions of `FUNCTION` that appear in the files in `FILELIST`, with a separate `TEDIT` allocated for each definition. If `FILELIST` is not provided, then `WHEREIS` is used to locate the definitions, just as with `PF`. By default the definition characters are simply copied from the source file to the `TEDIT` stream, but if `REPRINT` is `T` the definition is read and then reprinted. This produces useful output for definitions that were not pretty-printed. Also, if `FUNCTION` is not provided, definitions for the last invocation will be reprinted.

`ts FILE (WINDOW)`

shows the contents of `FILE` in a scrollable read-only `TEDIT WINDOW`. This uses the function `TEDIT-SEE` (also used for the `FILEBROWSER See` command), which interprets any font changes if `FILE` is a Lisp source file.

`TEDIT-PF-SEE` loads the `REGIONMANAGER` package, and the default regions for `tf` and `ts` are of type `PF-TEDIT` and `SEE-TEDIT` respectively. The function `SET-TYPED-REGIONS` can be used to predefine the regions where the output for `tf` and `ts` will appear.

---

---

**TEdit-Process-Killer**

---

---

By: Steve Bagley (Bagley.pa) and Randy Trigg (Trigg.pa)

This document last edited on Apr 23, 1987

**INTRODUCTION**

TEDIT-PROCESS-KILLER provides a simple interface to removing and restoring the process of a TEdit window. The processes of TEdit windows can be killed selectively or a TEDIT-KILLER process can be started to keep the total number of active TEdit processes at or near some threshold level. TEdit processes are automatically rebuilt when you button in their windows.

**INTERFACE**

(KILL-PROCESS-OF-TEDIT-WINDOW *WINDOW*) [Function]

kills the processes associated with the main window of *WINDOW*, and all of the attached windows. Each process is killed in such a way that the TEdit can be restarted. It is not an error to call this function on a TEdit whose process has already been killed.

(WITHOUT-TEDIT-PROCESS *WINDOW*) [Function]

returns T if this window does not have a process, because the process was killed by KILL-PROCESS-OF-TEDIT-WINDOW, NIL otherwise.

(RESTART-PROCESS-OF-TEDIT-WINDOW *WINDOW*) [Function]

restarts the TEdit processes for the main window of *WINDOW* and all attached windows if the processes have been killed by KILL-PROCESS-OF-TEDIT-WINDOW.

(START-TEDIT-KILLER) [Function]

starts up a process called TEDIT-KILLER which wakes up at regular intervals to kill off the least recently used TEdit processes. There are two global vars available to the user to affect its operation:

TEDIT-PROCESS-LIMIT [Variable]

Defaults to 10. The preferred threshold of running TEdit processes. Every time TEDIT-KILLER wakes up, it kills off enough TEdit processes to bring the total down to this limit.

TEDIT-KILLER-WAIT-TIME [Variable]

Defaults to 10000. The time in milliseconds between wake-ups of TEDIT-KILLER.

(STOP-TEDIT-KILLER) [Function]

kills any running TEDIT-KILLER process.

**NOTES**

In order to force a TEdit to be killed off when shrunk, simply do

```
(WINDOWADDPROP <Win> 'SHRINKFN (FUNCTION KILL-PROCESS-OF-TEDIT-WINDOW))
```

and, if you like,

```
(WINDOWADDPROP <Win> 'EXPANDFN (FUNCTION RESTART-PROCESS-OF-TEDIT-WINDOW))
```

## **BUGS**

We don't kill lafite sendmessage processes.

---

---

**TEKTRONIX**

---

---

By: Jim Blum (Unsupported)

Last Modified: James Turner (Turner:Lexington:Xerox)

TEKTRONIX 4010 EMULATOR

FILES: TEK4010CHAT, TEK4010

This document last edited on: 9-May-88 23:52:18

There are two LispUsers Modules, TEK4010 which takes an INSTREAM and OUTSTREAM as arguments and emulates a TEKTRONIX 4010 storage tube terminal, and a version called TEK4010CHAT which works with CHAT. The details on how and what a TEKTRONIX 4010 terminal does are described in a TEKTRONIX 4010 users manual. The CHAT program is described in the Interlisp Reference Manual. This document will point out the differences from the default DM2500 emulated terminal which CHAT uses, and the relevant issues unique to running the TEK4010 emulator under Interlisp.

TEK4010 is called as follows:

(TEK4010 *INSTREAM OUTSTREAM*)

[Function]

where OUTSTREAM must be a displaystream (or window). This version does not fully support the TEKTRONIX 4010 terminal. Specically, the graphic input mode (which displays the crosshair and waits for a key to be typed), and does not support the double column text mode. This version supports a limited scaling feature. Both the TEKTRONIX 4010 X and Y coordinates are divided by one global integer called TEKPTSPERPOINT which can be set(q) by the user. No attempt is made to scale the text or line spacing (leading) in this version, nor does reshaping the window automatically change TEKPTSPERPOINT.

The CHAT version (TEK4010CHAT) unlike the DM2500 emulator, runs in non-scroll, paint mode, with the right margin set at the width of the window. Although not elegant, this is how the TEK4010 terminal works and there may be cases where users are dependent on its inherent mode of operation. The visible implications of this on INTERLISP are: The screen is not cleared or scrolled when a linefeed is received on the last line. Graphics and text are overlayed (OR'd) using the PAINT mode in BITBLT, since selective areas on a storage tube cannot be erased. Graphics are scaled to the window size and so is the position of the second margin, but the characters in the font (GACHA 10) are not scaled. Scaling is changed when the CHAT window is reshaped. In order to get accurate positioning of both the text and graphics, the Interlisp window size should match the TEK4010 screen resolution which is 1024 horizontally by 768 vertically.

In order to run the TEK4010 emulator under CHAT, the following recipe must be followed:

1. Make sure you have the latest version of CHAT loaded.
2. Load the TEK4010 emulator module, ie, (LOAD 'TEK4010CHAT.LCOM)
3. Edit the variable CHAT.DISPLAYTYPES (ED 'CHAT.DISPLAYTYPES) to add one or more lists of the form (host number TEK4010) where host is the name of the host that you want to chat to with this emulator. Use the number 4010 for the number field which would be used by the CHAT.SETDISPLAYTYPE function. Add as many entries as there are hosts you want to use the TEK4010 emulator with.
4. Then bug CHAT in the background MENU and select or enter the host that you want to chat to.



---



---

## TILED-SEDIT

---



---

By: Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester)

This document last edited on: September 23, 1987

### SUMMARY

TILED-SEDIT is a facility for automagically positioning SEdit windows according to a specified pattern. SEdit windows appear in any of the four corners of the screen, with overlapping windows slightly offset so they can still be brought to top (by clicking on them). Users can specify which corners, in what order, how thick a margin around the screen, and the size of the offset.

### DESCRIPTION

(TILED.SEDIT.RESET *Tiling-Order XShift YShift Screen*) [Function]

If *Tiling-Order* is NIL, this resets the SEdit window tiling facility, and SEdit reverts back to its old behavior (i.e., prompting for a window region). Otherwise *Tiling-Order* should be either T or a keyword or an arbitrarily long list of keywords from the following set { :TL :TOP-LEFT :TOP.LEFT :TOPLEFT :BL :BOTTOM-LEFT :BOTTOM.LEFT :BOTTOMLEFT :TR :TOP-RIGHT :TOP.RIGHT :TOPRIGHT :BR :BOTTOM-RIGHT :BOTTOM.RIGHT :BOTTOMRIGHT }. If *Tiling-Order* is T, the list (:TL :BL :TR :BR) is assumed. SEdit will place new windows in the corners specified by *Tiling-Order* (which is indefinitely repeated if necessary).

If a new SEdit window would overlap an existing SEdit window, the new one is offset by *XShift* pixels right and *YShift* pixels down. *XShift* and *YShift* default to 15. Tiled.SEdit will compute the tile size and placement on the basis of the region *Screen* such that you can go three times through the default four corner loop before the right or bottom windows start crossing the edge of *Screen*. If *Screen* is neither a region nor a fixp, *Screen* defaults to 25. If *Screen* is a fixp M, *Screen* is assumed to be (CREATEREGION M M SCREENWIDTH-M SCREENHEIGHT-M). The default setting leaves room enough for a scrollbar on the left and the bottom.

Invoking TILED.SEDIT.RESET with a non-NIL *Tiling-Order* will cause all currently open SEdit windows to be repositioned according to *Tiling-Order*.

### EXAMPLES

(TILED.SEDIT.RESET T) [Function]

This is executed when you load TILED-SEDIT. It provides for automatic SEdit window creation in the corners TopLeft, BottomLeft, TopRight, BottomRight, TopLeft, BottomLeft, ... Each time around the loop windows are shifted 15 pixels to the right and downward. A 25 pixels margin is preserved at the left and bottom edge of the screen.

(TILED.SEDIT.RESET :TL) [Function]

This causes SEdit to create windows in the TopLeft corner only. Each new window is shifted 15 pixels to the right and downward. A 25 pixels margin is preserved at the left and bottom edge of the screen.

(TILED.SEDIT.RESET '(:TR :BR) NIL 35)

[Function]

This causes SEdit to create windows in the TopRight and BottomRight corners only. Each time around the two corner loop windows are shifted 15 pixels to the right and 35 pixels downward. This has the advantage that the title of each SEdit window remains visible, but the disadvantage that each window is smaller. A 25 pixels margin is preserved at the left and bottom edge of the screen.

#### **CAVEAT**

TILED.SEDIT.RESET is independent of SEDIT.RESET. It will not invoke SEDIT.RESET, nor does it require that all SEdit windows are closed prior to invocation. It is strictly used for controlling the window tiling.

---

---

## TKDORADO

---

---

By: Mike Dixon (MikeDixon.pa)

INTERNAL

### **TKDorado: TEditKey for the Dorado keyboard**

TKDorado makes the full range of TEditKey commands available from the Dorado keyboard. To do so it adds TEditKey bindings for a number of meta-control key combinations:

|                          |                                    |
|--------------------------|------------------------------------|
| Meta-Control- <b>V</b> : | set to default (vanilla) looks     |
| Meta-Control- <b>B</b> : | bold on                            |
| Meta-Control- <b>N</b> : | bold off                           |
| Meta-Control- <b>I</b> : | italics on                         |
| Meta-Control- <b>O</b> : | italics off                        |
| Meta-Control- <b>D</b> : | overbar on                         |
| Meta-Control- <b>F</b> : | overbar off                        |
| Meta-Control- <b>G</b> : | strikethru on                      |
| Meta-Control- <b>H</b> : | strikethru off                     |
| Meta-Control- <b>J</b> : | underlining on                     |
| Meta-Control- <b>K</b> : | underlining off                    |
| Meta-Control- <b>[</b> : | smaller font                       |
| Meta-Control- <b>]</b> : | larger font                        |
| Meta-Control- <b>↑</b> : | superscript                        |
| Meta-Control- <b>_</b> : | subscript                          |
| Meta-Control- <b>C</b> : | center/justify/left this paragraph |

The functions which change character looks operate on the current selection, if any characters are selected, and otherwise change the caret looks.

The new key bindings are also added to the help menu.

In addition, TKDorado rebinds the ESC key to be REDO, rather than Expand (which is still available as Control-X), and closes the DLion keys window (if TEditKey has left it lying around).

---

---

# TMAX

---

---

## Chapter 1. Introduction

TMAX stands for Tedit Macros And eXtensions and it enhances TEdit by providing a convenient way to do things such as numbering, indexing, creating a table of contents, and more. At SUMEX these sort of operations are done with Scribe<sup>1</sup>. Scribe is a powerful document preparation language but it consumes all together too many cycles on our mainframe. Furthermore with Scribe you must hardcopy your document to see what it looks like. If you want to change the format, you must add the appropriate commands to the Scribe input file, run Scribe again, and hardcopy the output file. This sort of "batch processing" wastes both yours and the mainframe's time. TEdit is a WYSIWYG (What You See Is What You Get) text editing and formatting system. You see what your document will look like while you are creating it.

TMAX makes no attempt to mimic Scribe in TEdit nor does it have any facility to translate a Scribe .MSS (i.e. source) file into a TEdit file. Rather it implements some of the more commonly used features of Scribe in TEdit. Currently there are four main areas in TMAX; indices, numbering, endnotes, and forward and backward references. The TMAX features described here plus the editing and formatting features already available in TEdit make it an attractive alternative to Scribe. For more information on TEdit, please read the first part of the **Text Editing** section in **The Lisp Library Packages Manual**.

All the features described here are merely additions to a TEdit document. TMAX does not alter your text in any way. When you invoke one of these features, TMAX inserts a "special character" (i.e. an Image Object<sup>2</sup>) into your document at the current cursor position. These "special characters" may appear to be strings but they are really just single characters. This means you can delete any TMAX feature you add by simply deleting the corresponding "special character" just as you would delete any other character. The features described in this document are also used throughout this document. Rather than including pictures of each menu (and there are lots of them), we have decided to show you how to pop-up these menus yourself. This document is both an explanation and example of how to use TMAX. We suggest you read this document under TMAX/TEdit and try these features as you read about them. This document was written assuming you are reading it in a TEdit window.

## Chapter 2. Menus

When you first load TMAX.DCOM, a new item called **TMAX Menu** is added to the main TEdit pop-up menu. Since you probably used this menu to **Get** this document, you may have already seen this new item. Buttoning **TMAX Menu** brings up the TMAX menu which is attached to the top of your TEdit window. Now you can invoke the TMAX features by simply buttoning items in this menu. You can remove this window by right buttoning the mouse in its title bar and selecting Close (just as you would remove any other TEdit menu). The small window just above this window is called the "prompt window". TMAX uses this window both to prompt you for text input and to report current values and status. Please pop up the TMAX Menu now by pointing the mouse at the black bar above this window, left or middle buttoning it, and selecting **TMAX Menu**.

There are three types of fields in the TMAX menu; labels, values, and commands. You select items by pointing the mouse at the item and pressing either the left or middle button.

- 1) The fields on the far left of each line (Miscellany:, References:, etc.) are simply labels that specify the nature of the items on the rest of that line. You cannot select these label fields.
- 2) There are eight items that are used to set values. Five of the items are toggle switches, two define output filenames, and one pops up a menu of possible settings.
  - 2.a) The five toggle switches are ***Set AutoUpdate***, ***NGroup Menu***, ***Text Before***, ***Text After***, and ***Manual Index***. They are all in a bold italic font to distinguish them from other items. These 2-way toggle switches appear in normal video (as they are

- above) when they are off and in reverse video when they are on. Buttoning these items complements their current setting.
- 2.b) **TOC Filename:** and **Index Filename:** are used to define the name of the Table-Of-Contents and Index file respectively. When these items are buttoned, the cursor appears to the right of the colon and TMAX waits for you to specify a filename. On a **Get** these items are defaulted to the name of the input file with the extensions .TOC and .INDEX.
  - 2.c) The last value item is **Reference By** and the light face field following it is its current setting. Buttoning this item pops up a menu of possible settings.
  - 3) All the other items (everything except labels and values) are commands. Buttoning these items invokes the corresponding TMAX feature. These items turn to reverse video while they are active and then return to normal video when they finish.

Many of the TMAX features use pop-up menus. Buttoning the mouse outside any pop-up menu is equivalent to no selection and will make that menu disappear. Whenever TMAX prompts you for a new value, it always displays the current value in the prompt window.

We will now discuss each item in the TMAX menu in detail. Since many of the items interact with other items, it is difficult to find a starting point. Instead we have described the TMAX menu row by row and left to right within each row. We suggest you first peruse this document to get a general idea of the features and then focus on the areas that interest you.

## Chapter 3. Miscellany

### 3.1 Update

When you use TMAX to do some sort of numbering, TMAX inserts a "marker" rather than the actual number. It does this for speed since it is much faster to insert a marker rather than figure out the corresponding numeric value each time. Buttoning **Update** will convert all these markers to their corresponding (and consecutive) numeric values. In addition, any References (see chapter 4) to these numbers will be updated. Finally, if the Endnotes (see chapter 5) have already been inserted, **Update** will reinsert them but this time with the actual Endnote numbers. Currently the only numbering constructs in TMAX are Endnotes, Number Groups, and References to them. Please button **Update** now and watch all the changes.

### 3.2 Undo Update

Buttoning this item undoes everything that **Update** does. That is it converts all the Endnote and Number Group numbers back to their corresponding markers along with any References to these numbers. If the Endnotes have already been inserted, **Undo Update** will reinsert them but this time with the Endnote numbers replaced by their markers.

### 3.3 Set AutoUpdate

When this toggle switch is on, all numbering markers are immediately updated to their corresponding numeric value as they are inserted. Note that it only updates inserted markers; it does not automatically update any markers in a TEdit file that you load via **Get**. We do not encourage setting this switch for large documents. This switch causes TMAX to check every number it generated in the entire document whenever a new number is inserted. For large documents this could take some time.

### 3.4 Current Date/Time

Buttoning this item inserts the current date into your document. For example, the date enclosed in the following parentheses (April 22, 1987) was inserted by buttoning **Current Date/Time**. Middle buttoning this inserted date pops up a Date/Time menu. If you'd like to see this menu, go ahead and middle button the date in the parentheses. This menu allows you to change the format of the date/time, replace the date with the time, or update an old date/time to the current date/time. The font is the same as the font currently in effect in your TEdit document when the date/time was inserted.

## Chapter 4. References

These commands allow you to reference Endnotes and Number Groups by either their numeric value or by the page number they appear on. A Reference is simply an association between a "Tag" and a reference to that Tag. To assign a Tag, middle button the Endnote or Number Group

marker (or its numeric value if it has already been updated). A menu will pop up and one of the items will allow you to define a Tag for the Endnote or Number Group. If the Endnote or Number Group is already tagged, the menu will allow you to change or delete the Tag. All the Tags in a document must be unique and TMAX will not allow you to create a Tag that is already defined. If you COPY a tagged Endnote or Number Group within the same window, TMAX will remove the Tag on the copied object. If you COPY a tagged object to another TEdit window, TMAX will preserve the tag assuming that Tag name isn't defined in the other TEdit window. The font of a Reference is the same as the font currently in effect in your document regardless of what font is used to display the Endnote or Number Group.

As an example, we have tagged both the Endnotes chapter (a Number Group on page <5/Page>) with "EN Chap" and the Endnote at the end of this sentence with "EN Note"<sup>3</sup>. Now we can reference the Endnotes chapter as 5 and the previous Endnote as 3. Also the page number of the Endnotes chapter above was generated by a Reference to "EN Chap" by page instead of by value. You will have to button **Update** to convert the "<Chapter/Value>" and "<Note#/Value>" to their respective numeric values. Of course, if **Set AutoUpdate** was on, the "<Chapter/Value>" and "<Note#/Value>" would be converted as soon as they were inserted.

#### 4.1 Reference

TMAX will prompt you for a Tag name when you button this item. A bare carriage return cancels this command. It doesn't matter if the Tag you specify is defined yet or not. If the Tag is not defined, the Reference marker is "<Reference Tag/Type>" where Tag is the Tag name and "Type" is either Value or Page. In this case the Tag name is embedded in the marker. If the Tag is defined, the Reference marker is either <Note#/Type> for Endnotes or the Number Group marker enclosed in angle brackets (e.g. <Chapter/Type>). In this format there is no indication of what the Tag name is. If you middle button a Reference marker (whether it is updated or not), TMAX will display the corresponding Tag in the prompt window. All Reference markers that have not been updated are enclosed in angle brackets (i.e. < and >) to distinguish them for normal text.

#### 4.2 Previous References

Buttoning this item brings up a menu of all the defined Tag names in alphabetical order. You can then create a Reference to one of these Tags by simply buttoning the corresponding Tag in this menu. Although it doesn't matter if you define the Tags or References to these Tags first, we suggest you define the Tags first. Once the Tags are defined you can use this command to easily create the References to these Tags. If you button **Previous References** now you will see all the Reference Tags defined in this document.

#### 4.3 Reference By

This item allows you to select whether you want References by numeric value (of the Endnote or Number Group) or by the page number (on which the Endnote or Number Group appear). Buttoning this item pops up a menu with three items; Ask, Value, and Page. If you select Value or Page then all References you add will be of the type you specified. If you select Ask then TMAX will prompt you for the type every time you make a Reference. As you can see in the TMAX menu above, the default Reference type is by Value.

There is one minor inconvenience with doing forward references by page. You must do a dummy hardcopy and then button **Update** before the real hardcopy. The reason is TEdit only knows the current page number while it is hardcopying a document. As TMAX encounters Endnotes and Number Groups, it saves their corresponding page numbers. If you reference an Endnote or Number Group that TMAX hasn't seen yet then it can't possibly know the corresponding page number. In this case, TMAX will change the Reference marker to the Endnote's or Number Group's numeric value followed by "/Page" both enclosed in angle brackets (e.g. <1.2/Page>) to indicate that the page number is not known yet. This only applies to forward references by page. You don't have to do a dummy hardcopy if you are referencing by value or doing backward references by page.

## Chapter 5. Endnotes

These commands allow you to generate a numbered list of notes. Endnotes are like footnotes except the numbers and corresponding text appear at the end of the document rather than the bottom of the page. (TMAX does not support footnotes yet.) The Endnote numbers are consecutive and always start at 1. If you add an Endnote in the middle of a document, all the following Endnote numbers will be adjusted automatically. Suppose for example you have three Endnotes and you add another between Endnotes 1 and 2. The new Endnote becomes 2 and the Endnotes that were formerly

2 and 3 are now 3 and 4. This will happen when you button **Update** (or immediately if **Set AutoUpdate** is on). The font of the Endnote numbers, text, and title line is determined by the **Set Style** command and the default font is Gacha 10 Standard.

If you middle button an inserted Endnote marker, a menu will pop up allowing you to define a Tag for this Endnote marker or edit the text associated with this Endnote marker. If this Endnote already has a Tag, this menu will give you the option of changing, deleting, or displaying the Tag or editing the Endnote text. If you choose to edit the Endnote text, you will be prompted to open another TEdit window where the text will be displayed. When you are done editing the text, move the mouse to the title bar (i.e. the thick black bar at the top of the new TEdit window) and left or middle button the mouse. A menu will pop-up giving you the option of saving or aborting the changes. There is a sample Endnote at the end of this sentence you can play with<sup>4</sup>.

### 5.1 Endnote

Buttoning this item inserts an Endnote marker at the current cursor position. The Endnote marker is a superscript "Note#". You will also be prompted for the text associated with this Endnote.

### 5.2 Insert Endnotes

This item inserts the title line "Notes" followed by the Endnote numbers and their corresponding text after the last line in your TEdit document. If you have already inserted the Endnotes and button this command again, TMAX will delete the old Endnotes and then reinsert them. It does this in case you have added or deleted any Endnotes since the last time they were inserted.

### 5.3 Delete Endnotes

Buttoning this item undoes what **Insert Endnotes** does. It deletes the title line "Notes" and the Endnote numbers and text that follow from the end of your document.

If you already inserted the Endnotes and wish to add more text at the end of your document, you *must* first delete the Endnotes with this command. If you don't, the additional text at the end of your document will be deleted when the Endnotes are reinserted. Suffice it to say this is due to the way Endnotes are implemented rather than a bug in TMAX. A future version of TMAX will fix this awkwardness.

### 5.4 Set Style

Buttoning this item pops up the Endnote Fonts menu. This menu has three items; Number, Title, and Text. You can use this pop-up menu to change the font of the Endnote numbers, title line, and text. The title line is always "Notes". When you select one of the items from this menu, the current font of that item will be displayed in the prompt window before TMAX prompts you for the new font.

## Chapter 6. Number Groups

The Number Group (NGroup) commands allow you to number any arbitrary objects in your TEdit document. You define the hierarchy, font, and format of each NGroup member and then insert them wherever you want something numbered. The NGroup numbers are consecutive and start at 1 by default although you can change the starting value. This allows you to number individual pieces of a document without having to load the entire document. Some NGroup values depend on other NGroup values. For example, the sections of this document each begin with the corresponding chapter number. Each time the chapter number changes, the section number is reset to its starting value. You can nest NGroups as deep as you like. If you insert a new NGroup member in the middle of a document, all the following NGroup members that depend on this new member will be adjusted when you button **Update** (or immediately if **Set AutoUpdate** is on). This makes it trivial to number (and renumber) things. All you do is insert an NGroup member wherever you want something numbered and TMAX takes care of all the rest. The default NGroup font is Gacha 10 Standard.

### 6.1 NGroup Menu

When this toggle switch is on, the NGroup menu "graph" is displayed in a window attached to the top of the TMAX menu window. The Number Group graph is a tree structure showing the hierarchy and order of all the Number Group members. If you haven't buttoned **NGroup Menu** yet, please do so now.

## 6.2 New NGroup

This item allows you to create new Number Group members. When it is buttoned, TMAX first turns on **NGroup Menu** if it was off so you can see the NGroup graph. It then prompts you for the name of the NGroup member. Finally TMAX prompts you for the parent of this new NGroup member by popping up a menu of the known NGroup names and you select the parent from this menu. By parent we mean the node to the immediate left in the NGroup graph. If you don't select any parent (i.e. button the mouse outside the pop-up menu) the new NGroup becomes a "top level" node. A top level node is one whose parent is the boxed **NGroups** (at the far left). The NGroup graph is always built from left to right. Since the first NGroup member defined will have no parents, TMAX just adds this member to the graph. You can create as many NGroup members as you like even if you don't use them all. As long as there is at least NGroup in your document, the entire NGroup graph will be saved/restored over a **Put/Get**.

## 6.3 Text Before

When this toggle switch on, you will be prompted for a preceding text string each time you insert a NGroup member. This text string becomes part of the inserted NGroup marker. You can use this string as a heading for the chapter, section, figure, example, etc. you have numbered. The text string is always printed in the same font as the corresponding NGroup member and TMAX always inserts one space between the preceding text string and the NGroup marker. If you would like more spaces, you must put them in the **Text Before** string yourself. Any tabs in this string are automatically converted to spaces. The default **Text Before** string is the name of the NGroup. In this document all the Chapter NGroup markers were inserted with **Text Before** on and the Section NGroup markers were inserted with **Text Before** off.

## 6.4 Text After

When this toggle switch on, you will be prompted for a succeeding text string each time you insert a NGroup member. This text string becomes part of the inserted NGroup marker. You can use this string as a heading for the chapter, section, figure, example, etc. you have numbered. The text string is always printed in the same font as the corresponding NGroup member and TMAX always inserts one space between the NGroup marker and the succeeding text string. If you would like more spaces, you must put them in the **Text After** string yourself. Any tabs in this string are automatically converted to spaces. There is no default **Text After** string. In this document all the Chapter and Section NGroup markers were inserted with **Text After** on.

## 6.5 Changing the default Font and Format

Consider the Number Group graph in this document. The boxed **NGroups** at the far left is a special node and buttoning it does nothing. The order of the nodes in a branch is important but the order of the branches themselves isn't. If you would like to see a more complex NGroup graph then just add some more members to this graph. Don't worry; adding extra NGroup members doesn't affect anything.

When you create NGroup members by buttoning **New NGroup**, you are actually defining "prototype" NGroups. The font and format of these prototypes determine the font and format of the NGroups you insert into your document. To change the font and/or format of a prototype NGroup member, simply point the mouse to the member name in the Number Group menu graph and **middle** button it. A menu will pop up with four items; Change Font, Show Font, Change Format, and Show Format. Buttoning Show Font or Show Format will display the font or format of the selected prototype NGroup in the prompt window.

Buttoning Change Font or Change Format allows you to change the font or format of the selected prototype NGroup. This is a *global* change. If you change the font/format of a prototype NGroup, TMAX will apply that change to every occurrence of that NGroup in your document. Before TMAX prompts you for the new font/format, it always shows you the current font/format in the prompt window. Change Font and Change Format both have subitems that allow you to change part of the font/format without changing anything else.

The subitems for Change Font are Family, Size, and Face. If you button one of these subitems, TMAX will pop up the corresponding menu and only that part of the font will be changed. If you button Change Font instead, TMAX will prompt you for all three values via pop-up menus.



There are seven subitems for Change Format. You can change any part of the format by selecting one of these subitems. If you button Change Format instead, TMAX will prompt you for all seven values. Six of the values use pop-up menus and the other requests input in the prompt window. The seven parts of the prototype format are:

- i: Delimiter Before  
This is the delimiter that precedes the NGroup. TMAX pops up a menu of commonly used delimiters. You can select one of these or select Other in which case TMAX will prompt you for a delimiter string in the prompt window. The default Delimiter Before is a null string.
- ii: Display Type  
This is how the NGroup's numeric value is displayed. TMAX pops up a menu of the various ways a NGroup can be displayed. These ways are as an Arabic numeral, a null string, or an upper/lowercase letter or Roman numeral. The default is as an Arabic numeral.
- iii: Delimiter After  
This is the delimiter that follows the NGroup. It uses the same mechanism as the Delimiter Before. The default Delimiter After is a period (i.e. ".").
- iv: Abbreviate Level  
Normally a NGroup's value is the concatenation of all its parents values plus its own value. This subitem allows you to specify how far up the NGroup branch to go when computing a NGroup's value. TMAX pops up a menu containing this NGroup name and all its parents. You control how much of the NGroup value to display by selecting the first NGroup to be used in this NGroup's value. Note that this subitem doesn't actually change the NGroup value; it only determines how much of this value to display. Since top level NGroups only have a single value, you cannot abbreviate them. The subitems listed here are numbered with the NGroup "Sect Cntr" which as been abbreviated such that it only prints its own value without any of its parents values.
- v: Starting Value  
This is the starting value of the NGroup. TMAX prompts for the new value in the prompt window rather than using a menu. The default Starting Value is 1.
- vi: Table-Of-Contents  
This is a flag that says whether or not this NGroup will be included in the Table-Of-Contents file should you decide to create one. The default is to include all NGroups in the Table-Of-Contents file.
- vii: Manual Index  
This is a flag that says whether or not this NGroup should be included in the manual-style index. Note that this item is only offered if the **Manual Index** (see section 8.4) toggle switch is on. The default is to not include any NGroups in the manual-style index.

## 6.6 NGroup Delimiters

The Delimiter Before/After construct allows TMAX to support any numbering format. Since the default Delimiter Before is a null string and the default Delimiter After is a period, the default number format is of the form "1.", "1.2.", "1.2.3." etc. You may have noticed that this document uses this default form for the Chapter numbers but not for the Section numbers. To see how we did this just middle button Chapter in the NGroup graph above and select the Show Format item from the pop up menu. Then do the same for Section. The field following "Display=" consists of the Delimiter Before, the Display Type, and the Delimiter After in that order.

There is one small anomaly with delimiters, namely the case where one NGroup has a Delimiter After (e.g. Chapter) and the following NGroup has a Delimiter Before (e.g. Section). If the Delimiter Before is not a null string then it will *always* override the Delimiter After in the preceding NGroup regardless of what the Delimiter After is. For example, if the Delimiter After for Chapter was a colon and the Delimiter Before for Section was a dash then chapter numbers would look like "1:" and section numbers would look like "1-2."

## 6.7 Inserting Number Groups

You create NGroup members by buttoning **New NGroup**. To insert an NGroup member into your document, simply point the mouse to the appropriate name in the Number Group menu graph and **left** button it. The NGroup member name enclosed in square brackets (e.g. [Chapter]) will be inserted

at the current cursor position in whatever font you have specified for this member. All NGroup markers that have not been updated are enclosed in square brackets to distinguish them from normal text. TMAX will warn you if you insert NGroups out of order. For example, in this document you should not insert "Section" until you insert "Chapter" because the value of Section depends on the value of Chapter.

## 6.8 Customizing inserted Number Groups

Although the font/format of the prototype NGroup determines the font/format of the inserted NGroup, you can change the font and certain parts of the format after the NGroup is inserted. Any change you make is *local* to that particular NGroup; it does not affect any other NGroups. If you middle button an inserted NGroup, a menu will pop up allowing you to define a Tag and show/change the font and format of the selected NGroup. If the NGroup already has a Tag, this menu will give you the option of changing, deleting, or displaying the Tag. The Show Font and Change Font items work exactly the same as those for the prototype NGroups but the Show Format and Change Format are slightly different. The Starting Value and the Table-of-Contents and Manual Index flags can only be set for the prototype NGroup. You cannot change these for a particular inserted NGroup.

There are six subitems for Change Format. You can change any part of the format on an inserted NGroup by selecting one of these subitems. If you button Change Format instead, TMAX will prompt you for all six values. Four of the values use pop-up menus and the other two request input in the prompt window. The six parts of an inserted NGroup's format are:

- i: Delimiter Before  
This works exactly the same as the Delimiter Before subitem for the prototype NGroups except the change doesn't affect any other NGroups.
- ii: Display Type  
This works exactly the same as the Display Type subitem for the prototype NGroups except the change doesn't affect any other NGroups.
- iii: Delimiter After  
This works exactly the same as the Delimiter After subitem for the prototype NGroups except the change doesn't affect any other NGroups.
- iv: Abbreviate Level  
This works exactly the same as the Abbreviate Level subitem for the prototype NGroups except the change doesn't affect any other NGroups.
- v: Text Before  
This allows you to add, change, or delete the text string preceding the NGroup regardless of the setting of the **Text Before** toggle switch. TMAX will prompt you for the new string in the prompt window. A bare carriage return deletes the text before the NGroup. If you do specify a string, remember that TMAX will always append a space to the end of the string. Unlike inserting a NGroup with **Text Before** on, there is no default string.
- vi: Text After  
This allows you to add, change, or delete the text string following the NGroup regardless of the setting of the **Text After** toggle switch. TMAX will prompt you for the new string in the prompt window. A bare carriage return deletes the text after the NGroup. If you do specify a string, remember that TMAX will always append a space to the beginning of the string.

## 6.9 Pruning the NGroup graph

As mentioned before, the entire NGroup graph is saved/restored over a **Put/Get** even if some of the NGroup members aren't used in the document. We have described how to add members to this graph but not how to delete them. TMAX has two ways to delete unused NGroup members from the graph.

The first way is to Copy text (including the inserted NGroups) from one TEdit window to another. TMAX will only copy enough of the NGroup graph to handle the NGroups in the copied text. For example, you can remove every unused NGroup member from the graph by copying the entire document to another TEdit window.

The other way is to set the global variable TMAX.PRUNE.NGRAPH to T, open a TEdit window, and then **Get** the document. On a **Put** TMAX writes out the NGroup graph data structure. On a **Get**, TMAX checks this flag and if it is true, it creates the NGroup graph from the NGroups that appear in the

document rather than from the NGroup graph data structure. There is a potential problem with using this feature. TMAX creates the NGroup graph from the first occurrence of each NGroup in the document. If the first occurrence of a particular NGroup has been modified (see section 6.8) then TMAX will use this modified format as the new prototype format. There are no problems if the first occurrence of each NGroup has not been modified.

## Chapter 7. Contents File

One of the benefits of using NGroups is the Table-Of-Contents (TOC). By default all NGroups are included in the TOC along with their corresponding **Text Before** and/or **Text After** strings if any. If you don't want certain NGroup members included in the TOC, you can specify this by changing the NGroup's prototype format. Although this document has several NGroups, we have decided to include only the Chapter and Section NGroups in the TOC. The page numbers in the TOC are always printed in the Gacha 10 Standard font.

### 7.1 Create TOC

Buttoning this item creates a TEdit Table-Of-Contents file. You must first specify the name of the TOC file via the **TOC Filename:** item. Each line in the TOC consists of the **Text Before** string (if any), the NGroup number, the **Text After** string (in any), a dotted leader, and the page number on which this NGroup appears. Note that the TOC file itself does not contain any TMAX features. It is a simple TEdit text file.

### 7.2 View TOC

Buttoning this item first creates the TOC file (via **Create TOC**) and then prompts you for a TEdit window where TMAX displays the TOC file it just created.

### 7.3 TOC Filename:

This item allows you to specify the name of the TOC file. When this item is buttoned, the cursor will appear just to the right of the colon. You then type the name of the TOC file and terminate it with a carriage return. You can edit this filename string anytime. If you have already terminated the string, just button the item again. The editing features available here are the same as those available in the EXEC window. If you **Get** a TEdit/TMAX file, the **TOC Filename:** defaults to that file with a **.TOC** extension.

NOTE... You must hardcopy your document before creating the TOC file. The reason is TMAX needs the page numbers for the TOC file but TEdit only knows the page numbers while the document is being hardcopied. During the hardcopy process, TMAX saves the page number for each NGroup member. If you create the TOC without first hardcopying your TEdit document, the page numbers in the TOC will be NIL.

## Chapter 8. Indices

TMAX allows you to insert index requests into your document and create a sorted file of these indices including the page number each index appears on. There are two types of index requests; simple and extended. The format of the index marker depends on the type of index request. But, regardless of the type of index, it is important to note that the index markers are only displayed in the TEdit window. If you hardcopy the TEdit document, you will **not** see these index markers. TMAX always encloses index requests in curly braces (i.e. { and }) to distinguish them from normal text. **Update** and **Set AutoUpdate** have no effect on index requests.

### 8.1 Index

When this item is buttoned TMAX will prompt you for the index "Key". It will then insert the marker "{Index key}" into your document at the current cursor position. TMAX uses these keys to sort the indices and the sorting is case independent. The key is also printed in the Index file along with the page number(s) it appears on. Currently all simple indices are printed in the Gacha 10 Standard font. For example, we have indexed the phrase "Indexing requests" in the following parentheses (). Of course you will not see this index marker if you hardcopied this document. You can change the index key by middle buttoning the index marker. TMAX will bring up a one item menu (Change Index). If you button this item, TMAX will prompt you for the new index key.

### 8.2 Extended Index

This is a fancy form of indexing. When this item is buttoned TMAX first prompts you for the key to sort on. TMAX then prompts you for the "Entry" and its font. This is what is printed in the Index file instead of the key. If you do not specify an entry, it defaults to whatever the key is and the font defaults to Gacha 10 Standard. Finally, TMAX prompts you for the index page number option. There are three options; print the normal page number, print a fixed page number that you supply, or don't print any page number at all. TMAX then inserts the index marker "{Index Key=key, Entry=entry, Option}" into your document. The option is "Yes" if the page number is to be included in the index file, "No" if the page number is not included, or the numeric value if a fixed number is to be used in the index file. For example, we have (extended) indexed the word "Spies" but we want "Boris & Natasha" printed instead in the Helvetica 12 Italic font. The extended index is enclosed in the following parentheses (). Of course you will not see this extended index marker if you hardcopied this document. You can change any of the fields in an extended index request by middle buttoning the index marker. TMAX will bring up a one item menu (Change Extended Index). If you button this item, TMAX will prompt you for the new index key, entry, font, and number values.

### 8.3 Known Indices

Buttoning this item brings up a menu of all the indices and extended indices specified so far in alphabetical order. You can insert another **Index** or **Extended Index** request by simply buttoning the appropriate item in this pop-up menu. This makes it trivial to index the same items throughout a document. Indices are simple items but extended indices have subitems because several extended indices can have same key but completely different entries, fonts, and/or page number options. To insert an extended index, you must button the appropriate subitem; buttoning the extended index item has no effect. The extended index subitem shows the entry, font, and page number option. If you button **Known Indices** above, you will see the two indices used here as examples.

### 8.4 Manual Index

When this toggle switch is on, the page numbers in the index file are printed in "manual format". By manual format we mean something like "III:25.7" for chapter 3, section 25, page 7 (assuming the chapter's format has been changed to print Roman numerals following by a colon). You specify which NGroup members are included in the manual index page number. To include a NGroup member in the manual format page number, first make sure this toggle switch is on. Then change the format of the NGroup members you want included. The code to change the NGroup format checks the **Manual Index** switch setting. If the switch is on, the code asks if you want the selected NGroup member included in the manual index. You can only include the members of one major branch of the NGroup graph in the manual index page numbers. Using this document's NGroup graph as an example, you can include either or both members in the Chapter-Section branch **or** the single member in the NG Format branch but **not** both. TMAX will only allow you to include an NGroup member in the manual index if there are no other members included yet or the included members are in the same branch as the new member. The reason is there is no correlation between the numbers in disjoint branches in the NGroup graph. If you don't specify manual indexing, TMAX defaults to "book format" indexing. With book format the index file page references are just the page numbers themselves.

## Chapter 9. Indices File

These commands allow you to write out an index file sorted by the index keys. As we mentioned before, the sorting is case independent. The page numbers (for both "book" and "manual" style) are always printed in the Gacha 10 Standard font.

### 9.1 Create Index

Buttoning this item creates a TEdit sorted index file. You must first specify the name of the index file via the **Index Filename:** item. For simple indices each line in the index file consists of the index key followed by the page number(s) on which this index key appears. Extended indices are treated a little differently. Each extended index is printed on a separate line. Note that the index file does not contain any TMAX features. It is a simple TEdit text file.

### 9.2 View Index

Buttoning this item first creates the index file (via **Create Index**) and then prompts you for a TEdit window where TMAX displays the index file it just created.

### 9.3 Index Filename:

This item allows you to specify the name of the index file. When this item is buttoned, the cursor will appear just to the right of this item. You then type the name of the index file and terminate it with a carriage return. You can edit this filename string anytime. If you have already terminated the string, just button the item again. The editing features available here are the same as those available in the EXEC window. If you **Get** a TEdit/TMAX file, the **Index Filename:** defaults to that file with a **.INDEX** extension.

NOTE... You must hardcopy your document before creating the index file. The reason is TMAX needs the page numbers for the index file but TEdit only knows the page numbers while the document is being hardcopied. During the hardcopy process, TMAX saves the page number(s) on which each index appears. If you create an index file without first hardcopying your TEdit document, the page numbers in this index file will be NIL.

## Chapter 10. Specifying a Font

TMAX uses the same mechanism whenever it prompts you for a font. To specify a font you must select a value from each of three different menus. The first pop-up menu is used to specify the font family. The font families are Classic, Gacha, Helvetica, Modern, and Times Roman. After you select the family, the font size menu pops up. This menu contains the sizes 6, 8, 10, 12, 14, 18, 24, and 36. After you select the size, the font face menu pops up. The font faces are Standard, Italic, Bold, and Bold Italic.

The default for any value not specified (i.e. the mouse is buttoned outside the pop-up menu) is whatever it was before. The default font always starts out as Gacha 10 Standard. Therefore, if you select a new family (say Helvetica) and button the mouse outside the font size and face menus, the new font would be Helvetica 10 Standard. Now if you were to button the mouse outside the font family and size menus and select a new face (say Bold), then the new font would be Helvetica 10 Bold.

Some TMAX functions (like **Set Style**) require you to select values from three different menus. Other functions (like changing a NGroup font) allow you to change one of the three fields directly without changing the other two. This is exactly equivalent to selecting a value from one menu and buttoning the mouse outside the other two menus.

## Chapter 11. Random thoughts and hints

- o You can Copy any of the TMAX features. from one TEdit window to another and TMAX will automatically set up the internal data structures necessary to support that feature in the destination window. For example, if you Copy a NGroup marker, TMAX will automatically set up the NGroup graph. Note that it only sets up enough to support what was copied; TMAX does not set up the entire NGroup graph.
- o TMAX does not support Move. If you want to move any TMAX features, you will have to Copy and then delete the features.
- o If you just want to select a TMAX feature, you should do so with the left mouse button. The right button will select the feature as well but it also causes TMAX to pop up a menu. Every TMAX feature pops up a menu if it's middle buttoned.
- o Regarding forward page References and doing a "dummy" hardcopy, we do the dummy hardcopy to a file whose device is {NODIRCORE} and whose extension corresponds to the eventual hardcopy device (e.g. {NODIRCORE}X.INTERPRESS). NODIRCORE is the bit bucket and the extension causes TEdit to load the appropriate fonts so the real hardcopy will go a little faster.
- o Remember that changing the font/format of a prototype NGroup will also change every occurrence of that NGroup in the document. Therefore you should settle on the font/format of the prototype NGroups before you start customizing any inserted NGroups (see section 6.8).
- o Occasionally TEdit (not TMAX) has problems displaying some TMAX features. For example, if you change the prototype NGroup font then every occurrence of that NGroup also changes. Sometimes it looks like only the first occurrence has changed when in fact they all have. If you

Redisplay the window, it will look like it should. It is a good idea to Redisplay the TEdit window whenever you think it doesn't look right.

---

---

**TMENU**

---

---

By: Mark Stefik, Daniel G. Bobrow, and Chris Tong (Stefik.pa@Xerox.com)

The TMENU package provides the following features:

**TMenus.**

These are interactive menus intended for reducing the amount of typing to an Interlisp-D program. When an item is selected in a menu, an expression is inserted into the TTY input buffer. It can be used to simplify the entry of common LISP commands or long names.

**Windowshades.**

Windowshades are a modification that can be made to any window in order to conserve screen space. They make a window "roll up" when not in use, leaving behind only the title. When the mouse is clicked inside the remaining bar, the window unrolls for interaction, and rolls up again at completion. Windowshades can be used with TMenus.

**Msgs to the Prompt window.**

Two functions are provided for clearing and printing to the Interlisp-D prompt window. These functions take an arbitrary number of arguments and preserve the black background shade of the window.

**INTERACTION WITH TMENUS**

TMenus are placed on the display under program control using the functions described below. Once a menu is on the screen, items may be selected using the LEFT mouse button. This causes some text to be inserted into the teletype buffer. In the default case, this text is just the item in the window, but it can alternatively be the result of evaluating an expression. In the default case, the text is followed by a blank space in the buffer, but it can be followed alternatively by an arbitrary string (such as the empty string or a carriage return). Non-default cases are controlled by arguments to the TMenu function. The MIDDLE mouse button is used for user-interactions that change the menu. When the middle button is depressed inside a menu, another pop-up menu (a meta-menu!) appears that provides several options for changing the menu, such as adding or deleting items. One of the options is to recompute the set of items by evaluating an expression associated with the menu. The RIGHT mouse button is used for the usual window commands. These commands work in the standard way except that when a TMenu is reshaped, internal functions are invoked to adjust the configuration of rows and columns in order to create a menu that is visually appealing.

**MAIN FUNCTIONS**

TMenu (*itemExpr title displaySpec windowShadeFlg buttonFn defaultTrailerString*)

[Function]

TMenu is the function for creating a menu in a window on the display. Its arguments are as follows:

*itemExpr* an expression for computing the list of items that is saved with the menu. In the usual case, *itemExpr* is a list, whose items are either atoms or lists of the form: (displayThis evalThis comment trailerString) where displayThis is the form displayed in the menu, evalThis is the form evaluated to create the text for the input buffer, comment is displayed in the prompt window if the LEFT button is held over an item for an extended period, and trailerString is the string inserted after an expression in the input buffer. Most of these fields are optional. The default value for evalThis is the entry in displayThis. The default trailerString can be specified for a TMenu in the defaultTrailerString argument above. Otherwise it is a space if the expression is an atom, and the empty string otherwise. This definition of fields for menu items is compatible with the usual set for the Interlisp-D menu package, with the addition of the trailerString field.

**Special cases:**

If *itemExpr* is an atom, it must be the name of a global variable to be evaluated to yield the list of the form described above (e.g., MYFNS). If *itemExpr* is a list and the first element of the list has a functional definition, then that function is evaluated to yield the list. This is intended for cases where the list is to be computed.

*title* The title of the menu. This title is used as the title of the window containing the menu.

*displaySpec* This argument has several possible interpretations that control the display of the menu. If *displaySpec* is a region, then the window for the menu is placed in that region. If *displaySpec* is a number, that number is used as the number of columns in the menu display and a minimum size window is allocated for displaying the entire menu. The user is prompted with a ghost box to place the menu on the display. If *displaySpec* is T, then the number of columns is computed by TMenu assuming a maximum of 15 rows per column and the user is prompted for placement as before. If *displaySpec* is NIL, then the user is prompted to place a bounding box for the menu and TMenu tries to compute an arrangement of rows and columns that is visually pleasing.

*windowShadeFlg* If T, the window containing the menu is augmented with a window shade so that the menu "rolls up" if not in use. If *windowShadeFlg* is NIL, the menu is placed in a window on the screen.

*buttonFn* Optional argument that allows the caller to specify his own function for handling the LEFT and MIDDLE buttons.



*defaultTrailerString* Optional argument that allows the caller to specify the default string to follow each item printed. Can be overridden for specific items by the *trailerString* argument in the *itemExpr*.

### EXAMPLE

The following expression: (TMenu 'MYFNS "Common Fns" T) would create a window titled "CommonFns" and display the list of functions in the window. The window would contain columns of up to 15 functions each, and would be placed on the screen under user control. If the user later used the MIDDLE button to add or delete items from the menu, then the list in the variable MYFNS would be updated as the menu is updated.

MakeFileMenus (*fileName*) [Function]

MakeFileMenus is the function for creating a set of menus for the functions and variables in a file. A window is created for each menu. The menus are placed under user control and are all given window shades. The argument *fileName* is the name of a file.

### Example

If the file is MYFILE, then the fns on MYFILEFNS and the vars on MYFILEVARS would be displayed in menus. MakeFileMenus would look for file commands of the form (FNS \* FnsLst) and (VARS \* VarsLst) on the command.

CloseFileMenus (*fileName*) [Function]

Closes all of the TMenu windows associated with the given file.

Window Shades MakeWindowShade (*window*) [Function]

Modifies the given window to provide a window shade. If the window argument is NIL, then the window is selected which is under the cursor. If the window argument is T, it waits for the CTRL key to be depressed, and then selects the window under the cursor. Prompt Window Functions PROMPT (arg1 arg2 arg3 ...) Prints an arbitrary number of arguments to the prompt window, after first clearing the window. A call with no arguments simply clears the window.

CPROMPT (*arg1 arg2 arg3 ...*) [Function]

Same as prompt except the arguments are printed centered in the window.

---

---

## Trajectory-Follower

---

---

By: D. Austin Henderson, Jr. (AHenderson.pa@Xerox.com)

### INTRODUCTION

Trajectory-Follower provides a function which causes a "snake" to crawl along a trajectory. Comments on both interface and functionality are welcomed.

### FUNCTIONS

(TRAJECTORY.FOLLOW *KNOTS CLOSED N DELAY BITMAP WINDOW*) [Function]

The trajectory is specified by *KNOTS* (a set of knots) and *CLOSED* (a flag indicating whether it is an open or closed curve). *N* is the length of the snake in points along the curve. *DELAY* is the time (in milliseconds) between each move along the curve; *DELAY* = 0 or NIL means go as fast as you can. *BITMAP* is the brush to be used at each point in creating the snake. *WINDOW* is the window in whose coordinate system the knots are given and in which the snake is to be drawn; if NIL, then the SCREEN bitmap is used. The snake is moved by INVERTing the bitmap at the points along the curve, and then INVERTing the bitmap back out again.

### Examples

A demonstration function is also provided with the module:

(TRAJECTORY.FOLLOWER.TEST) [Function]

Interacts with the user through prompting in the promptwindow to gather up arguments for TRAJECTORY.FOLLOW and then carries it out. Closed curves are snaked around repeatedly until the left shift key is found depressed when it reaches the curve's starting point.

### Internal Functions

The internal functions used by this module are also available for use. They are:

(TRAJECTORY.FOLLOWER.SETUP *WINDOW N DELAY BITMAP*) [Function]

Initializes drawing variables.

(TRAJECTORY.FOLLOWER.POINT *X Y WINDOW*) [Function]

Defines the next point on the curve. Note that the argument structure of this function is appropriate for use as a BRUSH with the curve drawing functions DRAWCURVE, DRAWCIRCLE, and DRAWELLIPSE. (For an example, see the demonstration function TRAJECTORY.FOLLOWER.TEST)

(TRAJECTORY.FOLLOWER.WRAPUP) [Function]

Finishes the job after all the points have been defined.

Sender: SenderNameTooLong%SUMEX-AIM.STANFORD:EDU:Xerox  
Date: 5 Jan 88 18:39:54 PST (Tuesday)  
Subject: Interlisp to Commonlisp conversion package  
From: darrelj%sm.unisys:COM  
To: info-1100%sumex-aim.stanford  
cc: darrelj%sm.unisys:COM

Return-Path: <@SUMEX-AIM.Stanford.EDU:sdcrdcf!sm.unisys.com!darrelj@PRC.Unisys.COM>  
Redistributed: Info-1100-Redistribute^.X  
Received: from SUMEX-AIM.Stanford.EDU by Xerox.COM ; 05 JAN 88 18:38:18 PST  
Received: from burdvax.PRC.Unisys.COM (PRC-GW.PRC.UNISYS.COM) by SUMEX-AIM.Stanford.EDU with TCP;  
Tue, 5 Jan 88 17:44:10 PST  
Received: by burdvax.PRC.Unisys.COM (burdvax) [5.54/1.0] id AA15346; Tue, 5 Jan 88 20:44:28  
EST  
Received: by rdcf.sm.unisys.com (sdcrdcf) (5.51/Domain/jpb/2.7) id AA28679; Tue, 5 Jan 88  
16:35:37 PST  
Message-Id: <8801060035.AA28679@rdcf.sm.unisys.com>  
Received: from XAVIER by sdcrdcf with PUP; Tue, 5 Jan 88 16:35 PST  
Original-Date: 5 Jan 88 16:34 PST (Tuesday)

Interlisp-D into Xerox Commonlisp, we developed a collection of tools  
to automate the conversion as much as possible.  
These have been placed in {parcvax.xerox.com}/lisp/exchange.  
While we at Unisys have reasonable confidence in these tools, they are  
being made available with no promises of accuracy, completeness or  
support (though we would appreciate feedback).

The tools run in Xerox Lyric Common Lisp. The following files are parts of it:

TRANSOR -- A slightly modified version of Transor, to fix a few  
Lyric-related problems and provide the ability to emit a  
DEFINE-FILE-INFO expression and to use the value of TRANSOUTREADTABLE  
as the output readtable. We used TRANSOR because we were familiar  
with it, and it handles a lot of details needed to safely and surely  
traverse the code to be translated. The biggest impediment to adding  
to the transforms is that they are specified as teletype editor  
commands, and only old-time Interlisps have much experience with those.  
TSET -- The same version dating back to 1979. This is the part of  
transor used for interactively developing and testing translation rules.  
TRANSOR.LCOM -- contains the compilation of BOTH the above files.  
TO-COMMONLISP.XFORMS -- translation rules for 428 functions, 98  
remarks and 4 auxiliary functions. It covers a large portion of  
Interlisp, including most Clisp constructs, and specifically handles  
any function with the same name in both Interlisp and Commonlisp, so  
that holes in a translation should result in calls to undefined  
functions. In many cases, nice transformations are used for easy  
cases, and ugly ones only for hard cases. This file sets  
TRANSOUTREADTABLE to be a copy of the XCL readtable which is case  
SENSITIVE, MYLOAD below reads it case INsensitive, so the resulting  
file will ultimately lose most case distinctions on reloading into  
Xerox Lisp (or other common lisps). This was a much debated point  
internally, but this seemed the best of three bad possibilities (e.g.  
print one of  
Cased AS ORIGINAL which becomes CASED AS ORIGINAL on load,  
|Cased| AS |ORIGINAL|, or  
CASED AS ORIGINAL

INTERLISP-COMMONLISP.TEDIT -- A document describing the  
transformations and formacro.

LOADTRAN -- contains a few functions which prevent many breaks on  
loading the translated file. The function MYLOAD is intended to load  
a translated file.  
LOADTRAN.DFASL -- compiled version

FORMACRO and FORMACRO.DFASL -- Still another portable iteration macro  
for commonlisp. Its main claims are almost 100% compatibility with  
the semantics of the Interlisp-Clisp FOR (especially when used the the  
XFORMS which fix a few incompatibilities); and user extensibility  
(unfortunately not compatible with IL:I.S.OPR). Embedded keywords  
(e.g. IN, COLLECT) may be in any package.

COMMON-MAKE and COMMON-MAKE.LCOM -- still another version of code to  
generate a more "common" source file. It handles more filepkg command  
types than most. Also, when used with COMMENTHACKS will successfully  
print ALL comments in semicolon format. Call  
IL:COMMON-MAKEFILE(file). It checks the MAKEFILE-ENVIRONMENT property  
to select a package and base.

COMMENTHACKS and COMMENTHACKS.LCOM -- patches to the prettyprinter and

to the DEFUN editor. The prettyprinter patches will print Interlisp  
 (\* --) comments as semicolon comments when \*PRINT-SEMICOLON-COMMENTS\*  
 is 'IL:ALL.  
 This file also redefines the ED method for DEFUNs so that the initials  
 and date of editing get updated for DEFUNs just as Interlisp has  
 always done for FNS.

Because of the way things developed, these tools are not as fully  
 integrated as they could have been. If we were doing it over, the  
 TRANSOR step could have more carefully coordinated the new COMS so  
 that COMMON-MAKE would be able to do the right thing. As it stands,  
 the COMS generally have to be edited to change FNS to FUNCTIONS, etc,  
 but you tend to need a few iterations of editing things before the  
 compiler is completely happy anyway.

The steps needed to do translations are roughly as follows:

```
(LOAD 'TRANSOR.LCOM)
(LOAD 'TO-COMMONLISP.XFORMS)
(SETQ FIXSPELLDEFAULT 'N) ;; Otherwise DWIM gets too clever
(SETQ XlatedRecords NIL) ;; This is currently set to records
specific to the system we translated.
TRANSOR files containing record declarations. The records MUST be
translated before any code containing create/fetch/replace since the
translation depends on the type of records. Also, the record
declarations should be LOADED. In a large translation effort, save a
file containing all needed declarations and the value of XlatedRecords
computed by translating them.
(TRANSOR 'file1) ... ;; results in file1.TRAN and file1.LSTRAN, see
TRANSOR documentation.
```

To load translated files into a fresh xerox lisp system:

```
>From an XCL exec:
(IL:SETPROPLIST '*COMMENT* (IL:GETPROPLIST 'IL:*))
(IL:PUTASSOC '*COMMENT* 'IL:* IL:PRETTYEQUIVLST)
(LOAD 'LOADTRAN)
(SETQ IL:*DEFAULT-MAKEFILE-ENVIRONMENT* '(:READTABLE "XCL" :PACKAGE ???))
(SETQ IL:CMLRDTBL (IL:FIND-READTABLE "XCL"))
(LOAD 'FORMACRO.DFASL) if used interlisp for's
;; may need to import USER:FOR depending on packages you've set up.
(MYLOAD 'translated-records)
(MYLOAD 'file1.tran) ...
```

A little work with ED and FILES? and you should be able to save a  
 commonlisp version of your files (well, OK, a lot of work).

Suggestions and questions to one of:  
 darrelj@RDCF.SM.UNISYS.COM or darrel@CAM.UNISYS.COM,  
 or fritzson@bigburd.prc.UNISYS.COM

----- End Forwarded Messages -----

---



---

## TRICKLE

---



---

By: Nick Briggs (Briggs.pa@Xerox.com)

Uses: PROMPTREMINDERS

This document last edited on October 12, 1987

### INTRODUCTION

Trickle provides a very simple cover for COPYFILES to do periodic (every 24 hours) updating of one directory from another, with processing of the log files generated by COPYFILES to mail a note to some designated person indicating what COPYFILES did.

### USE

There is only one function of interest to the user:

(Trickle *Source Destination RootLogfileName MailAddress ScheduleAnotherOne*  
*DontReplaceOldVersions*) [Function]

*Source* and *Destination* should be patterns acceptable to COPYFILES. *RootLogfileName* should be a host, directory, and partial file name to which Trickle will append the date in the form yymmdd, and the extension .CopyLog. On completion of the copy Trickle will mail a message to *MailAddress* if it is non-NIL. If *ScheduleAnotherOne* is T then another Trickle will be scheduled (randomly) between 1 am and 5:59 am of the next day, alternatively, if *ScheduleAnotherOne* is a time that would be acceptable to IDATE (Trickle will prepend the actual date, just give the time) then another Trickle will be scheduled at exactly that time. *DontReplaceOldVersions* signals that Trickle should not use the COPYFILES option REPLACE, use of which causes problems with NS file servers (at least in Koto).

### Example

To update the directory {cf}<lispusers>koto>\* from {eris}<lispusers>koto>\* storing the log files starting with {core}eluk-870512.copylog, mailing notification to Briggs.pa, scheduling this to run every night, and using COPYFILES' REPLACE option one would execute:

```
(SETREMINDER NIL NIL
 '(Trickle '{eris}<lispusers>koto>* '{cf}<lispusers>koto>*
 "{core}eluk-" "Briggs.pa" T)
 "12-May-87 03:00")
```

Two versions of the log file will be created; version 1 with the complete log output of COPYFILES, and version 2, with all the "skipped" files removed. It is this version that is mailed to the designated recipient.

The mail messages that are sent out indicate whether there were any files processed: the subject line will include the string "(Empty)" if no files were Trickled, and the string "(Error?)" if there were no files in the source directory (may not be an error, but may be worth investigating)

---

---

## TURBO-WINDOWS

---

---

By: Andrew J. Cameron, III (Cameron.pa@Xerox.com or cameron@cs.wisc.edu)  
New Owner: Atty Mullins (Mullins.pa@Xerox.com)

Uses: WDHACKS (LispUsers) [optional]

This document last edited on Sept. 8, 1988.

### INTRODUCTION

Turbo-Windows does not have anything to do with speeding up primitive window operations, but rather it helps speed up your use and manipulation of windows by providing most of the right button menu functions via shift keychords. In this way one can Move, Shape, Copy, Shrink, Close, etc., a window without having to wait for the right button menu to appear and then select from it.

Also, when providing the INITIAL shape of a window, pressing the middle button yields a large default size suitable for TEdit, etc. (Recall that using the middle button during a RESIZING operation allows you to keep roughly the original window shape and then move the corner nearest the cursor when the middle button was pressed.)

One can bring up a brief cribsheet for all the TurboWindow keychord commands by holding down the HELP key and RIGHT buttoning on the background (not in any window). This can also be produce by typing (TW.HELP) to an InterLisp EXEC.

### OPERATION

Before discussing how to use this utility, a description of how the key-chords were chosen is in order. They are based loosely on the effect of the shift keys in TEdit. Recall that in TEdit, pressing and holding the Shift key Copies whatever is selected. Also, pressing and holding the Control key (sometimes labeled PROPS or EDIT) Deletes whatever is selected. Pressing both Shift and Control performs both a copy and a delete, which ends up Moving the selected item. The only additional piece of information that you need to know is that the Meta key (sometimes labeled KEYBOARD or ALT) modifies an operation or in some way makes it different. With this general interpretation, most of the key-chords are rather easy to remember.

If the following keys are chorded (held down together) while the right mouse button is pressed in the region of a window which would normally bring up the right-button menu (by convention, at least the title bar should provide the right button menu), the listed operation will be invoked without actually bringing up the right button menu.

SHIFT (using the LEFT SHIFT key or CAPS LOCK key)

Makes a copy of a window by snapping it.

### CONTROL

Closes (deletes) a window. (Since this is a destructive operation, a small safeguard is built into this operation. If one holds the CONTROL key and depresses the right mouse button and continues to hold them, the window to be operated on (closed) will blink. If this is not the window you want to close

you can cancel the Turbo-Close by either moving outside the window (or by releasing the CONTROL key before releasing the right mouse button). If you abort the Turbo-Close in this manner, the normal right button menu will appear. Clicking outside of the menu will make it go away. Sometimes unexpected things occur when trying to Turbo-Close windows with attached windows, e.g. FileBrowsers, but hopefully this safeguard is conservative enough to avoid inadvertent closing of the wrong window.) [Holding down CONTROL while Right Buttoning on the background activates Window Slamming, if the LispUsers utility WDHACKS is loaded.]

#### META

Shape (makes different) a window.

#### SHIFT-CONTROL

Moves a window. (Due to the design of the InterLisp window system, this operation works in a rather strange way. You press and hold both CONTROL and SHIFT and then press the right mouse button while in the appropriate part (title bar) of the window you want to move. You then need to release the right button to be able to actually move the window. In order to "drop" the window (here is the strange part) you need to press the LEFT (or middle) button. Pressing the right button merely allows you to move to a different corner of the shadow box.)

#### META-CONTROL

Shrinks ("deletes" in a different way) a window.

#### META-SHIFT

Redisplays (copies in a different way) a window.

#### META-SHIFT-CONTROL

Buries (moves in a different way) a window. [You might also think of this as pushing the window down to the bottom, as you are pressing down all three shift keys.]

#### RIGHT-SHIFT

Clears a window.

#### HELP

Pressing the HELP key while the cursor is in the background (or typing (TW.HELP) to an InterLisp EXEC) displays a cribsheet for the Turbo-Window KeyChords. Some additional capabilities not listed here are given on that cribsheet. The "OTHER" keychords which are marked with an asterisk (\*) indicate that some side-effect (potentially quite harmful) might occur depending on where the TTY is when those alternate access methods are used. You are warned!

### GETTING STARTED

[If any of the operations described below do not perform properly, it might be the case that your keys are not defined in the way that this utility expects. See INTERNALS below for more information.]

You might want to get familiar with Turbo-Windows by first bringing up the cribsheet by depressing the HELP key and right-buttoning on the background. Next, make a copy of the cribsheet by depressing SHIFT (the left shift key) and right-buttoning on the cribsheet. Drop the new copy of the cribsheet by releasing all keys and buttons and then pressing the left mouse button. [Note: The cribsheet is merely written to the TTY window, which happens to be sensitive to the right mouse button everywhere. Other windows may only be sensitive to the right button (for the purpose of bringing up the right button menu, in their title bar.) Now try moving the copied cribsheet by pressing both SHIFT and CONTROL

(PROPS or EDIT) and right-buttoning on the copy of the crib sheet. Again, release everything (well, just releasing the mouse button will do) and press the left mouse button to drop it. Press and hold both META (KEYBOARD) and CONTROL while right buttoning in the copy of the crib sheet to shrink it to an icon. Release and click the left mouse button to drop the icon. Reopen (expand) the icon by middle buttoning on it. Reshape the copy of the crib sheet by pressing META and right buttoning on the copy's window. Release and rubberband the new shape with the left mouse button. (Do you know what would happen if you used the middle button after releasing instead? Try it.) Assuming the copy of the crib sheet is overlapping another window and some part of the background (if not, Turbo-Move it so it is), press and hold all three (META, SHIFT, and CONTROL) and right button in the crib sheet copy's window to bury it. Right button (holding no other keys) in the partially exposed area of the now buried crib sheet copy to bring it back to the top. Finally, close the copied crib sheet window by pressing CONTROL while right buttoning in the copy's window. [O.K. which shift key combination hasn't been used yet? Consult the original crib sheet (or produce it again), if necessary. Give that combination a try in the original crib sheet's window. [Did you notice the message in the prompt window?] And don't forget to give the Right Shift key (Clears a window) a try as well. [Remember, the crib sheet can be brought back at any time using HELP-RightButton on the background.] ) To see how to cancel a Turbo-Close, depress the CONTROL key and press AND HOLD the right mouse button while in the original crib sheet window. Notice that the window blinks. Before you release the right mouse button move the cursor outside the crib sheet's window and then release the right mouse button. The crib sheet's window is not closed because releasing outside the window that flashed cancels the Turbo-Close. The normal right button menu appears instead. Click outside it to get rid of it. Now, actually close the original crib sheet window. And with that, may I welcome you to the fast paced world of Turbo-Windows.

## INTERNALS

The right button events are intercepted by a piece of advice placed on DOWINDOWCOM. The middle button sizing capability is provide by advice on \GETREGIONTRACKWITHBOX.

The window snapping Turbo-Window feature (LeftShift-RightMouseButton) is also added as a submenu to the normal right button menu provided by the window system.

A common problem is that the META key is not defined to be at the proper place (attached to the key named KEYBOARD). To remedy this, type:

```
(KEYACTION 'KEYBOARD ' (METADOWN . METAUP))
```

to an InterLisp EXEC. The following should also be the case:

```
(KEYACTION 'EDIT ' (CTRLDOWN . CTRLUP))
```

```
(KEYACTION 'LSHIFT ' (1SHIFTDOWN . 1SHIFTUP))
```

```
(KEYACTION 'RSHIFT ' (2SHIFTDOWN . 2SHIFTUP))
```

These can be verified by using, for example:

```
(KEYACTION 'EDIT)
```

TW.NO-FLASH-CLOSE

[Variable]

Initially NIL, if set to T, windows will not flash to indicate there impending closure.



TW.DONT-GROW-SNAP-BORDER [Variable]

Initially NIL, if set to T, windows will be copied without a small border. The small border is quite handy in telling the original window from its Turbo-Snapped copy.

TW.SNAP-HERE [Variable]

Initially NIL, if set to T, windows will be copied directly on top of the window they are duplicating. Normally (when NIL) the user must position the copy.

GETREGIONDEFAULT [Variable]

This variable can be bound dynamically by an application to provide the region afforded by middle buttoning when prompted for an initial region of a window. It is initially set to roughly 7x9 inches, and is useful for TEdit windows, FileBrowsers, etc. [See the LispUsers utility RESIZE-FILEBROWSER for an even better way of dealing with FileBrowsers.]

- In order to edit/compile the source of this utility, the InterLisp Source file WINDOW must be loaded in order to provide the SCREEN record definition used by the window system internals. The loading of this source file occurs automatically when this utility's source file is loaded.
- This utility interacts poorly with other utilities that redefine any of the shift keys. TEDITKEY and PC-Emulation (among others) are dubious in this regard.

---



---

## TWODGRAPHICS

---



---

By: Jan Pedersen (Pedersen.PA @ Xerox.com)

Uses: UNBOXEDOPS

TWODGRAPHICS implements viewports. A viewport is a subregion of a window (or image stream) within which graphics is clipped and a linear transformation from a world coordinate system to the window (or image stream) coordinates.

A given window (or image stream) may have any number of viewports defined and the viewports may be arbitrarily nested or overlapping. If a window is reshaped the subregions of all currently defined viewports are proportionately reshaped.

Viewports will operate in the context of any image stream, (Interpress printers, etc.), although not all DIG (Device independent graphics) primitives are supported.

(CREATEVIEWPORT *stream streamsubregion source*) [Function]

Creates a viewport on stream. Stream is the target stream. Streamsubregion is a region in stream coordinates that defines the extent of the viewport.

Source may be a REGION in world coordinates, in which case the world to stream linear transformation is set up to map left to left and bottom to bottom, etc., or a VIEWPORT, in which case the new viewport inherits its world to stream transformation.

Returns a VIEWPORT

(SETWORLDREGION *region viewport*) [Function]

(Re)sets the worldregion of viewport and recomputes the transformation.

(SETSTREAMSUBREGION *region viewport*) [Function]

(Re)sets the streamsubregion of viewport and recomputes the transformation.

Modified versions of selected DIG primitives are supplied to take advantage of the world to stream transformation.

(TWODGRAPHICS.BITBLT *source sourceleft sourcebottom destinationviewport  
destinationleft destinationbottom width height  
sourcetype operation texture clippingregion*) [Function]

World coordinates may be used where it makes sense. The destination must be a VIEWPORT. Destination left and bottom default to the viewport's stream subregion left and bottom. The clippingregion argument is always in destinationviewport world coordinates. The source may be a VIEWPORT, a BITMAP, or NIL in the case of texture patterns.

In the following, all coordinates must be world coordinates.

|                                                                                                         |            |
|---------------------------------------------------------------------------------------------------------|------------|
| (TWODGRAPHICS.MOVETO <i>x y viewport</i> )                                                              | [Function] |
| (TWODGRAPHICS.MOVETOPT <i>position viewport</i> )                                                       | [Function] |
| Here position is a POSITION in world coordinates                                                        |            |
| (TWODGRAPHICS.RELMOVETO <i>dx dy viewport</i> )                                                         | [Function] |
|                                                                                                         |            |
| (TWODGRAPHICS.RELMOVETOPT <i>dposition viewport</i> )                                                   | [Function] |
| (TWODGRAPHICS.DRAWTO <i>x y width operation viewport color dashing</i> )                                | [Function] |
| (TWODGRAPHICS.DRAWTOPT <i>position width operation viewport color dashing</i> )                         | [Function] |
| (TWODGRAPHICS.RELDRAWTO <i>dx dy width operation viewport color dashing</i> )                           | [Function] |
| (TWODGRAPHICS.RELDRAWTOPT <i>dposition width operation viewport color dashing</i> )                     | [Function] |
| (TWODGRAPHICS.DRAWLINE <i>x1 y1 x2 y2 width operation viewport color dashing</i> )                      | [Function] |
| (TWODGRAPHICS.DRAWBETWEEN <i>position1 position2 width operation viewport color dashing</i> )           | [Function] |
| (TWODGRAPHICS.DSPRESET <i>viewport</i> )                                                                | [Function] |
| Does a “DSPRESET“ on the VIEWPORT                                                                       |            |
| (TWODGRAPHICS.DSPFILL <i>region texture operation viewport</i> )                                        | [Function] |
| region must be in world coordinates                                                                     |            |
| The following function is an extension which may be of use to those who wish to produce analytic plots. |            |
| (TWODGRAPHICS.PLOTAT <i>position glyph viewport operation</i> )                                         | [Function] |
| Bitblts glyph to position with operation, with glyph centered at position.                              |            |
| Several functions provide access to the world to stream transformations.                                |            |
| (WORLDTOSTREAM <i>position viewport oldposition</i> )                                                   | [Function] |
| Position is in world coordinates. Oldposition is smashed if provided.                                   |            |
| Returns the corresponding position in stream coordinates.                                               |            |
| (WORLDREGIONTOSTREAMREGION <i>region viewport</i> )                                                     | [Function] |
| Region is in world coordinates                                                                          |            |
| Returns the corresponding region in stream coordinates                                                  |            |
| (WORLDTOSTREAMX <i>x viewport</i> )                                                                     | [Macro]    |
| Returns x in stream coordinates.                                                                        |            |
| Uses unboxed floating point arithmetic                                                                  |            |
| (WORLDTOSTREAMY <i>y viewport</i> )                                                                     | [Macro]    |
| Returns y in stream coordinates                                                                         |            |
| Uses unboxed floating point arithmetic.                                                                 |            |

|                                                                                                                                                                                                                                                  |            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| (WORLDXLENGTH <i>dx viewport</i> )                                                                                                                                                                                                               | [Macro]    |
| Returns the length dx in stream coordinates                                                                                                                                                                                                      |            |
| Uses unboxed floating point arithmetic.                                                                                                                                                                                                          |            |
| (WORLDYLENGTH <i>dy viewport</i> )                                                                                                                                                                                                               | [Macro]    |
| Returns the length dy in stream coordinates.                                                                                                                                                                                                     |            |
| Uses unboxed floating point arithmetic.                                                                                                                                                                                                          |            |
| (STREAMTOWORLD <i>position viewport oldposition</i> )                                                                                                                                                                                            | [Function] |
| Returns position in world coordinates.                                                                                                                                                                                                           |            |
| (STREAMTOWORLDDX <i>x viewport</i> )                                                                                                                                                                                                             | [Macro]    |
| Returns x in world coordinates.                                                                                                                                                                                                                  |            |
| Uses unboxed floating point arithmetic.                                                                                                                                                                                                          |            |
| (STREAMTOWORLDDY <i>y viewport</i> )                                                                                                                                                                                                             | [Macro]    |
| Returns y in world coordinates.                                                                                                                                                                                                                  |            |
| Uses unboxed floating point arithmetic.                                                                                                                                                                                                          |            |
| (STREAMXLENGTH <i>dx viewport</i> )                                                                                                                                                                                                              | [Macro]    |
| Returns dx in world coordinates.                                                                                                                                                                                                                 |            |
| Uses unboxed floating point arithmetic.                                                                                                                                                                                                          |            |
| (STREAMYLENGTH <i>dy viewport</i> )                                                                                                                                                                                                              | [Macro]    |
| Returns dy in world coordinates.                                                                                                                                                                                                                 |            |
| Uses unboxed floating point arithmetic.                                                                                                                                                                                                          |            |
| For those who desire tighter control over the two-stage process, transform into stream coordinates, and then clip against the viewport, the following functions provide primitive clipping for line drawing and text output in any image stream. |            |
| (CLIPPED.BITBLT <i>clippingregion source sourceleft sourcebottom<br/>destination destinationleft destinationbottom<br/>width height sourcetype operation texture</i> )                                                                           | [Function] |
| As in BITBLT, although the operation is clipped against clippingregion in destination stream coordinates.                                                                                                                                        |            |
| (CLIPPED.DRAWLINE <i>clippingregion x1 y1 x2 y2 width operation stream<br/>color dashing</i> )                                                                                                                                                   | [Function] |
| As in DRAWLINE, although the operation is clipped against clippingregion in stream coordinates.                                                                                                                                                  |            |
| (CLIPPED.DRAWTO <i>clippingregion x y width operation stream color dashing</i> )                                                                                                                                                                 | [Function] |
| As in DRAWTO, although the operation is clipped against clippingregion in stream coordinates.                                                                                                                                                    |            |
| (CLIPPED.DRAWBETWEEN <i>clippingregion pt1 pt2 width operation stream color dashing</i> )                                                                                                                                                        | [Function] |
| As in DRAWBETWEEN, although the operation is clipped against clippingregion in stream coordinates.                                                                                                                                               |            |

(CLIPPED.PLOTAT *clippingregion position glyph stream operation*)

[Function]

BITBLT glyph to stream centered at position and clipped against clippingregion.

(CLIPPED.PRIN1 *clippingregion expr stream*)

[Function]

PRIN1 expr on stream clipped against clippingregion.

---



---

## UNBOXEDOPS

---



---

By: Jan Pedersen(Pedersen.PA @ Xerox.com] and Larry Masinter (Masinter.PA @ Xerox.com]

The module UNBOXEDOPS is intended to assist those interested in high-performance, scalar, floating-point arithmetic. The basic trick is to perform floating point arithmetic on the stack, utilizing special, unboxed, floating-point opcodes, an ugly but usually effective solution. This method of eliminating floating-point number boxes is likely to change, but in the interim a combination of compiler declarations and explicit evocations of unboxed operations, as described below, will allow the interested user to eliminate a high percentage of floating-point number boxes. This module and the methods described are "safe", i.e., the declarations won't cause your programs to crash, and if it works with the declarations it will also work without them.

Unboxed floating point tricks help out only 1108's with floating point hardware or 1186's with floating point microcode. Unfortunately, they may make performance even worse on 1108's without floating point hardware, although the performance degradation is probably not too severe.

There exist opcodes which perform floating point arithmetic on the stack (that is, on the bits of those numbers, rather than pointers to those bits). These opcodes are only emitted by the byte compiler if arithmetic occurs in an unboxed context. One example of an unboxed context is arithmetic on a record field defined to be of type FLOATP, another is arithmetic on a variable declared to be of TYPE FLOAT . However, the compiler will box across function boundaries and in a return context. Furthermore, there exist more unboxed opcodes than are used by the compiler (unboxed comparison springs to mind).

UNBOXEDOPS defines macros/functions so that these additional opcodes may be exploited in an unboxed context. These macros/functions include:

UFABS, UFEQP, UFGEQ, UFGREATERP, UFIX, UFLEQ, UFLESSP, UFMAX, UFMIN, UFMINUS, and UFREMAINDER,

which behave identically to there non-U namesakes, except that the operations are done on the stack without generating floating point boxes.

For those unfamiliar with unboxed compiler declarations a short description follows:

### Using (DECLARE (TYPE FLOATING x y z)) to reduce number boxes

Consider the silly function:

```
(DEFINEQ (FIE (N)
 (bind (SETQ X 0.0) (SETQ Y 2.0) for I from 1 to N
 do (SETQ X (FPLUS X (FTIMES Y Y)))
 finally (RETURN X)))
 (TIMEALL (FIE 100))
```

returns a CPU time of .025 and reports 200 FLOATP boxes produced. Now, consider

```
(DEFINEQ (FOO (N)
 (bind (SETQ X 0.0) (SETQ Y 2.0) for I from 1 to N
 declare (TYPE FLOAT X Y)
 do (SETQ X (FPLUS X (FTIMES Y Y)))
 finally (RETURN X)))
 (TIMEALL (FOO 100))
```

returns a CPU time of .003 seconds and reports just one floatp box produced.

Essentially the (TYPE FLOAT X Y) declaration is a promise to the compiler that X and Y will hold FLOATP's , so arithmetic may be done unboxed (that is on the value itself, instead of on a pointer to the value, which is the usual case) if possible. The key issue is what is meant by "if possible".

The compiler is conservative. It will perform unboxed arithmetic only on built-in arithmetic functions (PLUS , TIMES, DIFFERENCE, etc), which have unboxed counter parts, and will otherwise box across function boundaries regardless of compiler declarations.

For example:

```
(DEFINEQ (FOOBAR (N)
 (bind (SETQ X 0.0) (SETQ Y 2.0) for I from 1 to N
 declare (TYPE FLOAT X Y)
 do (SETQ X (FPLUS X (LOG Y)))
 finally (RETURN X)))
```

then

```
(TIMEALL (FOOBAR 100))
```

returns a CPU time of .049 with 601 FLOATP boxes produced (some of which come from the LOG (five per function call)).

Also, the compiler will box in a return context. For example

```
(DEFINEQ (BAR (N)
 (bind (SETQ X 0.0) for I from 1 to N
 declare (TYPE FLOAT X)
 do (SETQ X
 (PROG ((Y 2.0))
 (DECLARE (TYPE FLOAT Y))
 (RETURN (FTIMES Y Y))))
 finally (RETURN X)))
```

then

```
(TIMEALL (BAR 100))
```

returns a CPU time of .022 with 301 FLOATP boxes produced -- notice that BAR seems like it should behave like FOO.

Indeed that is the the greatest drawback of the unboxed arithmetic as it stands now -- it is not always easy to predict what is going to happen -- there are even traps where indiscriminate uses of TYPE FLOAT declarations will actually produce MORE boxes than without them. This is the case if, for

example, you use comparison operators (GREATERP, etc) since the compiler boxes each operand before invoking them.

The BAR example may be fixed up as follows:

```
(DEFINEQ (BAR (N)
 (bind (SETQ X 0.0) for I from 1 to N
 declare (TYPE FLOAT X)
 do (SETQ X
 (PROG ((Y 2.0) RESULT)
 (DECLARE (TYPE FLOAT Y RESULT))
 (RETURN (SETQ RESULT (FTIMES Y Y))))
 finally (RETURN X)))
```

then

```
(TIMEALL (BAR 100))
```

returns a CPU time of .008 with 101 FLOATP boxes produced. Note that the compiler still boxes the result returned by the PROG.

The best way to find out what is happening is to use a combination of TIMEALL and INSPECTCODE . Unanticipated boxing behavior will show up as BOX opcodes -- if you find a sequence of opcodes UNBOX , BOX , function call, UNBOX , then you know you are in trouble. TIMEALL will report the total number of boxes produced.

Basically TYPE FLOAT declarations are best used in tight inner loops of the sort illustrated in FOO.

With all these caveats, I think it is only fair to say that considerable performance improvements can be realized with judicious use of the TYPE FLOAT declarations; my measurements indicate a factor of ten.

Additional note: TYPE FLOAT vars are by necessity LOCALVARS.

Lyric compatibility note: All the entries described for this module are in the Interlisp package. Only the Byte compiler pays attention to TYPE FLOAT declarations -- i.g. use of TYPE FLOAT declarations will be ignored by the XCL compiler.



---

---

**UNDIGESTIFY**

---

---

By: Steven C. Bagley (Bagley.pa)

This document last edited on May 27, 1986

**INTRODUCTION**

This Lafite package allows you to unpack an Arpa Network digest, such as AList, into its constituent messages. A new item, "Undigest," is placed on the browser menu. When a single message (presumably a digest) is selected, clicking on this item will delete the selected message, and append the constituent messages to the end of the mail folder. If the selected message is not a digest, or is a digest in a format that cannot be parsed properly, then a message will be printed and nothing will happen to the mail folder.

**USER OPTIONS**

\*DELETE-DIGEST-FLAG\*, if T means that the digest message should be deleted if it is successfully parsed. The default is T.

\*MOVE-TO-FIRST-DIGEST-MESSAGE-FLAG\*, if T means to select the first constituent message, if NIL means to select the first undeleted message after the digest message. The default is NIL.

\*DONT-UPDATE-HEADERS-FLAG\*, if T means not to copy the To: field from the digest to each constituent message. The default is NIL.

**PLANNED ENHANCEMENTS**

Inserting the contained messages immediately after the digest, rather than appending.

Moving the contained messages to a different mail folder.

**NOTES**

Many digests are not in the correct format. The parser used in this program tries to be very forgiving, and hence, is relatively slow (about 10 seconds to parse a digest on a Dorado). If everyone adhered to RFC934, the parser could be optimized for speed, but, alas, this is not the case.

---

---

## UUENCODE

---

---

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on October 7, 1987.

UUENCODE provides facilities for encoding files into printing ASCII characters for transfer by electronic mail. It is compatible with the UNIX™ facility of the same name. For details of the file format see the UNIX™ manual page on 'uuencode'.

(UUENCODE *FILES INTO-FILE*) [Function]

Encodes the files named by FILES into INTO-FILE. FILES may be either a list of files or a single file name. Note that UNIX™ uuencode does not support multiple files encoded in one file. Thus one should only pass a single file name to UUENCODE if the file is to be decoded under UNIX™. Returns the name of the file written.

(UUDECODE *FILE-OR-STREAM ONLY-ONE-FILE?*) [Function]

Decode from FILE-OR-STREAM writing the decoded files in the connected directory. FILE-OR-STREAM may be either a file name or a stream. If ONLY-ONE-FILE? is non-NIL then only one file will be extracted from FILE-OR-STREAM, and an error will be reported if no encoded file is found. This can be thought of as UNIX™ compatibility mode. Returns the list of the names of the files extracted.

(UUENCODE-INTERNAL *INS OUTS DECODE-NAME FILE-MODE*) [Function]

Called by UUENCODE to encode one file. Encodes all bytes from the stream INS to the stream OUTS. DECODE-NAME is the name the file should be given when it is decoded. FILE-MODE is the UNIX™ file mode for the file. DECODE-NAME defaults to (FULLNAME INS) and FILE-MODE defaults to the value of the variable UU.MODE-DEFAULT. Returns OUTS.

UU.MODE-DEFAULT [Global Variable]

The default UNIX™ file mode to encode files under as an integer. UNIX™ uuencode will use this when creating the decoded file. The initial value is 644Q (read & write by owner, read by group and other).

(UUDECODE-INTERNAL *INS ONE-FILE-ONLY?*) [Function]

Called by UUDECODE to decode one file. INS should be a stream open for input. Returns the name of the file extracted or NIL if none is found and ONE-FILE-ONLY? is NIL.

UUENCODE was inspired by Christopher Lane's BMENCODE package.

---

---

## VSTATS

---

---

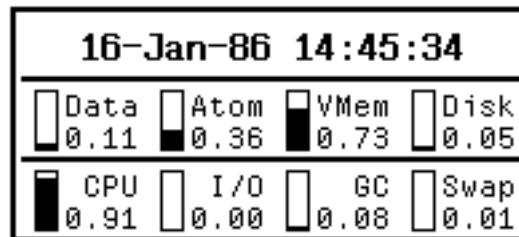
By: Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester)

Uses: SYSTATS, READNUMBER

This document last edited on November 20, 1987

### INTRODUCTION

Loading VSTATS will put a VStats entry on the background menu, and execute (VSTATS 'On), which will cause the following display to be created and continuously updated:



### DESCRIPTION

VSTATS is a facility for continuously displaying various interesting aspects of a running system. It can display the current time of day, with or without seconds, and/or display memory and disk space utilization, and/or display machine utilization in terms of CPU, I/O, GC and swap time. The display can be regular or inverse-video. The display is updated at user settable intervals, either always or only if the display window is completely visible.

Closing the VSTATS window will remove the background update function.

Left buttoning the VSTATS window causes it to be recomputed and redisplayed entirely. Otherwise display updates only affect those parts that have actually changed, making for a visually quiet and efficient facility.

Middle buttoning the VSTATS window will bring up an Inspector window onto the VSTATS list of options. Left buttoning an option name prints an explanation of the option to the Prompt window. Left buttoning an option value selects it, and middle buttoning an option value presents a menu from which a new value can be selected. The options window looks like this:

| VStats Options        |          |
|-----------------------|----------|
| Display.Color         | Normal   |
| Update.Always?        | No       |
| Show.Disk.Space?      | DSK1     |
| MUtil.Hysteresis      | 20       |
| Space.Panic.Level     | Disabled |
| Clock.Update.Interval | 1        |
| Space.Update.Interval | 300      |
| MUtil.Update.Interval | 1        |

VSTATS is highly optimized for speed and implemented as a BACKGROUNDFN rather than as a separate process so as to minimize overhead. As a result, VSTATS can easily be run with display update intervals equal to 1 second.

## DETAILS

(VSTATS *on/off*) [Function]

If *on/off* is either ON, On, on, or T, the VStats display is turned on; otherwise off.

VSTATS.CLOCK.INTERVAL [Variable]

If the global variable VSTATS.CLOCK.INTERVAL is a positive number, VSTATS displays an alphanumeric clock (e.g., "1-Aug-85 14:30"), which is updated every VSTATS.CLOCK.INTERVAL seconds. If this interval is less than 1 minute VSTATS displays seconds as well. For those of you who keep their machines running overnight (say, with IDLE or BOUNCE), if the clock display is enabled, VSTATS will resynchronize the local clock with the network daily at midnight. (My machine loses about 15 minutes a week, otherwise!)

VSTATS.SPACE.INTERVAL [Variable]

If the global variable VSTATS.SPACE.INTERVAL is a positive number, VSTATS displays, both graphically and alphanumerically, the utilization of Data, Atom, and VMem spaces and optionally Disk space, which is updated every VSTATS.SPACE.INTERVAL seconds.

VSTATS.SPACE.PANIC.LEVEL [Variable]

If VStats is displaying space utilization, and VSTATS.SPACE.PANIC.LEVEL is a percentage between 1 and 100 (or a fraction between 0 and 1), and any of the memory space utilizations (other than disk) exceed this percentage, VSTATS will flash its window in proportion to the excess, whether the window is occluded or not.

VSTATS.SPACE.SHOW.DISK? [Variable]

If VStats is displaying space utilization, then if VSTATS.SPACE.SHOW.DISK? is non-NIL, Disk space utilization is displayed as well, provided VStats can figure out the total disk size. If VSTATS.SPACE.SHOW.DISK? is T, the default DSK is used, for instance {DSK19} on a Dorado, or {DSK}<LispFiles> on a Dandelion. Alternate Dorado partitions or Dandelion volumes may be assigned to VSTATS.SPACE.SHOW.DISK? as well. If assigned through the options window, VStats will figure out which volumes or partitions are displayable.

VSTATS.MUTIL.INTERVAL [Variable]

If the global variable VSTATS.MUTIL.INTERVAL is a positive number, VSTATS displays, both graphically and alphanumerically, the machine utilization in terms of CPU time, time spent on disk and

Ethernet I/O, garbage collection time, and swapping time, which is updated every VSTATS.MUTIL.INTERVAL seconds.

VSTATS.MUTIL.HYSTERESIS [Variable]

If VStats is displaying machine utilization and VSTATS.MUTIL.HYSTERESIS is a positive number, the relative percentages are based on the average over VSTATS.MUTIL.HYSTERESIS intervals, otherwise they are based on the total time since VSTATS was invoked.

VSTATS.POSITION [Variable]

If the global variable VSTATS.POSITION is a POSITION, the VSTATS display will be put there, otherwise the user is prompted for a POSITION.

VSTATS.BLACK? [Variable]

If the global variable VSTATS.BLACK? is non-NIL, VSTATS displays with inverse video.

VSTATS.ALWAYS? [Variable]

If the global variable VSTATS.ALWAYS? is non-NIL, VSTATS will always update its display when its timers expire, causing its window to come to the top if it isn't already there; otherwise, VSTATS will only update the display if its window is neither partially nor wholly occluded. If it is occluded, VSTATS will, of course, continue to update its internal timers and the display will be updated the first time the timers expire after the display becomes wholly visible again.

#### Defaults

|                          |                             |
|--------------------------|-----------------------------|
| VSTATS.BLACK?            | NIL                         |
| VSTATS.ALWAYS?           | NIL                         |
| VSTATS.POSITION          | top right corner of display |
| VSTATS.CLOCK.INTERVAL    | 1 second                    |
| VSTATS.SPACE.INTERVAL    | 300 seconds (5 minutes)     |
| VSTATS.SPACE.PANIC.LEVEL | 95 %                        |
| VSTATS.SPACE.SHOW.DISK?  | T                           |
| VSTATS.MUTIL.INTERVAL    | 1 second                    |
| VSTATS.MUTIL.HYSTERESIS  | 20 intervals                |

If different values are preferred, these variables should be set by the user before loading VSTATS to affect initial display. They can of course be altered anytime using the options menu.

#### Extras:

A number of functions are required (and supplied) by VSTATS which the author believes might well be part of standard Interlisp-D, viz.,

(COVEREDWP *window*) [Function]

Returns T if *window* is partially or completely covered by some other window; NIL otherwise.

(CLOCKTICKS *interval timerunits*) [Function]

Returns the (machine dependent!) number of internal clock ticks over the interval. For instance, on the D'Lion

(CLOCKTICKS 2.5 'MINUTES) = 5211900

(ALTOPARTITIONS)

[Function]

On the Dorado, returns a list of partitions set up with an Alto exec, i.e., containing a system boot file. Especially useful with the recently added Extended VMem option, where not all partitions are bootable. Returns NIL on any other machine type. Note: this list of partitions takes between 15-20 seconds to compute.

(DISKUSEDPAGES *dsk recompute*)

[Function]

Returns the total number of disk pages in use (complementing DISKFREEPAGES). On Dorado, this is only an estimate, unless *recompute* is non-NIL in which case you wait ~ 8 seconds for the answer.

(DISKTOTALPAGES *dsk recompute*)

[Function]

Returns the total number of disk pages available (sum of DISKFREEPAGES and DISKUSEDPAGES).

---

---

WALKFILES

---

---

Johannes A. G. M. Koomen  
(Koomen.wbst@xerox or Koomen@cs.rochester.edu)

October 27, 1989

## SUMMARY

WALKFILES is a facility for searching loaded files for arbitrary objects. It complements MasterScope, in that it can do some things that MasterScope can't (like looking for arbitrary substrings in functions); other things Masterscope can do as well or better. WALKFILES does not require an analysis of the files prior to action.

## DESCRIPTION

(WALKFILES *pattern file{s} editcommands confirmflg quietflg filepackagetypes*) [Function]

Invokes WALKDEFS on all the objects on each *file{s}* of the types given in *filepackagetypes*. If *file{s}* is NIL, the value of FILELST is used. If *filepackagetypes* is NIL, all the filepackage types in PRETTYTYPEELST are used.

(WALKDEFS *pattern name{s} filepkgtype editcommands confirmflg quietflg*) [Function]

Walks over the *filepkgtype* definition of each name in *name{s}* looking for *pattern*. Pattern can be anything acceptable to EDITFINDP. For each occurrence, first prints the name (unless *quietflg* = T) and then prints the occurrence if *editcommands* is NIL, or invokes the editor in interactive mode if *editcommands* is T (asking for confirmation if *confirmflg* = T), or applies *editcommands* to the name and *filepkgtype* if *editcommands* is a function, or otherwise invokes the editor with *editcommands*.

## EXAMPLES

(WALKFILES 'elseif 'MYFILE T) [Example]

Brings up the editor on every definition in MYFILE which contains the symbol elseif.

(WALKFILES '\$FOO\$) [Example]

Prints every occurrence of any symbols with FOO as a substring in every definition in any file in FILELST.

(LET (THEFNS) [Example]  
 (WALKFILES '(\*ANY\* \$FOO\$ BAR) NIL '(LAMBDA (FN) (PUSH THEFNS FN)) NIL T 'FNS)  
 THEFNS)

Collects all functions containing either the symbol BAR or any symbols with FOO as a substring on every file in FILELST.



---

---

**WDWHACKS**

---

---

By: Atty Mullins (Mullins.pa@Xerox.com)

Revised by: Ron Kaplan (kaplan@parc.xerox.com)

Some short hacks that make window management slightly easier.

Loading this file forces the menu entries: **SLAMWS** (in the background menu) with the subitem **INSPECTORS**, **SEDIT**, and **FILEBROWSERS**, and **POPSHAPE** (under "Close" in the window menu), and replaces the action for **SHAPE** (window menu) with its own call.

When **SLAMWS** is selected, you are asked for a region of the screen, and all of the windows that intersect that region are closed (by call to **CLOSEW**). If one of the **INSPECTORS**, **SEDIT**, or **FILEBROWSERS** subitems is selected, all open (not shrunken) windows of the indicated type are closed.

When **SHAPE** is selected, the old shape of the window is stored, and then selecting **POPSHAPE** will reshape the window to the stored shape. When **POPSHAPE** is called, the current shape is stored, so that doing **POPSHAPE** multiple times will rotate between the two shapes.

The shape/popshape hacks are useful if you want to either get something out of the way temporarily (but not lose it entirely); or to enlarge something temporarily (e.g., for doing a SEE in the typescript window).

---

---

# WHEELSCROLL

---

---

By Ron Kaplan

This document was last edited in June 2023.

This small file adds the ability to scroll (scrollable) windows by rotating the wheel on a wheel mouse or by moving (2?) fingers on a track pad. The capability is enabled when `WHEELSCROLL.LCOM` is loaded. It is toggled on and off by

`(ENABLEWHEELSCROLL ON)` [Function]

Initially `(ENABLEWHEELSCROLL T)`.

The vertical scrolling speed is controlled by the variable

`WHEELSCROLLDELTA` [Variable]

The number of points to scroll for each click of the wheel. Higher values give faster scrolling. A negative value reverses the scrolling direction. Initially value: 20.

`HWHEELSCROLLDELTA` [Variable]

If non-NIL, this is the delta used for horizontal scrolling.

## Implementation:

Lisp receives a key transition on `PAD1` or `PAD2` for vertical scrolling when the wheel rotates and no other keys are down. `(ENABLEWHEELSCROLL T)` modifies the keyaction table so that it maps these transitions to characters 156 and 157. Those characters are defined as interrupts that invoke the vertical scrolling action. For horizontal scrolling sideways pushes of a wheel (if it has that) produce transitions on `PAD4` and `PAD5`, which map to interrupt-characters 158 and 159. (156-159 are the highest right-panel characters of character-set 0 that correspond to left-panel control characters, so typically have no other conflicting meaning.)

`(ENABLEWHEELSCROLL NIL)` causes `PAD1`, `PAD2`, `PAD4`, and `PAD5` to be ignored.

## Current negative features:

1. When the wheel is depressed for middle-button effect (and no other keys are down), an accidental rotation of the wheel during the transition (up and/or down) may cause unintended scrolling.

We need to develop a strategy, either in Lisp, Maiko, or X, to discriminate intended middle-button pushes from intended scrolling. This is not an issue for track-pad scrolling.

2. When the wheel is rotated over a window that partially occludes a Tedit window with a caret blinking in its unoccluded region, both the target window and the partially obscured Tedit window may scroll.

---

---

## WHOCALLS

---

---

By: Bill van Melle (vanMelle.pa@Xerox.com)

This file contains two useful functions for quick crossreference:

(whocalls *callee usage*) [Function]

maps over all symbols in the current environment, looking for any function that mentions callee according to usage:

values for usage:

USES VAR VARS BOUND USEDFREE GLOBALS

All mean: mention as a variable

NIL CALLS

means calls as a function

(distribute.callinfo) [Function]

inverts all of the call, use global, use free, bound relations for functions, variables from compiled code. Operates by mapping over all symbols in the sysout that are defined as compiled code, and analyzing their definitions. Anything that is called has a CALLED BY property of all of the things that call it; any variable bound has a BOUND BY with the list of functions that bind it, variables that are used globally have a USED GLOBAL BY and variables that are used freely have a USED FREE BY.

(References from interpreted code, etc are not detected, so it isn't 100% guaranteed that if something doesn't have a CALLED BY that it isn't called.....)

---

---

## WHO-LINE

---

---

By: SML (Lanning.pa@Xerox.com)

### INTRODUCTION

Need to know what package you're in? Don't know what your connected directory is? Fret not. The Who-Line is here.

The Who-Line is a window that displays this information on your screen. It is continually updated to reflect the current state of the world (thanks to an entry on BACKGROUNDFNFS). Additionally, items in the Who-Line can act as menu items, allowing you to change the state of the machine.

### Defining the information displayed in the Who-Line

The values displayed in the Who-Line are determined by the setting of the variable `*WHO-LINE-ENTRIES*`.

`*WHO-LINE-ENTRIES*` [Global Variable]

`*WHO-LINE-ENTRIES*` is a list that describes the items that will be displayed in the who-line. Each item in the list should be a list of up to five things: the name of the item; a form that, when evaluated, will produce the value to display; the maximum number of characters in the value; an optional function to call if the item is selected (with the mouse) in the Who-Line; an optional form that will reset any internal state of the entry when evaluated; and an optional string that describes the value displayed by the entry.

[[NOTE: Since the items on the Who-Line are evaluated rather often, it is best if they are fast and efficient (= don't CONS or allocate any space).]]

The following are standard members of `*WHO-LINE-ENTRIES*`.

`*WHO-LINE-USER-ENTRY*` [Variable]

Displays the current user in the Who-Line. Selecting this item in the Who-Line will let you change the logged in user.

`*WHO-LINE-HOST-NAME-ENTRY*` [Variable]

Displays the (ETHERHOSTNAME) of the machine you are running on.

`*WHO-LINE-PACKAGE-ENTRY*` [Variable]

Displays the package of the current TTY process in the Who-Line. Selecting this item in the Who-Line will let you switch the package of the current TTY process.

`*WHO-LINE-READTABLE-ENTRY*` [Variable]

Displays the (name of the) readtable of the current TTY process in the Who-Line. Selecting this item in the Who-Line will let you switch the readtable of the current TTY process.

**\*WHO-LINE-TTY-PROC-ENTRY\*** [Variable]

Displays the name of the current TTY process in the Who-Line. Selecting this item in the Who-Line will let you give the TTY to a different process.

**\*WHO-LINE-DIRECTORY-ENTRY\*** [Variable]

Displays the current connected directory in the Who-Line; the directory is shown in the format "Dir>Subdir>...>Subdir on {Host}". Selecting this item in the Who-Line will let you connect to another directory: the variable **\*WHO-LINE-DIRECTORIES\*** (see below) is used to produce a menu of interesting directories. If you are holding down a SHIFT key when you select an item from this menu, the directory name will be COPYINSERTed into the current tty input stream, otherwise you will be connected to that directory.

**\*WHO-LINE-VMEM-ENTRY\*** [Variable]

Displays the percentage of the VMem file that is currently being used in the Who-Line. If the VMem file is inconsistent, the number will be preceded by an asterik ("\*"). Selecting this item in the Who-Line will let you do a (SAVEVM).

**\*WHO-LINE-SYMBOL-SPACE-ENTRY\*** [Variable]

Displays the percentage of symbol space that is currently in use.

**\*WHO-LINE-TIME-ENTRY\*** [ Variable]

Displays the current time in the Who-Line. Selecting this item in the Who-Line will let you do a (SETTIME). If you hold down a shift key when you select this item, the current time will be COPYINSERTed into the current tty input stream instead.

The default value of **\*WHO-LINE-ENTRIES\*** contains all these items

### **Other ways to tailor the Who-Line**

**\*WHO-LINE-ANCHOR\*** [Variable]

**\*WHO-LINE-ANCHOR\*** describes where the who-line will be displayed. If **\*WHO-LINE-ANCHOR\*** contains the symbol :TOP, the Who-Line will be anchored at the top of the screen; if it contains the symbol :BOTTOM it will be anchored at the bottom of the screen. If **\*WHO-LINE-ANCHOR\*** contains the symbol :LEFT, it will be anchored to the left side of the display; if it contains the symbol :CENTER it will be centered on the screen; if it contains the symbol :JUSTIFY it will run the width of the screen; if it contains the symbol :RIGHT it will be anchored to the right side of the screen. Finally, if **\*WHO-LINE-ANCHOR\*** is a POSITION, it will be used as the lower left corner of the Who-Line. The default value is (:CENTER :BOTTOM).

**\*WHO-LINE-NAME-FONT\*** [Variable]

The font used to display the names of the items in the who-line. The default is HELVETICA 8 BOLD.

**\*WHO-LINE-VALUE-FONT\*** [Variable]

The font used to display the values in the who-line. The default is GACHA 8.

**\*WHO-LINE-COLOR\*** [Variable]

The color of the Who-Line. Legal values are the keywords :WHITE and :BLACK. The default is :WHITE.

**\*WHO-LINE-BORDER\*** [Variable]

The border width of the Who-Line window. The default is 2.

**\*WHO-LINE-TITLE\*** [Variable]

The title of the Who-Line window. The default is NIL.

**\*WHO-LINE-DISPLAY-NAMES?\*** [Variable]

If **\*WHO-LINE-DISPLAY-NAMES?\*** is true, the names of items in the who-line will be displayed; otherwise they will not be shown. The default value is T.

**\*WHO-LINE-UPDATE-INTERVAL\*** [Variable]

The number of milliseconds between updates of the who-line. The default is 100 milliseconds.

### Installing new Who-Line options

Changing the above variables has no direct effect on the who-line. These values need to be installed in the Who-Line before they can take effect.

**(INSTALL-WHO-LINE-OPTIONS)** [Function]

INSTALL-WHO-LINE-OPTIONS installs the above options in the Who-Line, and updates the Who-Line accordingly.

The Who-Line supports an easy way to interactively add or remove entries. If you click on the Who-Line while holding down the EDIT or CONTROL key, you will be given a chance to add or remove items from the Who-Line.

**\*WHO-LINE-ENTRY-REGISTRY\*** [Global Variable]

A list of all known Who-Line entries. This is used to construct the menu of possible new entries for the Who-Line.

### Who-Line process state

The who-line entry **\*WHO-LINE-TTY-STATE-ENTRY\*** tries to display the current state of the TTY process.

**\*WHO-LINE-TTY-STATE-ENTRY\*** [Variable]

A Who-Line entry that displays the "state" of the current TTY process in the Who-Line. The typical state of a process is the name of the function that is currently running in that process. This simple minded result can be altered by use of the following items.

[[NOTE: Because of the nature of the Lisp scheduler, this information is almost always out of date.]]

The Who-Line "state" can be explicitly controlled from code. If the special variable `*WHO-LINE-STATE*` is bound, its value is taken to be the state of that process. You can use this feature to provide visual indication of the state of your code by using the programming idiom:

```
(LET ((*WHO-LINE-STATE* indicator))
 (BLOCK) ;Give the Who-line a chance to run
 ...your-code...)
```

This will run the `...your-code...` with the Who-Line state of the process set to (the value of) *indicator*. The call to BLOCK insures that the Who-Line has a chance to update before `...your-code...` is run.

`*WHO-LINE-STATE-UNINTERESTING-FNS*` [Global Variable]

If there is no declared who-line state (via a WITH-WHO-LINE-STATE form), then the name of the function that is currently running is used as the who-line state. However, if the function is on the list `*WHO-LINE-STATE-UNINTERESTING-FNS*`, the function that called it is used instead. The default value of `*WHO-LINE-STATE-UNINTERESTING-FNS*` is (BLOCK AWAIT.EVENT).

WHO-LINE-STATE [Property]

If the function that is currently running has a WHO-LINE-STATE property, the value of that property is used as the who-line state. This is used to convert functions like \TTYBACKGROUND to meaningful values like "TTY wait".

(WHO-LINE-REDISPLAY-INTERRUPT) [Function]

Updates the Who-Line. It is intended that this function be installed on an interrupt character, so that the user can easily force an update of the Who-Line. For example,

```
(ADVISE 'CONTROL-T 'BEFORE ' (WHO-LINE-REDISPLAY-INTERRUPT))
```

will cause a ^T interrupt to update the Who-Line as well as its current behavior of printing state information in the Prompt window. Alternatly, you can define a new interrupt character that will force an update of the Who-Line;

```
(INTERRUPTCHAR (CHARCODE ^U) ' (WHO-LINE-REDISPLAY-INTERRUPT) 'MOUSE)
```

will cause the Who-Line to be updated whenever the user hits a ^U.

### Other interesting things

`*WHO-LINE-DIRECTORIES*` [Global Variable]

A list of interesting directories used to generate a pop-up menu of directories to connect to when you select the DIRECTORY item in the Who-Line. The default value is a list containing just your LOGINHOST/DIR. When the Who-Line notices that you have changed your connected directory, it updates this list to contain the new directory.

(CURRENT-TTY-PACKAGE) [Function]

Returns the name of the package of the current TTY process. This function is used in the default value of `*WHO-LINE-ENTRIES*`.

(CURRENT-TTY-READTABLE-NAME) [Function]

Returns the name of the readtable of the current TTY process, or the string "Unknown" if it can't figure out the name. This function is used in the default value of `*WHO-LINE-ENTRIES*`.

(SET-PACKAGE-INTERACTIVELY)

[Function]

Pops up a menu of currently defined packages. If the user selects one of them, the current package is changed to the selected package.

(SET-READTABLE-INTERACTIVELY)

[Function]

Pops up a menu of currently known readtables. If the user selects one of them, the current readtable is changed to the selected readtable.



---

---

## WINK

---

---

By: Larry Masinter (Masinter.pa@Xerox.com)

This is a file containing bitmap demos. To bring up a bitmap of Marilyn Monroe winking, type:

(MARILYN)

This file also has bitmaps EINSTEIN and LINCOLN:



"AL"

"ABE"

---

---

## WORDFNS

---

---

By: Ron Kaplan (Kaplan.pa@Xerox.com)  
Becky Burwell (Burwell.pa@Xerox.com)

Uses: SETSTRINGLENGTH

This document last edited on August 19, 1988.

### INTRODUCTION

WORDFNS is a set of functions for manipulating files of words. There are functions to do the following: sort files, manipulate sorted files, provide common i/o functions for word files, provide mapping and translation mechanisms, provide common translation functions, and provide packaged mapping utilities.

The idea behind the mapping mechanism is that you can translate a file or list of files by specifying a read function to operate on each chunk of a file (the obvious two chunks are words and lines). You can specify file specific translation functions, default functions (when file specific functions are not provided) and common translation functions for all files. The input to the first translation function is the result of applying the read function to an input stream open on a file. The output of the first translation function is passed as input to the second translation function, etc.

### USE

Note: for any file, if NIL or T is specified then the results are printed in the executive window.

### Sorting Files

(SORTWORDFILE *IFILES* *OFILE* *COMMONTRANSFNS* *DEFAULTTRANSFNS* *READFN*  
*COMMONCOMPAREFN* *KEEPDUPLICATES* *FIELDS* *SEPARATOR* *REVERSEORDERFLG*  
*FASTFLG*) [Function]

The functions sorts the words on *IFILES* and stores the result back on *OFILE*. Th duplicates are eliminated unless *KEEPDUPLICATES* is non-NIL. For a description of the function of the arguments *COMMONTRANSFNS*, *DEFAULTTRANSFNS* and *READFN* see the section entitled "Translation Mechanisms". The argument *FIELDS* is used to specify the sorting order. The separator of the fields is specified in *SEPARATOR*. If *REVERSEORDERFLG* is T the result of the sort is reversed. *FASTFLG* set T causes the sort to caches the fields by consing allowing for a quicker sort (but consumes memory).

*FIELDS* is one of: NIL, a list of field numbers or else a list of one, two or three element lists of the form: FieldNumber Type CompareFn where type is either STRING (the default) or NUMBER. The default comparefn for STRING is ALPHORDER; for NUMBER is NUMORDER [Argument]

*SEPARATOR* is one of the following: a character string, a bittable, a list of single character atoms or numbers or one of the special atoms WHITESPACE (indicating a space or tab) or the atom TAB. The default is WHITESPACE.

[Argument]

>> should I put NUMORDER and GetNthField here?<<

Note: two related functions, NUMORDER and GetNthField, are described in the miscellaneous section.

### Functions for use with sorted files

In each of the following functions:

*COMMENTFILE* contains the details of the result of the function (for example, the number of strings that were read in from each file) [Argument]

(COMMONSORTEDFILES *file1 file2 ofile COMMENTFILE*) [Function]

Computes the intersection of two sorted files, *file1* and *file2* and the results are stored on *ofile*. The files are read a line at a time. The value is the full name of *ofile*.

(COMPARESORTEDFILES *file1 file2 ofile IMINUS2 2MINUS1*

*COMMENTFILE COMMENT*) [Function]

The two sorted files, *file1* and *file2*, are compared a line at a time. The common lines are stored on *ofile*. The output is in two columns: the left column for those lines in *file1* that do not exist in *file2* and the right column for those lines in *file2* that do not exist in *file1*. The two flags *IMINUS2* and *2MINUS1* are used to determine how the comparisons will be performed. If they are not specified they are both assumed to be T thus meaning that the comparison will be performed by subtracting *file2* from *file1* and *file2* with *file1* subtracted. If only one of *IMINUS2* or *2MINUS1* is specified then only the specified one way comparison will be done. *COMMENT* is intended to be a string which, by default, is the string "Comparison". This string is inserted at the top of the file. The value is the full name of *ofile*.

(DIFFSORTEDFILES *FILE1 FILE2 OUTFILE COMMENTFILE*) [Function]

The result of subtracting *FILE1* from *FILE2* is stored on *OUTFILE*. The files are read a line at a time. The value is the full name of *ofile*.

## I/O Functions

In the two major read functions, DREADLINE and DREADWORD, the SPACE, CR, LF and ^Z in any character set are interpreted to be the corresponding character in character set zero.

(DREADLINE *stream string skipsemicolons*) [Function]

Words are read from the word-stream *stream*, smashing them into *string*, which grows as needed. Returns NIL at EOF. Skips leading and trailing separators, and if *skipsemicolons* is non-NIL then sequences from ";" to EOL are treated as a composite separator or end-marker. Unlike DREADWORD (described later in this section), segments are separated only by EOL, so compounds are not split into components. Note that *stream* must have been set up so that BIN/READCCODE returns NULL on EOF.

(DREADLINESKIPSC *stream string skipsemicolons*) [Function]

Calls DREADLINE with *skipsemicolons* bound to T.

(DREADWORD *stream string*) [Function]

Words are read from the word-stream *stream*, smashing them into *string*, which grows as needed. Returns NIL at EOF. Skips leading and trailing separators, and treats sequences from ";" to EOL as a composite separator or end-marker. Unlike DREADLINE, segments are separated by space as well as EOL, so splits compounds into components. Note that *stream* must have been set up so that BIN/READCCODE returns NULL on EOF.

(INPUTWORDSTREAM *FILE NOPRINT*) [Function]

Returns a stream that is guaranteed to be open for word-reading (e.g. using DREADLINE or DREADWORD) at the beginning of *FILE*. If *NOPRINT* is NIL then the fullname of the file will be output.

(OUTPUTWORDSTREAM *FILE*) [Function]

Returns and opens a stream for the output of words (sequential text) guaranteed closed when reset context is exited and deleted if there is an error.

## Translation Mechanisms

Translation mechanisms are supplied to allow great flexibility in translating one or more files which may be in different formats and have unique translations applied to them. To specify how a file is to be read a read function (READFN) can be specified. Two common read functions, described previously, are DREADLINE and DREADWORD.

As mentioned earlier each file can have unique or common translation functions. A translation function is a function which takes two arguments: a string (the input to be translated) and an optional scratch string which can be destructively modified. The output of the translation function is one of the following: a string, the value T or the value NIL. Readers may wish to refer to the LispUsers module SETSTRINGLENGTH. The special value T denotes that the output is the same as the input. A value of NIL means that nothing will be kept.

**>> a better name for translation set <<**

**>> what is the syntax?<<**

*IFILES* is either a single file name, a single translation set, a list of file names or a list of translation sets. Translation sets have the form: (READFN READFN TRANFN1 TRANFN2 ...). (Note the first element of the list is the actual atom READFN and the second element [Argument]

*READFN* is the read function that is used for reading the files. It is passed a stream. Unless specified otherwise, the default read function is DREADWORD. [Argument]

*DEFAULTTRANSFNS* is a single function or list of functions which is first applied to the first file. What is given to the translation function is determined by what the read function passed. Unless specified otherwise, the default read function is DREADWORD. The result of applying the first function in *DEFAULTTRANSFNS* is input to the second function in *DEFAULTTRANSFNS*. This result is then passed on for application to the functions in *COMMONTRANSFNS*. The special value T denotes that the output is the same as the input. [Argument]

*COMMONTRANSFNS* [Argument]

*DONTPRINT* is the argument which decides whether the details of the translation functions will be printed. By default it is NIL meaning that the details will be printed. [Argument]

(TRANSLATEWORDFILE *IFILES* *COMMONTRANSFNS* *DEFAULTTRANSFNS* *READFN* *DONTPRINT*) [Function]

TRANSLATEWORDFILE produces an output file by translating each word in (possibly a list of) *IFILES* through a translation function. List elements of files are paired with their own idiosyncratic translation function. Otherwise the *DEFAULTTRANSFNS* is used. *COMMONTRANSFNS* are applied to the results of the default or file-specific translations to produce the translation string. If any translation function returns NIL, that string is skipped. A translation function is assumed to be an identity if it returns T, which makes simple predicates easy.

(COLLECTWORDFILE *IFILES* COMMONTRANSFNS DEFAULTTRANSFNS READFN DONTPRINT)[Function]

Returns the list of non-NIL values of functions applied to words in *IFILES*.

(MAPWORDFILE *IFILES* COMMONTRANSFNS DEFAULTTRANSFNS MAPFN READFN DONTPRINT) [Function]

Maps mapping function over words in *IFILES*. Nothing is setup for output.

### Packaged Mapping Utilities

(LONGESTWORDS *FILES* COMMONTRANSFNS DEFAULTTRANSFNS READFN DONTPRINT)[Function]

The list of longest translated words in *FILES* is returned.

(SEXPRCOUNT *FILE* RDTBL ) [Function]

Returns the number of s-expressions in *FILE* using *RDTBL* to read.

(WORDCOUNT *IFILES* COMMONTRANSFNS DEFAULTTRANSFNS READFN DONTPRINT ) [Function]

The total number of translated words in *IFILES* is returned.

(FINDPREFIXES *IFILES* *OFIL*E PREFIXES BUTNOT READFN ) [Function]

FINDPREFIXES produces an output file *OFIL*E of those strings read by *READFN* from *IFILES* which match at least one prefix in the list of prefix strings *PREFIXES* and do not match any prefixes in the list of prefixstrings *BUTNOT*.

(FINDSUFFIXES *IFILES* *OFIL*E SUFFIXES BUTNOT NOCAPS READFN ) [Function]

FINDPREFIXES produces an output file *OFIL*E of those strings read by *READFN* from *IFILES* which match at least one suffix in the list of suffix strings *SUFFIXES* and do not match any suffixes in the list of suffix strings *BUTNOT*. If *NOCAPS* is specified then the match succeeds if the string does not have the first letter capitalized.

(FINDSUBSTRINGS *IFILES* *OFIL*E SUBSTRINGS READFN ) [Function]

FINDSUBSTRINGS produces an output file *OFIL*E of those strings read by *READFN* from *IFILES* which match at least one substring in the list of substrings *SUBSTRINGS*.

**Translation Functions**

(MIXEDCASEP *W*) [Function]

Returns *W* if it contains mixtures of uppercase and lowercase characters after the initial character.

(PROPERP *W*) [Function]

Returns *T* if the first characters of *W* is uppercase.

(NOTPROPERP *W*) [Function]

Returns *T* if the first characters of *W* is not uppercase.

(REVERSESTRING *W STR*) [Function]

Reverses *W* into *STR* and returns *STR*.

**Examples**

This example will printout all the lines that have either the prefix "re" or "no" but not the prefix "non".

```
(FINDPREFIXES '({dsk}Myfile T '("re" "no") '("non") (FUNCTION DREADLINE))
```

This example will output to file {Phylum}<Project>Suffixes all the words in the files {dsk}File1 and {dsk}File2 that end in "ion" that do not have the first letter capitalized.

```
(FINDSUFFIXES '({dsk}File1 {dsk}File2) '{Phylum}<Project>Suffixes "ion" NIL
T (FUNCTION DREADWORD))
```

**Miscellaneous Functions**

(GETNTHFIELD *STRING N SEPARATOR FIELDTYPE*) [Function]

The *N*th field in *STRING* is returned using *SEPARATOR* as the field separator and the field type is coerced to type *FIELDTYPE*.

*N* is a simple positive integer [Argument]

*SEPARATOR* is the same as that for SORTWORDFILE [Argument]

*FIELDTYPE* is either the atom NUMBER or the atom STRING and indicates how the type that the field should be coerced to. The default *FIELDTYPE* is STRING.

**Example**

```
(GETNTHFIELD "So long and thanks for all the fish" 4 'WHITESPACE 'STRING)
```

returns the string "thanks".

```
(GETNTHFIELD "Joe Smith/5551212/12 Pleasant Lane/" 2 "/" 'NUMBER)
```

returns the integer 5551212.

```
(DCOPYSTRING W STR)
```

[Function]

Copies string *W* into string *STR* and returns *STR*.

```
(NUMORDER NUMBER1 NUMBER2)
```

[Function]

Returns >>??<<

**Example**

```
(SETQ MyString (ALLOCSTRING 1))
```

```
(DCOPYSTRING "This is a much longer string" MyString)
```



## XCL-BRIDGE

By: Jan Pedersen (pedersen.PA @ Xerox.com)

XCL-BRIDGE is a module that assists in the transformation of ascii Common Lisp source files to Lisp managed files and vice versa. In the text-to-managed-file direction, user interaction is employed to repair read-in forms before establishing a resident image of a Lisp managed file. In the managed-to-text-file direction, a few simple transforms are employed to translate common filepackagecoms to equivalent Common Lisp forms.

All entry points are external to the "XCL" package.

(XCL:TEXT-TO-MANAGED-FILE *pathname filename &key (package "USER")*  
(*readtable "XCL"*) (*read-base 10*) (*combine-comments t*) [Function]

Reads an ascii lisp source file named by "pathname" and converts it to a managed file with rootname "filename". The package, readtable, and read-base employed to read the ascii file may be specified via keywords arguments, or defaulted, as shown. If the reader environment arguments are defaulted and the source file has a emacs-style "mode line", then the package and read-base will be as indicated by the "mode line". The "combine-comments" keyword controls whether adjacent comments at the same ";" level should be combined when generating sedition-style comments for the converted file.

Note that forms are only read from the `ascii.lisp` source file, not evaluated. It is assumed that the converted file should be made (via `"il:makefile"`) and compiled before any evaluation should be attempted.

Text-to-managed-file proceeds incrementally and interactively to convert the specified file. First all the forms are read, and presented to the user for editing (via Sedit). If the user accepts this primary phase, a filecoms is generated and again, presented to the user for editing. If the user accepts the generated filecoms, a file (and its contained definitions) is instantiated, completing the conversion.

```
(XCL:MANAGED-TO-TEXT-FILE filename pathname &key (package "USER")
 (readtable "XCL") (print-base 10))
```

[Function]

Prints a managed file, with rootname "filename", whose source definitions must be resident, to an ascii file "pathname" in a form suitable for reading by any Common Lisp reader. The read-print environment of the managed file may be overwritten via the keyword arguments "package", "readtable", and "print-base". Many Interlisp "filepackagecoms" are translated to their Common Lisp equivalents. For example, "il:declare:" forms are transformed to "eval-when" forms and "il:files" forms are transformed to "require" forms. As an additional convenience, defdefiners are printed as equivalent defmacros.



*Venue*

# *LOOPS Library Modules Manual*

November, 1991  
Medley Release

---

Address comments to:  
Venue  
User Documentation  
1549 Industrial Road  
San Carlos, CA 94070  
415-508-9672

---

## LOOPS LIBRARY MODULES MANUAL

November, 1991

Copyright © 1988, 1991 by Venue.

All rights reserved.

LOOPS and Medley are trademarks of Venue.

UNIX® is a registered trademark of UNIX System Laboratories.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

---

The information in this document is subject to change without notice and should not be construed as a commitment by Venue. While every effort has been made to ensure the accuracy of this document, Venue assumes no responsibility for any errors that may appear.

Text was written and produced with Venue text formatting tools; Xerox printers were used to produce text masters. The typeface is Classic.

# TABLE OF CONTENTS

|                                        |    |
|----------------------------------------|----|
| PREFACE .....                          | v  |
| GAUGES.....                            | 1  |
| Description/Introduction .....         | 1  |
| Prerequisites .....                    | 2  |
| Installation/Loading Instructions..... | 2  |
| Application/Module Functionality.....  | 3  |
| Gauge Classes .....                    | 3  |
| Gauge Methods .....                    | 15 |
| Examples .....                         | 18 |
| Limitations .....                      | 19 |
| MASTERSCOPE .....                      | 21 |
| Description/Introduction .....         | 21 |
| Installation/Loading Instructions..... | 21 |
| Relations .....                        | 21 |
| Limitations .....                      | 24 |
| VIRTUAL COPIES .....                   | 27 |
| Description/Introduction .....         | 27 |
| Installation/Loading Instructions..... | 27 |
| Application/Module Functionality.....  | 27 |
| Overview of Operation .....            | 27 |
| Operands .....                         | 28 |
| Example .....                          | 30 |

## TABLE OF CONTENTS

---

|                   |    |
|-------------------|----|
| Limitations ..... | 31 |
| References.....   | 32 |

[This page intentionally left blank]

---

## Overview of the Manual

---

The *LOOPS Library Modules Manual* describes the Library Modules for Venue's Lisp Object-Oriented Programming System, LOOPS. These Library Modules, which can be loaded into Medley, provide additional functionality to LOOPS.

This manual describes the current release of the LOOPS Library Modules, which run under Medley.

---

## Organization of the Manual and How to Use It

---

This manual is divided into chapters, with each chapter focussing on a particular Library Module. A Table of Contents is included to help you find specific material.

---

## Conventions

---

This manual uses the following conventions:

- Case is significant in LOOPS and Medley. All selectors, methods, arguments, etc., must be typed as shown. Typically, this means that method names are capitalized and variables are not.
- You need to use an Interlisp Exec to enter all exec expressions.
- Arguments appear in italic type.
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a method appears as follows:

(*\_ self* **Selector** *Arg1 Arg2*)

- Examples are shown in the Interlisp Exec and appear in the following typeface:

89← (←LOGIN)

- All examples are typed into an Interlisp Exec. This is the recommended Exec for all LOOPS expressions.
- Methods with an exclamation mark (!) suffix usually perform operations deeply into class structure instead of only on a given object.
- Methods with a question mark (?) suffix usually are predicates; that is, truth functions.
- Methods often appear in the form **ClassName.SelectorName**.

- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

This manual describes the LOOPS items (functions, methods, etc.) by using the following template:

|                  |                                                                                                                                                                                                                                                                                       |                 |             |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|-------------|
| Purpose:         | Gives a short statement of what the item does.                                                                                                                                                                                                                                        |                 |             |
| Behavior:        | Provides the details of how the item operates.                                                                                                                                                                                                                                        |                 |             |
| Arguments:       | Describes each argument in the following format:<br><br><table> <tr> <td><i>argument</i></td><td>Description</td></tr> </table>                                                                                                                                                       | <i>argument</i> | Description |
| <i>argument</i>  | Description                                                                                                                                                                                                                                                                           |                 |             |
| Returns:         | States what the item returns, and does not appear if the item does not return a value. The phrase "Used as a side effect only." means that the purpose of the item is to perform a computation or action that is independent of any returned value, not to return a particular value. |                 |             |
| Categories:      | A way to group related methods. For example, all the methods related to Masterscope on the class <b>FileBrowser</b> have the category Masterscope, not <b>FileBrowser</b> . This item appears only for methods.                                                                       |                 |             |
| Specializes:     | The next higher class in the class hierarchy that contains a method with the same selector. For example, <b>RectangularWindow.Open</b> can specialize <b>Window.Open</b> . This appears only for methods.                                                                             |                 |             |
| Specializations: | The next lower class in the class hierarchy that contains a method with the same selector. For example, <b>Window.Open</b> is a specialization of <b>RectangularWindow.Open</b> . This appears only for methods.                                                                      |                 |             |
| Example:         | An example is often included to show how to use the item and what result it produces. Some examples may appear differently on your system, depending on the settings of various print flags. See the <i>LOOPS Reference Manual</i> for details.                                       |                 |             |

---

## References

---

The following books and manuals augment this manual.

*LOOPS Reference Manual*

*LOOPS Release Notes*

*LOOPS Users' Modules Manual*

*Interlisp-D Reference Manual*

*Common Lisp: the Language* by Guy Steele

*Common Lisp Implementation Notes*

*Lisp Release Notes*

*Lisp Library Modules Manual*



---

## Description/Introduction

---

Gauges are an important part of the LOOPS user interface for both developers and end users. Gauges assist in understanding the dynamic nature of the programs. This is in contrast to the more typical case of debugging programs using static means. In the creation of user-friendly interfaces, you can use gauges to display, in analog or digital form, various data that may be changing. Also, by employing active gauges, you can provide a convenient way to interact with a system.

One of the features of gauges is the ease with which you can use them in a system. In more traditional languages, if you want to understand how a variable is changing over the course of a computation, you must make modifications in your program wherever you want to begin or end the examination of a variable. Given the capabilities of active values used by gauges, you need only attach or detach a gauge to the data that you are interested in monitoring.

The following types of gauges are available:

- Meter; a circular instrument that wraps around any number of times.
- Dial; a bounded dial, like an automobile speedometer.
- LCD; a gauge that uses the entire window to display a value.
- Scale; a horizontal or vertical display of a gauge.
- ActiveScale; a scale that allows you to change the gauge value.

Gauges are an example of the combination of programming capabilities within LOOPS. The different types of gauges are defined within the context of an inheritance lattice. This allows the more general functionality and variables to be allocated to more general gauge classes, with specific functionality placed in more restricted classes. You can also see the use of mixins to add a small amount of functionality to several different classes of gauges.

Note: Mixins are classes that are used only in conjunction with another class to create a subclass.

The methods within gauges are built upon both function calling and message sending. Gauges are "attached" to objects through the mechanism of active values. Since gauges are built upon the mechanism of active values, gauges can only be attached to data within objects. It is not possible to use gauges to monitor any arbitrary Lisp variable.

Prerequisites

The default font for gauges is Modern 10.

Installation/Loading Instructions

Gauges are divided among several different files to allow you to load only those objects and functions that you need. The table below lists the files to load for each type of gauge. The filecoms for each file will try to load any other required gauge files from **LOOPSLIBRARYDIRECTORY**. The file GAUGES.DFASL and either GAUGEINSTRUMENTS.DFASL or GAUGEALPHANUMERICS.DFASL will always be loaded; other files may also be loaded.

| Gauge        | File to load             |
|--------------|--------------------------|
| LCD          | GAUGEALPHANUMERICS.DFASL |
| METER        | GAUGEMETERS.DFASL        |
| DIAL         | GAUGEDIALS.DFASL         |
| SCALE        | GAUGESCALES.DFASL        |
| ACTIVE SCALE | GAUGEACTIVE.DFASL        |

Additionally, the file GAUGESELFSCALEMIXIN.DFASL can be loaded to add the class **SelfScaleMixin**, and GAUGEALARMS.DFASL can be loaded to add the class **AlarmMixin**.

To load the required files, first set the value of **LOOPSDIRECTORY** to include the directory where the gauges files are stored, then type the following expression in the Executive:

(LOAD 'FILENAME)

To load all of the gauges, load the file GAUGELOADER and then enter (LOADGAUGES). GAUGELOADER also sets the variables: **GAUGEFILES** and **GaugeClasses**.

(LOADGAUGES LDFLG SOURCES?FLG) [Function]

|            |                                                                                                                                                                                                                                                                                                         |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Loads all the gauges.                                                                                                                                                                                                                                                                                   |
| Behavior:  | Assumes that all of the gauge files are on the <b>LOOPSDIRECTORY</b> search path.<br><br>All the gauge files will be loaded based upon the settings of <i>LDFLG</i> and <i>SOURCES?FLG</i> . A <b>FILESLOAD</b> expression is built up and evaluated.                                                   |
| Arguments: | <i>LDFLG</i> Can be NIL, PROP, or SYSLOAD. See the <b>LDFLG</b> discussion under loading in the <i>Interlisp-D Reference Manual</i> .<br><br><i>SOURCES?FLG</i> Can be NIL or T. If NIL, this attempts to load the compiled files before trying to load the sources. If T, only the sources are loaded. |
| Returns:   | Used for side effect only.                                                                                                                                                                                                                                                                              |

**GAUGEFILES**

[Variable]

Behavior: Initialized to (GAUGEACTIVE GAUGEALARMS GAUGEALPHANUMERICS GAUGEBOUNDEDMIXIN GAUGEDIALS GAUGEDIGIMETER GAUGEDIGISCALE GAUGEINSTRUMENTS GAUGEMETERS GAUGES GAUGESCALES GAUGESELFSCALEMIXIN)

**GaugeClasses**

[Variable]

Behavior: Initialized to (GaugeAV ActiveGaugeMixin Gauge AlarmMixin BoundedMixin SelfScaleMixin)

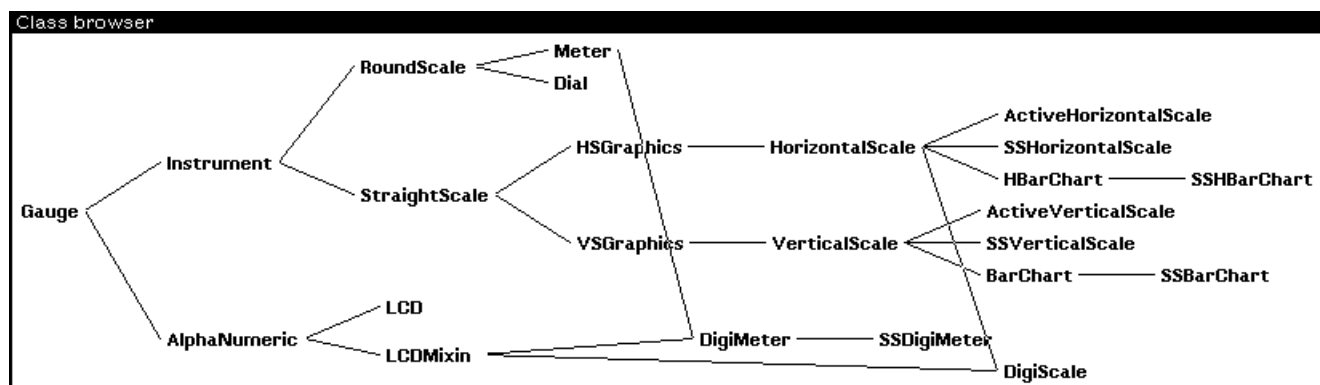
Call (Browse **GaugeClasses**) to open a browser of all of the gauge classes.

**Application/Module Functionality**

This section describes the gauge classes and methods.

**Gauge Classes**

This section describes the available gauges shown in the following browser.



Note: The browser does not include the optional mixin classes.

Within the class description of each class, the instance variables and class variables that are specializations only because they have different default values are not listed.

| Name                         | Type          | Description                                                                                                     |
|------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------|
| <b>ActiveGaugeMixin</b>      | AbstractClass | A gauge class that allows you to set the value of the variable being monitored with the cursor, via a SET menu. |
| <b>ActiveHorizontalScale</b> | Class         | An active gauge that displays the value on a horizontal scale.                                                  |
| <b>ActiveVerticalScale</b>   | Class         | An active gauge that displays its value on a vertical scale.                                                    |
| <b>AlarmMixin</b>            | AbstractClass | A mixin that adds alarm functionality to any gauge.                                                             |
| <b>AlphaNumeric</b>          | AbstractClass | A gauge that gives an alphanumeric display of a value.                                                          |

|                          |               |                                                                                                                                               |
|--------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>BarChart</b>          | Class         | A gauge that displays more than one <b>VerticalScale</b> side-by side.                                                                        |
| <b>BoundedMixin</b>      | AbstractClass | Creates a bounded scale for <b>displayVal</b> ; to be used as a mixin for instruments.                                                        |
| <b>Dial</b>              | Class         | A bounded dial, like an automobile speedometer.                                                                                               |
| <b>DigiMeter</b>         | Class         | A gauge that displays both an <b>LCD</b> and a meter.                                                                                         |
| <b>DigiScale</b>         | Class         | A gauge that displays both an <b>LCD</b> and a horizontal scale.                                                                              |
| <b>Gauge</b>             | AbstractClass | A class for objects that present a dynamic graphical image of a LOOPS value.                                                                  |
| <b>GaugeAV</b>           | Class         | An active value associated with a gauge.                                                                                                      |
| <b>HBarChart</b>         | Class         | A gauge that displays more than one <b>HorizontalScale</b> side-by side.                                                                      |
| <b>HorizontalScale</b>   | Class         | A labeled, bounded scale with a bar that fills to the right.                                                                                  |
| <b>HSGraphics</b>        | AbstractClass | Gauge that is displayed in the form of a single horizontal scale or bar.                                                                      |
| <b>Instrument</b>        | AbstractClass | A numeric gauge that is externally scaled by <b>inputLower</b> and <b>inputRange</b> and scaled internally by <b>lower</b> and <b>range</b> . |
| <b>LCD</b>               | Class         | Differs from <b>AlphaNumeric</b> in that the entire gauge window is the printing region.                                                      |
| <b>LCDMixin</b>          | AbstractClass | Computes print region differently from <b>LCD</b> .                                                                                           |
| <b>Meter</b>             | Class         | A circular instrument that wraps around any number of times.                                                                                  |
| <b>RoundScale</b>        | AbstractClass | Abstract Class for instruments with circular (arc) scales.                                                                                    |
| <b>SelfScaleMixin</b>    | AbstractClass | Provides for the gauge to rescale according to the reading.                                                                                   |
| <b>SSBarChart</b>        | Class         | A self-scaling version of <b>BarChart</b> .                                                                                                   |
| <b>SSDigiMeter</b>       | Class         | A self-scaling version of <b>DigiMeter</b> .                                                                                                  |
| <b>SSHBarChart</b>       | Class         | A self-scaling version of <b>HBarChart</b> .                                                                                                  |
| <b>SSHorizontalScale</b> | Class         | Gauge that is displayed in the form of a single scale or bar which rescales itself accordingly.                                               |
| <b>SSVerticalScale</b>   | Class         | Gauge that is displayed in the form of a single vertical scale or bar which rescales itself accordingly.                                      |
| <b>StraightScale</b>     | AbstractClass | Abstract Class for instruments with straight scales.                                                                                          |
| <b>VSGraphics</b>        | AbstractClass | Gauge that is displayed in the form of a single vertical scale or bar.                                                                        |
| <b>VerticalScale</b>     | Class         | Gauge that is displayed in the form of a single vertical scale or bar.                                                                        |

---

**ActiveGaugeMixin**

[Class]

Description: A gauge class that allows you to set the value of the variable being monitored with the cursor, via a **SET** menu.

MetaClass: AbstractClass

|                     |                                                                              |
|---------------------|------------------------------------------------------------------------------|
| Supers:             | Object                                                                       |
| Class Variables:    | None.                                                                        |
| Instance Variables: | <b>cursor</b> The cursor to use when changing the scale; the default is NIL. |

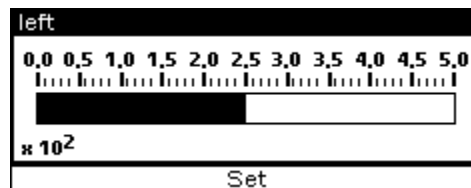
**ActiveHorizontalScale**

[Class]

|                     |                                                                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description:        | An active gauge that displays the value on a horizontal scale. This gauge shows the value of the data it is connected with and allows you to change that data with the gauge.   |
| MetaClass:          | Class                                                                                                                                                                           |
| Supers:             | ActiveGaugeMixin, HorizontalScale                                                                                                                                               |
| Class Variables:    | None.                                                                                                                                                                           |
| Instance Variables: | <b>cursor</b> Cursor to use when changing the scale; its property <b>:initform</b> is set to <b>HorizontalAGCursor</b> .                                                        |
| Example:            | These gauges have an attached menu at the bottom of the gauge. When you position the cursor over this menu and press a mouse button, the cursor changes to the following shape: |



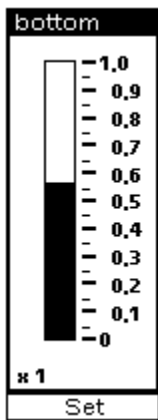
While the left button is held down, the system tracks movements of the cursor and changes the value that the gauge is monitoring.

**ActiveVerticalScale**

[Class]

|                     |                                                                                                                        |
|---------------------|------------------------------------------------------------------------------------------------------------------------|
| Description:        | Similar to <b>ActiveHorizontalScale</b> , except that a vertical scale is used.                                        |
| MetaClass:          | Class                                                                                                                  |
| Supers:             | ActiveGaugeMixin, VerticalScale                                                                                        |
| Class Variables:    | None.                                                                                                                  |
| Instance Variables: | <b>cursor</b> Cursor to use when changing the scale; its <b>:initform</b> property is set to <b>VerticalAGCursor</b> . |
| Example:            | Similar to <b>ActiveHorizontalScale</b> . When setting, the cursor changes to the following shape:                     |





**AlarmMixin** [Class]

Description: A mixin that adds alarm functionality to any gauge. An alarm is defined as warning object that is set off when the value being monitored falls outside of the specified range. The gauge flashes and stays inverted when the alarm is tripped.

**CAUTION**

When a new class of gauges is created that will use the properties of **AlarmMixin**, **AlarmMixin** should be the first class on the Supers list of the new class. This guarantees that the **AlarmMixin.Set** method is invoked.

- MetaClass: AbstractClass
- Supers: Object
- Class Variables: MiddleButtonItem
- Instance Variables:
  - lowTripPoint** Alarm is triggered when reading goes below this point.
  - hiTripPoint** Alarm is triggered when reading goes above this point.
  - flashNumber** Number of times alarm will flash when it is tripped.
  - flashInterval** Interval in milliseconds between flashes.

**AlphaNumeric** [Class]

- Description: This class contains some of the methods and data for the LCD classes. These gauges can display any type of character, letters, or numbers.
- MetaClass: AbstractClass
- Supers: Gauge
- Class Variables: None.

Instance Variables: **precision** Number of characters displayed in the reading. The default value is 5.

**BarChart**

[Class]

Description: A gauge that can display more than one **VerticalScale** at once, side-by side.

MetaClass: Class

Supers: VerticalScale

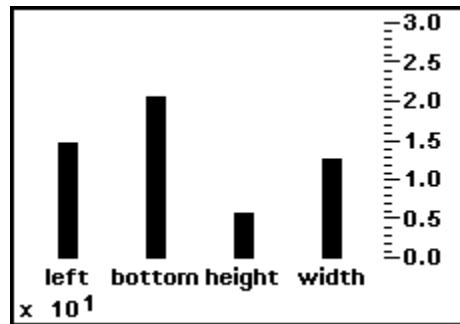
Class Variables: None.

Instance Variables: **maxLabelWidth**  
Maximum width of labels on each bar. Default value is 0 which means no limit.

**scaleLeft**  
Offset within the gauge window from the left for the leftmost bar. Default value is 3.

**scaleBottom**  
Offset within the gauge window from the bottom for all the bars. Default value is 30.

Example: Here is a **BarChart** showing the size and shape of a window. It is displaying the values 15, 21, 13, and 6.

**BoundedMixin**

[Class]

Description: This mixin is a super of the scale classes and **Dial**. If a gauge that has **BoundedMixin** as a super class tries to display a new setting that is outside of the range of the gauge, the gauge will display the minimum or maximum value as appropriate and place a "???" in the window.

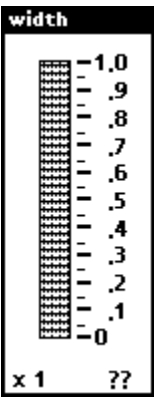
MetaClass: AbstractClass

Supers: Object

Class Variables: None.

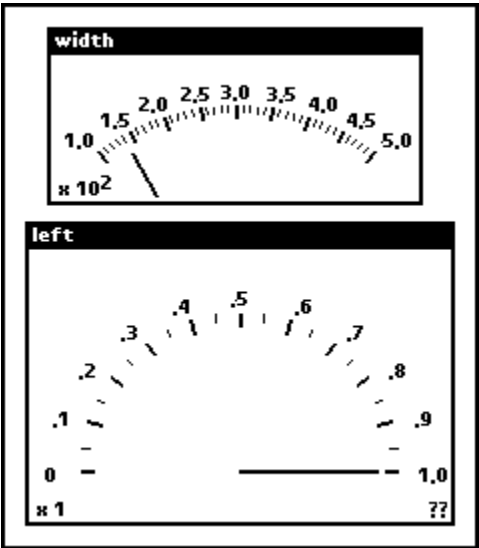
Instance Variables: None.

Example: Here is a vertical scale that displays a reading greater than its maximum.



**Dial** [Class]

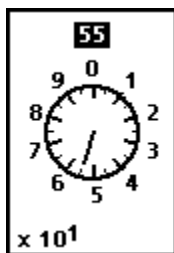
- Description: A bounded dial, like an automobile speedometer.
- MetaClass: Class
- Supers: BoundedMixin, RoundScale
- Class Variables: None.
- Instance Variables: This class specializes the same instance variables as **RoundScale**.
- Example: The angle of the arc changes with the shape of the window.



**DigiMeter** [Class]

- Description: A gauge that combines both a meter and an LCD.
- MetaClass: Class
- Supers: Meter, LCDMixin
- Class Variables: None.
- Instance Variables: **spaceForLCD**  
Vertical space required by LCD within the gauge. Defaults to 30.
- Example: This **DigiMeter** is displaying 55.



**DigiScale**

[Class]

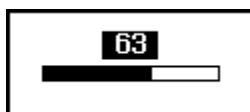
Description: A gauge that combines both a horizontal scale having no ticks and an LCD.

MetaClass: Class

Supers: HorizontalScale, LCDMixin

Class Variables: None.

Example: This **DigiScale** is displaying 63 with its scale set from 0 to 100.

**Gauge**

[Class]

Description: A class for objects that present a dynamic graphical image of a LOOPS value. This class provides most of the methods for using gauges.

MetaClass: AbstractClass

Supers: Window

Class Variables: **LeftButtonItem**  
Menu options associated with the left mouse button.

**MiddleButtonItem**  
Menu options associated with the middle mouse button.

Instance Variables: **reading** External value of reading. The default value is 0.

**containedInAV**  
Active value that connects the gauge to the data it is monitoring. It should be an instance of the class **GaugeAV**.

**font** Font that is used by a gauge; default value is (Modern 10).

**width** Width of a gauge; has property **min**, which specifies the minimum width for a gauge.

**height** Height of a gauge; has property **min**, which specifies the minimum height for a gauge.

**GaugeAV**

[Class]

Description: An active value that is associated with a gauge.

MetaClass: Class

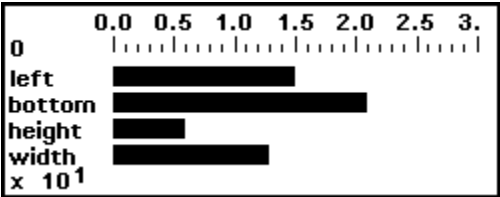
Supers: LocalStateActiveValue

|                     |                 |                                                                      |
|---------------------|-----------------|----------------------------------------------------------------------|
| Class Variables:    | None.           |                                                                      |
| Instance Variables: | <b>gauge</b>    | The gauge connected to this active value.                            |
|                     | <b>object</b>   | The object containing the variable associated with the active value. |
|                     | <b>propName</b> | The property name of the associated variable.                        |
|                     | <b>type</b>     | Data type of the associated variable.                                |
|                     | <b>varName</b>  | Name of the associated variable.                                     |

HBarChart

[Class]

|                     |                                                                                                                             |                                                                                        |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Description:        | A gauge that can display more than one <b>HorizontalScale</b> at once, side-by-side.                                        |                                                                                        |
| MetaClass:          | Class                                                                                                                       |                                                                                        |
| Supers:             | HorizontalScale                                                                                                             |                                                                                        |
| Class Variables:    | None.                                                                                                                       |                                                                                        |
| Instance Variables: | <b>maxLabelWidth</b>                                                                                                        | Maximum width of labels on each bar. Default value is 0 which means no limit.          |
|                     | <b>scaleLeft</b>                                                                                                            | Offset within the gauge window from the left for the leftmost bar. Default value is 3. |
|                     | Example: Here is an <b>HBarChart</b> showing the size and shape of a window. It is displaying the values 15, 21, 13, and 6. |                                                                                        |



HorizontalScale

[Class]

|                     |                                                                      |  |
|---------------------|----------------------------------------------------------------------|--|
| Description:        | A labeled, bounded scale with a bar that fills to the right.         |  |
| MetaClass:          | Class                                                                |  |
| Supers:             | HSGraphics                                                           |  |
| Class Variables:    | None.                                                                |  |
| Instance Variables: | None.                                                                |  |
| Example:            | This <b>HorizontalScale</b> is reading 350 on a scale from 0 to 500. |  |



HSGraphics

[Class]

|                     |                                                                                                                                                                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description:        | This class provides some of the methods for displaying the graphics of a horizontal scale.                                                                                                                                                                                                                          |
| MetaClass:          | AbstractClass                                                                                                                                                                                                                                                                                                       |
| Supers:             | StraightScale                                                                                                                                                                                                                                                                                                       |
| Class Variables:    | None.                                                                                                                                                                                                                                                                                                               |
| Instance Variables: | <b>scaleBottom</b> Bottom edge of scale in pixels. The default value is 10.<br><b>scaleLeft</b> Left edge of scale in pixels. The default value is 12.<br><b>scaleWidth</b> Width of inside of scale in pixels. The default value is 120.<br><b>scaleHeight</b> Height of scale in pixels. The default value is 15. |

**Instrument**

[Class]

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description:        | A class that provides additional methods and data for gauges that display only numerical data. This data is externally scaled by <b>inputLower</b> and <b>inputRange</b> , and scaled internally by <b>lower</b> and <b>range</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| MetaClass:          | AbstractClass                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Supers:             | Gauge                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Class Variables:    | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Instance Variables: | <b>ticks</b> Scale marks on the instrument; value is a number or NIL; <b>smallTicks</b> property indicates the number of smaller ticks between each large tick.<br><b>displayVal</b> Internal value relative to instrument.<br><b>range</b> Range for internal <b>displayVal</b> .<br><b>inputRange</b> Range for external reading.<br><b>lower</b> Lower bound for internal <b>displayVal</b> .<br><b>inputLower</b> Lower bound for external reading.<br><b>brushWidth</b> Scale factor for width of ticks, rays, and circles in pixels.<br><b>labels</b> The labels that will be displayed on the gauge.<br><b>labelScale</b> A dotted pair representing the sign and exponent of a reading.<br><b>spaceForLabelScale</b> Extra vertical space to display scale label. |

**LCD**

[Class]

|                     |                                                                                      |
|---------------------|--------------------------------------------------------------------------------------|
| Description:        | Differs from <b>LCDMixin</b> in that the entire gauge window is the printing region. |
| MetaClass:          | Class                                                                                |
| Supers:             | AlphaNumeric                                                                         |
| Class Variables:    | None.                                                                                |
| Instance Variables: | None.                                                                                |

Example: This **LCD** is displaying the string "Mumble", and has been **Shaped**to 120 x 60.



**LCDMixin** [Class]

Description: Computes printing region differently from LCD so that an LCD may be added into another window.

MetaClass: AbstractClass

Supers: AlphaNumeric

Class Variables: None.

Instance Variables: **precision** Number of characters displayed in the reading; the default value is 3. Its property is **readingRegion**; the default value is NIL.

**readingY** Y position of bottom of reading. The default value is 7.

**Meter** [Class]

Description: A circular instrument that wraps around any number of times. It displays a sign and exponent in the lower left corner of its window.

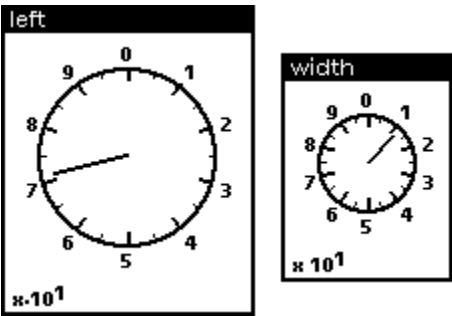
MetaClass: Class

Supers: RoundScale

Class Variables: None.

Instance Variables: This class specializes the same instance variables as **RoundScale**.

Example: The **Meter** on the left is displaying a negative value.



**RoundScale** [Class]

Description: Abstract Class for instruments with circular (arc) scales.

MetaClass: AbstractClass

Supers: Instrument

Class Variables: None.

Instance Variables: **needleLength** Radius of needle in pixels. The default value is 15.

|               |                                                                                                          |
|---------------|----------------------------------------------------------------------------------------------------------|
| <b>radius</b> | Radius of arc in pixels. The default value is 10.                                                        |
| <b>xc</b>     | x-coordinate window coordinate of center of arc. (See <b>DRAWARC</b> in the <i>Lisp Release Notes</i> .) |
| <b>yc</b>     | y-coordinate window coordinate of center of arc. (See <b>DRAWARC</b> in the <i>Lisp Release Notes</i> .) |

### **SelfScaleMixin** [Class]

|                     |                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description:        | Provides for the gauge to rescale according to the reading.                                                                                                   |
| MetaClass:          | AbstractClass                                                                                                                                                 |
| Supers:             | Object                                                                                                                                                        |
| Class Variables:    | None.                                                                                                                                                         |
| Instance Variables: | <b>lowScaleFactor</b><br>Rescales if reading shrinks so that it will fit more than <b>lowScaleFactor</b> times in <b>inputRange</b> . The default value is 5. |

### **SSBarChart** [Class]

|                     |                                             |
|---------------------|---------------------------------------------|
| Description:        | A self-scaling version of <b>BarChart</b> . |
| MetaClass:          | Class                                       |
| Supers:             | BarChart                                    |
| Class Variables:    | None.                                       |
| Instance Variables: | None.                                       |

### **SSDigiMeter** [Class]

|                     |                                              |
|---------------------|----------------------------------------------|
| Description:        | A self-scaling version of <b>DigiMeter</b> . |
| MetaClass:          | Class                                        |
| Supers:             | DigiMeter                                    |
| Class Variables:    | None.                                        |
| Instance Variables: | None.                                        |

### **SSHBarChart** [Class]

|                     |                                              |
|---------------------|----------------------------------------------|
| Description:        | A self-scaling version of <b>HBarChart</b> . |
| MetaClass:          | Class                                        |
| Supers:             | HBarChart                                    |
| Class Variables:    | None.                                        |
| Instance Variables: | None.                                        |

### **SSHorizontalScale** [Class]

|              |                                                                                                            |
|--------------|------------------------------------------------------------------------------------------------------------|
| Description: | Gauge that is displayed in the form of a single horizontal scale or bar which rescales itself accordingly. |
|--------------|------------------------------------------------------------------------------------------------------------|

MetaClass: Class  
Supers: VerticalScale  
Class Variables: None.  
Instance Variables: None.

---

**SSVerticalScale****[Class]**

Description: Gauge that is displayed in the form of a single vertical scale or bar which rescales itself accordingly.  
MetaClass: Class  
Supers: HorizontalScale  
Class Variables: None.  
Instance Variables: None.

---

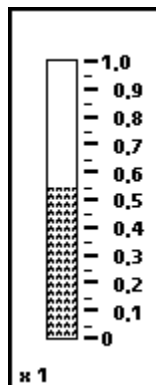
**StraightScale****[Class]**

Description: Abstract class for instruments with straight scales.  
MetaClass: AbstractClass  
Supers: BoundedMixin, Instrument  
Class Variables: None.  
Instance Variables: **shade** Shade of bar; numeric value from 0 to 65535. The default value is 65535, which is BLACKSHADE.

---

**VerticalScale****[Class]**

Description: Gauge that is displayed in the form of a single vertical scale or bar.  
MetaClass: Class  
Supers: VSGraphics  
Class Variables: None.  
Instance Variables: None.  
Example: This **VerticalScale** is displaying the value .55 and has its **Shade** set to 1258.



**VSGraphics****[Class]**

|                     |                                                                                                                                                                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description:        | Similar to <b>HSGraphics</b> but for vertical scales.                                                                                                                                                                                                                                                               |
| MetaClass:          | AbstractClass                                                                                                                                                                                                                                                                                                       |
| Supers:             | StraightScale                                                                                                                                                                                                                                                                                                       |
| Class Variables:    | None.                                                                                                                                                                                                                                                                                                               |
| Instance Variables: | <b>scaleBottom</b> Bottom edge of scale in pixels. The default value is 12.<br><b>scaleLeft</b> Left edge of scale in pixels. The default value is 15.<br><b>scaleWidth</b> Width of inside of scale in pixels. The default value is 15.<br><b>scaleHeight</b> Height of scale in pixels. The default value is 120. |

**Gauge Methods**

This section describes the available methods and functions which are used to manipulate gauges. In many cases, a particular gauge class specializes a method defined in the class **Gauge**. In this case, the specialized method definition is not explicitly defined; instead, this is noted in the Specializes/Specializations field of the description.

| Name               | Type   | Description                                                                  |
|--------------------|--------|------------------------------------------------------------------------------|
| <b>Attach</b>      | Method | Connects a gauge to an object.                                               |
| <b>Attached?</b>   | Method | Determines what the gauge is attached to.                                    |
| <b>ChangeFont</b>  | Method | Sets the gauge's instance variable <b>font</b> and updates the gauge.        |
| <b>Close</b>       | Method | Detaches the gauge and closes the window.                                    |
| <b>Destroy</b>     | Method | Destroys the gauge, detaching it first.                                      |
| <b>Detach</b>      | Method | Detaches the gauge from the variable it is attached to.                      |
| <b>Reset</b>       | Method | Resets the gauge's instance variable <b>reading</b> .                        |
| <b>SetScale</b>    | Method | Sets the scale for the gauge.                                                |
| <b>Shape</b>       | Method | Sweeps a new region.                                                         |
| <b>ShapeToHold</b> | Method | Shapes the gauge window to its smallest possible size.                       |
| <b>Update</b>      | Method | Reinitializes the gauge and its display window to reflect the current state. |

(← **self Attach** *obj varName propName type xOrPos y*) **[Method of Gauge]**

|            |                                                                                                                                                                                                                                                                                                                    |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Connects a gauge to an object.                                                                                                                                                                                                                                                                                     |
| Behavior:  | Displays the gauge on the screen and associates that gauge with the variable <i>varName</i> of <i>obj</i> . If <i>propName</i> is specified, the gauge will monitor the variable's property. If <i>xOrPos</i> and <i>y</i> are not specified, a small box will appear which must be positioned to place the gauge. |
| Arguments: | <i>obj</i> A pointer to the object to which the gauge is to be attached.                                                                                                                                                                                                                                           |

|                 |                                                                                                                                                                                                                                                                        |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>varName</i>  | The name of the instance variable, class variable, or method to which the gauge is to be attached.                                                                                                                                                                     |
| <i>propName</i> | If non-NIL, the gauge will be attached to this property.                                                                                                                                                                                                               |
| <i>type</i>     | One of IV, CV, or METHOD, within the object being connected to the gauge. If NIL, it defaults to IV.                                                                                                                                                                   |
| <i>xOrPos</i>   | A numerical value to specify where, in screen coordinates, the gauge will be placed on the display. If NIL, you are asked to place the gauge on the screen. This can be a number to specify the x coordinate or a position. If it is a number, also specify <i>y</i> . |
| <i>y</i>        | If <i>xOrPos</i> is not a position, this specifies the y coordinate in screen coordinates for the gauge.                                                                                                                                                               |

Returns: *self*

Specializations: **StraightScale.Attach** has an additional *shade* argument so that the shade of the scale may be specified at the time the gauge is attached. The following shows the argument list for this method:

(← (\$ instance OfHorizontalScale) **Attach** *obj varName shade propName type xOrPos y*)

The **Attach** methods for **BarChart**, **HBarChart**, and their subclasses take an additional *label* argument. If no *label* argument is given, the bar is labeled with *varName*. The *label* argument comes last, as follows:

(← (\$ instance OfBarChart) **Attach** *obj varName propName propName type xOrPos y label*)

---

(← *self* **Attached?** *don'tPrintFlg*) [Method of Gauge]

Purpose: Determines what a gauge is attached to.

Behavior: If *don'tPrintFlg* is non-NIL this returns the value of the gauge instance variable **containedInAV**. If *don'tPrintFlg* is NIL, the **object** and the **varName** the gauge is attached to will be printed in an attached window.

Arguments: *don'tPrintFlg*  
Suppresses displaying what the gauge is attached to.

Returns: NIL

---

(← *self* **ChangeFont** *newFont*) [Method of Gauge]

Purpose/Behavior: Sets the gauge's instance variable **font** to *newFont* and updates the gauge. If the gauge is too small for *newFont*, it is reshaped.

Arguments: *newFont* A font in which to display the gauge's text.

Returns: Previous value of **font**.

---

(← *self* **Close**) [Method of Gauge]

Purpose/Behavior: Detaches the gauge and closes the window.

Returns: CLOSED

---

(← *self* **Destroy**) [Method of Gauge]

Purpose/Behavior: Destroys the gauge, detaching it first before closing the window.



Returns: NIL

(← *self* **Detach**)

[Method of Gauge]

Purpose/Behavior: Detaches the gauge from the variable to which it is attached. This prints in an attached window that the gauge is being detached, and deletes all of the links connecting the gauge, active value, and object being monitored. Does not close the window.

Returns: NIL

(← *self* **Reset** *newReading*)

[Method of Gauge]

Purpose/Behavior: Sets the gauge's instance variable **reading** to *newReading* and updates the gauge. If the gauge is too small for *newReading* and it is **SelfScaling**, it is reshaped.

Arguments: *newReading* Sets the instance variable **reading** to *newReading*, and updates the gauge without going through any intermediate steps.

Returns: NIL if gauge is **AlphaNumeric** or **RoundScale**; otherwise *self*.

Specializations: **Alphanumeric.Reset**, **RoundScale.Reset**

Example: The following example causes the LCD to be redisplayed with the *newReading*:

```
13← (← ($ lcd1) Reset "New Title")
```

(← *self* **SetScale** *min max*)

[Method of Gauge]

Purpose/Behavior: Sets the scale for the gauge; computes the new scale values and redisplay if necessary.

Arguments: *min* Lowest value on scale.

*max* Highest value on scale.

Returns: *self*

(← *self* **Shape** *newRegion noUpdateFlg*)

[Method of Gauge]

Purpose/Behavior: If *newRegion* is NIL, you are prompted to sweep out a region which has a minimum sized based upon a **min** property of **IV width** and **height:;min**. If *newRegion* is non-NIL, it is first checked to guarantee that it is at least as large as **width:;min** by **height:;min**.

Arguments: *newregion* List specifying the external coordinates of the window in which the gauge is displayed; list is of the form (left, bottom, width, height).

*noUpdateFlg* If NIL, reshapes the gauge.

Returns: NIL

Specializes: Window

Specializations: **LCD**, **Meter**, **DigiMeter**. **Meter.Shape** has an extra argument *ExtraSpaceFlg*. If T, this will allow you to shape a fairly arbitrary region for the gauge; if NIL, the meter is constrained to be close to a square. This latter behavior is what the user sees when trying to shape the meter from the window menu.

**BarChart**, **HBarChart**, and their subclasses can only be freely **Shaped** in the direction their bars run (i.e., **BarCharts** can be **Shaped** vertically and **HBarCharts** can be **Shaped** horizontally). Their size along the other dimension is fixed by the number of values attached to the chart.

Example: This example reshapes the gauge to a location where the lower left corner is at (10,100) a width of 50 and a height of 150.

```
14←(← ($ lcd1) Shape ' (10 100 50 150))
```

---

**(← self ShapeToHold)****[Method of Gauge]**

Purpose/Behavior: Shapes the gauge window to its smallest possible size based on **width:,min** and **height:,min** and redisplay the gauge.

Returns: NIL

Specializations: **LCD.Shape**

---

**(← self Update)****[Method of Gauge]**

Purpose/Behavior: Reinitializes the gauge and its display window to reflect the current state.

Returns: *self*

Categories: Window

---

**Examples**

The typical use pattern for a gauge is to first create it, set the scale to the appropriate value, and attach it to the desired data.

To attach a horizontal scale to a LOOPS window, **w1**, first enter

```
15←(← ($ Window) New 'w1)
#,$(& HorizontalScale (|OZW0.1Y:.;h.Qm:| . 495))
```

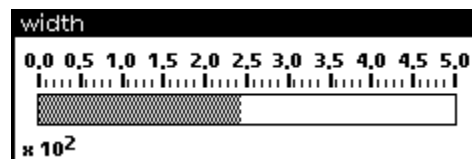
```
16←(← ($ HorizontalScale) New 'hs1)
#,$(& HorizontalScale (|OZW0.1Y:.;h.Qm:| . 496))
```

```
17←(← ($ hs1) SetScale 0 500)
NIL
```

Now make the connection.

```
18←(← ($ hs1) Attach ($ w1) 'width GRAYSHADE)
#,$(& HorizontalScale (|OZW0.1Y:.;h.Qm:| . 496))
```

The following gauge appears and you are prompted to place it .



The title of the gauge shows the instance variable being monitored.

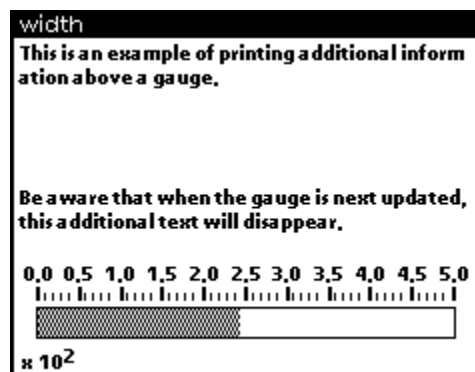
Gauges can be shaped larger. The graphics used to display scales do not change; extra white space is added to the top or right. You can use this space to print additional information, as follows:

```
19←(MOVETOUPPERLEFT (@ ($ hs1) window))
{WINDOW}#372,7104
```

```
20←(PRIN1 "This is an example of printing additional
information above a gauge.
```

Be aware that when the gauge is next updated, this additional text will disappear." (@ (\$ hs1) window))  
 "This is an example of printing additional information above a gauge.

Be aware that when the gauge is next updated, this additional text will disappear."



## Limitations

When a font is changed, a gauge occasionally needs to be updated to be correctly displayed.

Instruments can have only floating point numbers for labels, and cannot have integers.

[This page intentionally left blank]

---

## Description/Introduction

---

Masterscope has been modified to provide for analysis of files created under the Koto or Lyric/Medley release of LOOPS. A full explanation of Masterscope can be found in the *Lisp Library Modules Manual*. In addition to the relations explained there, LOOPS defines the relations described in this chapter.

Note: Masterscope data base files created under Buttriss Loops will not function properly in this release. Those data base files will have to be recreated.

---

## Installation/Loading Instructions

---

- Load MASTERSCOPE from your Lyric/Medley library floppies according to its loading instructions. This should load the compiled files MASTERSCOPE, MSANALYZE, and MSPARSE.
- Load LOOPSMS.DFASL from wherever you installed the LOOPS Library Modules. This should load versions of MASTERSCOPE and MSPARSE that extend Masterscope to handle LOOPS constructs.

---

## Relations

---

LOOPS defines the following relations:

| Name                | Type     | Description                                                                                                    |
|---------------------|----------|----------------------------------------------------------------------------------------------------------------|
| <b>SEND</b>         | Relation | Collects all places where the method is sent.                                                                  |
| <b>SEND SELF</b>    | Relation | Collects all places where the method is sent to <i>self</i> .                                                  |
| <b>SEND NOTSELF</b> | Relation | Collects all places where the method is sent to an object other than <i>self</i> .                             |
| <b>GET</b>          | Relation | Locates all places where the value of an instance variable is retrieved.                                       |
| <b>GET CV</b>       | Relation | Locates all places where the value of a class variable is retrieved.                                           |
| <b>PUT</b>          | Relation | Locates all places where the value of an instance variable is set.                                             |
| <b>PUT CV</b>       | Relation | Locates all places where the value of a class variable is set.                                                 |
| <b>IMPLEMENT</b>    | Relation | Locates all methods that specialize the given selector.                                                        |
| <b>SPECIALIZE</b>   | Relation | Locates all methods that specialize the given selector and use <b>←Super</b> in the body of the method.        |
| <b>OVERRIDE</b>     | Relation | Locates all methods that specialize the given selector and do not use <b>←Super</b> in the body of the method. |

|                   |          |                                                                                                                              |
|-------------------|----------|------------------------------------------------------------------------------------------------------------------------------|
| <b>USE IV</b>     | Relation | Used with an instance variable name to locate all places where the instance variable is used in a <b>GET</b> or <b>PUT</b> . |
| <b>USE CV</b>     | Relation | Used with a class variable name to locate all places where the class variable is used in a <b>GET</b> or <b>PUT</b> .        |
| <b>USE OBJECT</b> | Relation | Used with an object name to locate all places where the object is used.                                                      |

---

**SEND**[Relation]

---

Purpose/Behavior: Used between method names and selectors to collect all places where the method is sent. For example, the form

```
. WHO IS SENT BY 'Helicopter.Move
```

works, but

```
. WHO IS SENT BY Move
```

does not work.

Example: The following command allows you to edit all code that sends the message **New**.

```
. EDIT ALL WHO SEND New
```

---

**SEND SELF**[Relation]

---

Purpose/Behavior: Used between method names and selectors to collect all places where the method is sent to *self*. Places where

```
(← self methodName)
```

is found are collected, while places where

```
(← otherInstance methodName)
```

is found are not.

Example: The following command allows you to edit all code that sends the message **Clear** to *self*.

```
. WHO SENDS SELF Clear
```

---

**SEND NOTSELF**[Relation]

---

Purpose: Same as **SEND SELF**, except the only places where the message is sent to an object other than *self*.

Example: The following allows you to edit all code that sends the message **Clear** to any instance other than *self*.

```
. SHOW ALL WHO SEND NOTSELF Clear
```

---

**GET**[Relation]

---

Purpose: Used with an instance variable name to locate all places where the value of the instance variable is retrieved. This relation can be used along with the **SELF** and **NOTSELF** modifiers.

Example: This command allows you to edit all code that gets the value of the instance variable **width** from an instance other than **self** and the value of the instance variable **height** from **self**.

```
. SHOW ALL WHO GET NOTSELF width AND GET SELF height
```

**GET CV**[\[Relation\]](#)

Purpose: Same as **GET**, except that **GET CV** locates places where the value of the class variable is retrieved. This relation can be used with the **SELF** and **NOTSELF** modifiers.

Example: This command allows you to edit all code that accesses the value of the class variable **height** of **self**.

```
. SHOW ALL WHO GET CVSELF height
```

**PUT**[\[Relation\]](#)

Purpose: Used with an instance variable name to locate all places where the value of the instance variable is set. This relation can be used along with the **SELF** and **NOTSELF** modifiers.

Example: This command allows you to edit all code that sets the value of the instance variable **width**.

```
. EDIT ANY WHO PUT width
```

**PUT CV**[\[Relation\]](#)

Purpose: Same as **PUT**, except locates places where a specified class variable is set. This relation can be used along with the **SELF** and **NOTSELF** modifiers.

Example: This command list all the sections of code that set the value of the class variable **width** for an instance other than **self**.

```
. WHO PUTS CV NOTSELF width
```

**IMPLEMENT**[\[Relation\]](#)

Purpose: Used with a method name to locate all methods that specialize the given selector.

Example: This returns a list of classes where the method **Clear** is defined.

```
. WHO IMPLEMENTS Clear
```

**SPECIALIZE**[\[Relation\]](#)

Purpose: Used with a method name to locate all methods that specialize the given selector and use **←Super** in the body of the method.

Example: This command allows you to edit all the methods that are specializations of **Clear** and use the **←Super** form.

```
. EDIT ANY WHO SPECIALIZE Clear
```

**OVERRIDE**[\[Relation\]](#)

Purpose: Like **SPECIALIZE** above, except it locates all methods that specialize the given selector and **←Super** is not used in the body of the method.

Example: This command allows you to edit all the specializations of **Clear** that do not make use of the  $\leftarrow$ **Super** form.

```
. EDIT ALL WHO OVERRIDE Clear
```

---

**USE IV****[Relation]**

Purpose: Used with an instance variable name to locate all places where the instance variable is used in a **Get** or **Put**. It is equivalent to using the relation form of **GET IVName** or **PUT IVName**.

Example: This command allows you to edit all code that either sets or accesses the instance variable **width**.

```
. EDIT ANY WHO USE THE IV width.
```

---

**USE CV****[Relation]**

Purpose: Used with a class variable name to locate all places where the class variable is used in a **Get** or **Put**. It is equivalent to using the relation form: **GET CV CVName** OR **PUT CV CVName**.

Example: This command allows you to edit all code where the class variable **commonWindow** is either set or accessed.

```
. EDIT ANY WHO USE THE CV commonWindow
```

---

**USE OBJECT****[Relation]**

Purpose Uses an object name to locate all places where the object is used.

Example This command returns a list of all code where the object **Window** is used.

```
. WHO USES THE OBJECT Window??
```

---

**Limitations**

---

Masterscope has several limitations:

- Names of methods must be quoted when used with Masterscope; for example, the method name `Helicopter.Move` must be entered as `'Helicopter.Move'`.
- The following expression will not find a call to **GetValue** when in a method:

```
. WHO CALLS GetValue
```

Masterscope does not record calls to **GetValue** and **PutValue** explicitly; it records them under the **Get**- relation along with calls of the form

```
($\leftarrow$ foo Get 'bar')
```

Similarly, the following functions are recorded under relations instead of their names:



**GetClassValue**  
**PutClassValue**  
**GetClassIV**  
**PutClassIV**

Get CV  
 Put CV  
 Get IV  
 Put IV

If you want to find the explicit calls to Get/PutValue, use

```
. WHO GETS ANY AND NOT SENDS Get
```

- Masterscope currently assumes calls to **GetValue** and similar accessors are accessing instance variables; i.e.,

```
(GetValue foo 'bar)
```

records an access to the instance variable **bar**. This is not necessarily the case; **bar** could also be a class variable.

- The methods and functions that create class and instance variables populate the appropriate **PUT NOTSELF** relations. For example, a function that does

```
(←($ foo) AddCV 'bar)
```

will be found by the query

```
. WHO PUTS CV NOTSELF 'bar
```

An exception occurs with the generalized **Add** and **Delete** method. For example,

```
($ foo) Add 'IV 'bar)
```

will not be noticed as accessing the instance variable **bar**.

Also, the templates for methods and functions that accept property lists generally only notice the first property. For example,

```
((←($ foo) NewWithValues '((bar baz chain link
sausage)))
```

notices **baz** as a property, not a link.

[This page intentionally left blank]

---

## Description/Introduction

---

In many knowledge-based systems, it is useful to represent knowledge as interconnected sets of instances. A virtual copy mechanism allows a network of instances to be viewed as a prototype which can be copied. The copy of the prototype is virtual in that the contents of each instance is not completely copied at creation time. Instead, it inherits default values from the prototype (also called the original), thus continuing to share the parts not modified in the copy. The copied network is virtual also in the sense that only those instances needed in the processing are copied.

A virtual copy of an object in the prototype network has the following properties:

- It responds to at least the same set of messages as the prototype object and in the same way; that is, a copy has the same procedural behavior that is defined for the prototype.
- A copy inherits variables and their values from the prototype, and continues to do so until an explicit change is made in the copy. At that point, the new value is stored in the copy and it stops tracking the prototype for that variable. A fetch operation on a value that is not stored locally either finds or creates a virtual copy of the value obtained from the prototype.

---

## Installation/Loading Instructions

---

The implementation of virtual copies is contained in the file LOOPSVCOPY.LCOM. No other files are necessary.

---

## Application /Module Functionality

---

A network of instances is tied together through the values of instance variables within each of the instances. Assume an object A has an instance variable x, the value of which is the object B. A virtual copy of A will also have an instance variable named x. The value of x in the copy will point to B if B is a shared object, or x may point to a copy of B if it is to be virtual. Changing the value of x in the copy will not change the value in the original.

---

## Overview of Operation

---

By default, virtual copies share instance variables. This means that changing the value of an instance variable in the original will be tracked by the copy.

Virtual copies are implemented with two additional classes:

- **VirtualCopyMixin**

The class **VirtualCopyMixin** is a subclass of Tofu which contains two instance variables:

- % **copyMap**%
- % **copyOf**%

(These unusual names are used to avoid conflicts with any other instance variable names users may create.) This class contains several methods, most of which are required to implement virtual copies and are not used by a programmer.

Printing a virtual copy instance is a specialization of how regular instances are printed. All instances print as #,(\$& <class-name> UID). The class of a virtual copy is a dynamic mixin of the class **VirtualCopyMixin** and the class of the original object (see the *LOOPS Reference Manual* for more information on mixins). The virtual copy print function adds the name or unique identifier (UID) of the original object. For example,

#,(\$& (VirtualCopyMixin Container1) (JFW0.0X:.aF4.R>8 . 3) c1)

is a copy of the object named **c1**.

- **VirtualCopyContext**

The class **VirtualCopyContext** has no methods and only one instance variable, **copyMap**. Instances are used as an argument for calls to **MakeVirtualMixin**.

Since copies can be made of copies, you often need to determine the original object of a chain of copies with the **UltimateOriginal** function.

---

## Operands

This section describes the functions, methods, class variables, and instance variables that operate on virtual copies.

---

### VirtualIVs

[Class Variable]

Purpose/Behavior: Helps specify a class whose instances may be made into virtual copies. The value of this class variable should be either the symbol ALL, or a list of instance variables contained within instances of the class. If the value is ALL, all objects pointed to by any of the instance variables will be copied. If the value is a list of instance variables, only the instance variables on this list will have their values copied. Other instance variable values will be shared between the copy and the original.

---

### (MakeVirtualMixin x copyContextObj)

[Function]

Purpose: Creates a virtual copy of an object.

Behavior: Creates a dynamic mixin class combining the classes **VirtualCopyMixin** and the class of x. An instance of this resulting class is created and it is returned.

Arguments: *x* An object to be copied; must have the class variable **VirtualIVs** as described above.

*copyContextObj*

Usually NIL; used internally by **MakeVirtualMixin** when it calls itself. It can be an instance of **VirtualCopyContext** if you are creating an instance that is intended to be part of a currently existing network of copies starting from another entry point. See description in **Limitations** below for a further explanation of this point.

Returns: An object that is a copy of *x*.

Example: Refer to the section, "Example."

---

**% copyMap%** [Instance Variable of VirtualCopyMixin]

Purpose/Behavior: A mapping of original nodes (which are objects) in a network to the copied nodes. This map is stored in an instance of the class **VirtualCopyContext**.

---

**% copyOf%** [Instance Variable of VirtualCopyMixin]

Purpose/Behavior: Within an instance that is a copy, the value of this instance variable is a pointer to the object that was copied.

---

**(← self VirtualCopy?)** [Method of VirtualCopyMixin]

Purpose: Determines if an object is a virtual copy.

Returns: *self*

Categories: Object, VirtualCopyMixin

---

**copyMap** [Instance Variable of VirtualCopyContext]

Purpose/Behavior: The value of this instance variable is a list of dotted pairs. The CAR of each pair is the original; the CDR, the copy.

---

**(UltimateOriginal self)** [Function]

Purpose: Determines what an object is ultimately copying.

Behavior: If *self* is not a virtual copy, *self* is returned.

If *self* is a virtual copy, this recurses through the value of the instance variable **% copyOf%** until it finds the original and returns it.

Arguments: *self* A LOOPS object.

Returns: *self* or what is at the top of *self*'s copy chain.

## Example

Create a class called **test** and edit it as shown.

```
44←(←($ Class) New 'test)
#,($C test)
```

```
45←(ED 'test)
```

```
SEdit #,($C test) Package: INTERLISP
((MetaClass Class Edited%: ; Edited 11-Dec-87
 ; 16:50 by
)
 (Supers Object)
 (ClassVariables (VirtualIVs (atomCopy listCopy objCopy)))
 (InstanceVariables (atom NIL)
 (atomCopy NIL)
 (list NIL)
 (listCopy NIL)
 (obj NIL)
 (objCopy NIL))
 (MethodFns))
```

Create an instance called **t0** of this class and inspect it.

```
46←(←($ test) New 't0)
#,($& test (N↑W0.1Y%.:;h.Lh9 . 556))
```

```
47←(← ($ test)
 NewWithValues
 (BQUOTE ((atom 1)
 (atomCopy 2)
 (list (a b c))
 (listCopy (A B (\, (← ($ test) New (QUOTE t1))))
 (obj (\, (← ($ test) New (QUOTE t2))))
 (objCopy (\, (← ($ test) New (QUOTE t3)))))))
#,($& test (N↑W0.1Y%.:;h.Lh9 . 560))
```

```
48←(← IT SetName 't0)
#,($& test (N↑W0.1Y%.:;h.Lh9 . 560))
```

```
49←(INSPECT IT]
{WINDOW}#52,51234
```

```
All Values of test ($ t0).
atom 1
atomCopy 2
list (a b c)
listCopy (A B #,($ t1))
obj #,($ t2)
objCopy #,($ t3)
```

Make a copy called **t0copy** and inspect it.

```
57←(← (MakeVirtualMixin ($ t0))
 SetName
 (QUOTE t0copy))
#,($& (VirtualCopyMixin test) N↑W0.1Y%.:;h.Lh9 . 562)
```

```
58← (INSPECT IT)
{WINDOW}#53,10150
```

```
All Values of (VirtualCopyMixin test) ($ t0copy).
atom NIL
atomCopy NIL
list NIL
listCopy NIL
obj NIL
objCopy NIL
| copyOf | #,($ t0)
| copyMap | #,($& VirtualCopyContext (N+V0.1Y%.:;h.Lh9 . 563))
```

Make the following changes to **t0** and then reinspect **t0copy**.

```
60←(for iv in '(atom atomCopy list listCopy obj objCopy)
as val in (LIST 11 22 '(a b c d) '(A B C) ($ t3) ($ t1))
do (PutValue ($ t0) iv val]
NIL
```

```
61← (INSPECT IT)
{WINDOW}#53,10152
```

```
All Values of (VirtualCopyMixin test) ($ t0copy).
atom 11
atomCopy 22
list (a b c d)
listCopy (A B C)
obj #,($ t3)
objCopy #,($& (VirtualCopyMixin test) (N+V0.1Y%.:;h.Lh9 . 565) t1)
| copyOf | #,($ t0)
| copyMap | #,($& VirtualCopyContext (N+V0.1Y%.:;h.Lh9 . 566))
```

The copied instance variables have not changed since they do not track changes in the original object.

## Limitations

Some subtle issues are involved in building and using prototype structures so that the structure is preserved in the copied network. These involve how the network is typically traversed.

A general constraint is that all the links to any shared node in the prototype either all be marked as virtual variables, or none of them are. If they are all marked, then a single copy will be made and used. If none are, then the original object from the prototype will be used. Sharing with the prototype can be useful if this object is a repository for standard information that is independent of context. However, if this constraint is violated, the topology of the virtual copy will be different from that of the prototype.

In the simplest situation the network has a single entry node. In this case, a copy-map (see the section "Operands") can be created when the entry node object is first copied. After that all values are copied using this copy-map. The mechanism works well in this situation, even if there is sharing and there are cycles within the network.

At the other extreme, networks can have arbitrary connectivity, including multiple entries from outside the network, for example, from other networks or

non-objects. In this case, the following constraints are necessary to ensure correctness of the virtual copy mechanism.

The first constraint states that all access to the network must start through a copy of one of the nodes in the prototype. This condition is necessary because the criteria for copying are contained in the links from one object to another, not in the objects themselves, and a shared node could not specify a link to a node to be copied. This constraint ensures that all accesses from the outside will be copied if and only if that object would have been copied because of an internal link. Otherwise, an analogous situation would occur in which you could either reach a copy or the original node of the prototype itself depending upon which path you follow when the paths lead to the same node in the prototype.

The final constraint requires that all entries to the network should be passed the same copy-map if they are to share structure. The underlying concern in imposing these constraints is that a network be always copied the same way to maintain its topology regardless of where you start.

Suppose you want to make a virtual copy of a virtual copy, that is, to use a virtual copy of a network as a prototype itself. This is very useful if you are using a network to hold the state of a partial design and you want to try two alternative continuations of the design. Some hidden costs are associated with such multiple-level virtual copies.

Suppose further that a network N1 is used as a prototype and you make a virtual copy, N1-VC. Furthermore, N1-VC-VC is defined to be a copy of N1-VC. Values missing from N1-VC-VC are found in the corresponding object of N1-VC. If the value is missing there, the process recurs, and N1 is examined. If the value is to be a virtual copy, then this process will add a virtual copy in N1-VC, and then a second level copy in N1-VC-VC. This is necessary to preserve the semantics presented, but implies that many levels of virtual copy cannot easily do inexpensive incremental searches of a network.

---

## References

---

Mittal, S. , Bobrow, D. G., and Kahn, K. *Virtual Copies, Between Classes and Instances*. ACM OOPSLA-86 Conference Proceedings, Portland, Oregon, 1986.



## A

**ActiveGaugeMixin** (Class) 4  
**ActiveHorizontalScale** (Class) 5  
**ActiveVerticalScale** (Class) 5  
**AlarmMixin** (Class) 6  
**AlphaNumeric** (Class) 6  
**Attach** (Method of Gauge) 16  
**Attached?** (Method of Gauge) 16

## B

**BoundedMixin** (Class) 7

## C

**ChangeFont** (Method of Gauge) 17  
**Close** (Method of Gauge) 17  
**copyMap** (Instance Variable of VirtualCopyContext) 29

## D

**Destroy** (Method of Gauge) 17  
**Detach** (Method of Gauge) 17  
**Dial** (Class) 8  
**DigiMeter** (Class) 8  
**DigiScale** (Class) 9

## G

**Gauge** (Class) 9  
gauge classes 3  
gauge methods 15  
**GaugeAV** (Class) 9  
**GaugeClasses** (Variable) 3  
**GAUGEFILES** (Variable) 3  
gauges 1  
**GET** (Relation) 23  
**GET CV** (Relation) 23

## H

**HorizontalScale** (Class) 10  
**HSGraphics** (Class) 11

## I

**IMPLEMENT** (Relation) 23  
**Instrument** (Class) 11

## L

**LCD** (Class) 11  
**LCDMixin** (Class) 12  
**LOADGAUGES** (Function) 2

## M

**MakeVirtualMixin** (Function) 28  
Masterscope 21  
**Meter** (Class) 12

## O

**OVERRIDE** (Relation) 24

## P

**PUT** (Relation) 23  
**PUT CV** (Relation) 23

## R

**Reset** (Method of Gauge) 17  
**RoundScale** (Class) 12

## S

**SelfScaleMixin** (Class) 13  
**SEND** (Relation) 22  
**SEND NOTSELF** (Relation) 22  
**SEND SELF** (Relation) 22  
**SetScale** (Method of Gauge) 17  
**Shape** (Method of Gauge) 18  
**ShapeToHold** (Method of Gauge) 18  
**SPECIALIZE** (Relation) 23  
SSHHorizontalScale 14  
SSVerticalScale 14  
**StraightScale** (Class) 14

## U

**UltimateOriginal** (Function) 29  
**Update** (Method of Gauge) 18  
**USE CV** (Relation) 24  
**USE IV** (Relation) 24  
**USE OBJECT** (Relation) 24

## V

**VerticalScale** (Class) 14  
virtual copies 27  
**VirtualCopy?** (Method of VirtualCopyMixin) 29  
**VirtualIVs** (Class Variable) 28  
**VSGraphics** (Class) 15

## %

**% copyMap%** (Instance Variable of VirtualCopyMixin) 29  
**% copyOf%** (Instance Variable of VirtualCopyMixin) 29

ntionally left blank]

[This page inte

# **XEROX LOOPS LIBRARY MODULES MANUAL**

**XEROX**

XEROX LOOPS LIBRARY MODULES MANUAL

Lyric /Medley Release

July 1988

Copyright © 1988 by Xerox Corporation.

Xerox LOOPS is a trademark.

All rights reserved.

# TABLE OF CONTENTS

|                                   |    |
|-----------------------------------|----|
| PREFACE                           | v  |
| GAUGES                            | 1  |
| Description/Introduction          | 1  |
| Prerequisites                     | 2  |
| Installation/Loading Instructions | 2  |
| Application/Module Functionality  | 3  |
| Gauge Classes                     | 3  |
| Gauge Methods                     | 15 |
| Examples                          | 18 |
| Limitations                       | 19 |
| MASTERSCOPE                       | 21 |
| Description/Introduction          | 21 |
| Installation/Loading Instructions | 21 |
| Relations                         | 21 |
| Limitations                       | 24 |
| VIRTUAL COPIES                    | 27 |
| Description/Introduction          | 27 |
| Installation/Loading Instructions | 27 |
| Application/Module Functionality  | 27 |
| Overview of Operation             | 27 |
| Operands                          | 28 |

## TABLE OF CONTENTS

---

|             |    |
|-------------|----|
| Example     | 30 |
| Limitations | 31 |
| References  | 32 |

---

[This page intentionally left blank]

---

## Overview of the Manual

---

The *Xerox LOOPS Library Modules Manual* describes the Library Modules for Xerox's Lisp Object-Oriented Programming System, Xerox LOOPS (TM). These Library Modules, which can be loaded into your Xerox Artificial Intelligence Environment, provide additional functionality to Xerox LOOPS.

This manual describes the Lyric/Medley Release of the Xerox LOOPS Library Modules, which run under the Lyric and Medley Releases of Xerox Lisp.

---

## Organization of the Manual and How to Use It

---

This manual is divided into chapters, with each chapter focussing on a particular Library Module. A Table of Contents is included to help you find specific material.

---

## Conventions

---

This manual uses the following conventions:

- Case is significant in Xerox LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown. Typically, this means that method names are capitalized and variables are not.
- You need to use an Interlisp Exec to enter all exec expressions.
- Arguments appear in italic type.
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a method appears as follows:

```
(_ self Selector Arg1 Arg2)
```

- Examples are shown in the Interlisp Exec and appear in the following typeface:

```
89_ (_ LOGIN)
```

- All examples are typed into an Interlisp Exec. This is the recommended Exec for all Xerox LOOPS expressions.
- Methods with an exclamation mark (!) suffix usually perform operations deeply into class structure instead of only on a given object.
- Methods with a question mark (?) suffix usually are predicates; that is, truth functions.
- Methods often appear in the form **ClassName.SelectorName**.



- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

This manual describes the Xerox LOOPS items (functions, methods, etc.) by using the following template:

|                  |                                                                                                                                                                                                                                                                                       |                 |             |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|-------------|
| Purpose:         | Gives a short statement of what the item does.                                                                                                                                                                                                                                        |                 |             |
| Behavior:        | Provides the details of how the item operates.                                                                                                                                                                                                                                        |                 |             |
| Arguments:       | Describes each argument in the following format:<br><br><table> <tr> <td><i>argument</i></td><td>Description</td></tr> </table>                                                                                                                                                       | <i>argument</i> | Description |
| <i>argument</i>  | Description                                                                                                                                                                                                                                                                           |                 |             |
| Returns:         | States what the item returns, and does not appear if the item does not return a value. The phrase "Used as a side effect only." means that the purpose of the item is to perform a computation or action that is independent of any returned value, not to return a particular value. |                 |             |
| Categories:      | A way to group related methods. For example, all the methods related to Masterscope on the class <b>FileBrowser</b> have the category Masterscope, not <b>FileBrowser</b> . This item appears only for methods.                                                                       |                 |             |
| Specializes:     | The next higher class in the class hierarchy that contains a method with the same selector. For example, <b>RectangularWindow.Open</b> can specialize <b>Window.Open</b> . This appears only for methods.                                                                             |                 |             |
| Specializations: | The next lower class in the class hierarchy that contains a method with the same selector. For example, <b>Window.Open</b> is a specialization of <b>RectangularWindow.Open</b> . This appears only for methods.                                                                      |                 |             |
| Example:         | An example is often included to show how to use the item and what result it produces. Some examples may appear differently on your system, depending on the settings of various print flags. See the <i>Xerox LOOPS Reference Manual</i> for details.                                 |                 |             |

---

## References

---

The following books and manuals augment this manual.

*Xerox LOOPS Reference Manual*

*Xerox LOOPS Release Notes*

*Xerox LOOPS Users' Modules Manual*

*Interlisp-D Reference Manual*

*Common Lisp: the Language* by Guy Steele

*Xerox Common Lisp Implementation Notes, Lyric Release*

*Xerox Lisp Release Notes, Lyric and Medley Releases*

*Xerox Lisp Library Modules Manual, Lyric and Medley Releases*

# Writer's Notes -- Conventions

---

This file includes notes on conventions for *Xerox LOOPS Library Modules Manual*, Lyric Beta Release. This manual is packaged with the *Xerox LOOPS Release Notes* and *Xerox LOOPS Reference Manual* to form one binder.

Writer: Raven Kontur Brewster

Printing Date: >>DA<< >>MO<< 1988

## Directories and Files

---

The directory {ERIS}<Doc>Loops>Lyric>Beta>LibMods> contains the files for the manual. This directory has the following subdirectories:

- {ERIS}<Doc>Loops>Lyric>Beta>LibMods>Z-ReleaseInfo> contains this file on writing conventions and a file on production details.

Filenames describe the contents of the file. For example, the filename

{ERIS}<Doc>Loops>Lyric>Beta>LibMods>Gauges

contains the chapter on gauges.

Assemble the files in the following order for the manual:

```
{ERIS}<Doc>Loops>Lyric>Beta>LibMods>A1-TitlePage.tedit
{ERIS}<Doc>Loops>Lyric>Beta>LibMods>A2-TOC.tedit
{ERIS}<Doc>Loops>Lyric>Beta>LibMods>A3-Preface.tedit
{ERIS}<Doc>Loops>Lyric>Beta>LibMods>Gauges.tedit
{ERIS}<Doc>Loops>Lyric>Beta>LibMods>Masterscope.tedit
{ERIS}<Doc>Loops>Lyric>Beta>LibMods>VC.tedit
```

## Conventions

---

This manual uses the following conventions:

- Case is significant in Xerox LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown. Typically, this means that method names are capitalized and variables are not.
- Arguments appear in italic type.
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a method appears as follows:

(\_ self **Selector** *Arg1 Arg2*)

- Examples appear in the following typeface:

89\_(\_LOGIN)

- All examples are typed into an Interlisp Exec. This is the recommended Exec for all Xerox LOOPS expressions.

- Methods with an exclamation mark (!) suffix usually perform operations deeply into class structure instead of only on a given object.
- Methods with a question mark (?) suffix usually are predicates; that is, truth functions.
- Methods often appear in the form **ClassName.SelectorName**.
- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

This manual describes the Xerox LOOPS items (functions, methods, etc.) by using the following template:

|                  |                                                                                                                                                                                                                                                                                       |                 |             |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|-------------|
| Purpose:         | Gives a short statement of what the item does.                                                                                                                                                                                                                                        |                 |             |
| Behavior:        | Provides the details of how the item operates.                                                                                                                                                                                                                                        |                 |             |
| Arguments:       | Describes each argument in the following format:<br><br><table><tr><td><i>argument</i></td><td>Description</td></tr></table>                                                                                                                                                          | <i>argument</i> | Description |
| <i>argument</i>  | Description                                                                                                                                                                                                                                                                           |                 |             |
| Returns:         | States what the item returns, and does not appear if the item does not return a value. The phrase "Used as a side effect only." means that the purpose of the item is to perform a computation or action that is independent of any returned value, not to return a particular value. |                 |             |
| Categories:      | A way to group related methods. For example, all the methods related to Masterscope on the class <b>FileBrowser</b> have the category Masterscope, not <b>FileBrowser</b> . This item appears only for methods.                                                                       |                 |             |
| Specializes:     | The next higher class in the class hierarchy that contains a method with the same selector. For example, <b>RectangularWindow.Open</b> can specialize <b>Window.Open</b> . This appears only for methods.                                                                             |                 |             |
| Specializations: | The next lower class in the class hierarchy that contains a method with the same selector. For example, <b>Window.Open</b> is a specialization of <b>RectangularWindow.Open</b> . This appears only for methods.                                                                      |                 |             |
| Example:         | An example is often included to show how to use the item and what result it produces. Some examples may appear differently on your system, depending on the settings of various print flags.                                                                                          |                 |             |

## Style Sheet Addenda

---

Here are some guidelines I used when writing the LOOPS manuals. Items appear in rather random order.

- Avoid contractions.
- Avoid subscripts. Use WORD1 rather than WORD to avoid inconsistent line leading.
- Avoid wording that starts "Note that..." or "Notice that...". Either make it a note with correct format or eliminate the "Note that".
- Use semicolons rather than m-dashes.
- Each item in the template starts with an initial capital letter; e.g., "Describes..."
- The arguments are identical in the call and in the argument description.

- Parentheses appear around expressions and square brackets appear around the name of the functionality.
- The arrow in the expression is the NS character  $\leftarrow$ , not `_`. These characters appear similarly when printed, but differently on the screen. See the section, "Special Notes and Cautions," for details
- A period appears after the word None, after argument descriptions, and Returns: item.
- Items are set to or return T (instead of true).
- Menus contain options, not items or selections.
- You drag (not roll) the mouse to the right of a menu option to see its submenu.
- Use "above" and "below" when referring to things in the same section, section numbers and names when referring to things in the same chapter, and chapter numbers and names when referring to things in another chapter.
- Please study the following style sheet carefully before you start to edit. The various appearances of active value and annotated values are especially crazy making.

These things appear in **bold**:

class variables  
functions  
instance variables  
messages  
methods  
variables

**ActiveValue** - specific class/instance

active value - general information

activeValue - previous implementation of **ActiveValue**

annotatedValue - data type

**AnnotatedValue** - specific class

annotated values - general information

bitmap

data type

file package

filecoms

inspector

Lisp Library package

**localState** - instance variable

non-NIL

prettyprints

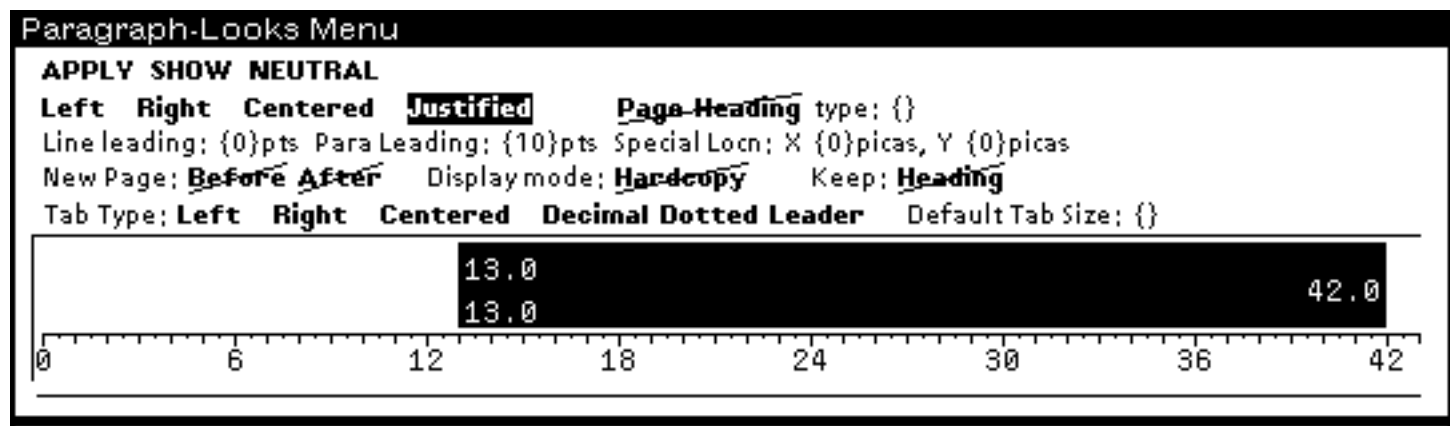
supers list

- Figures

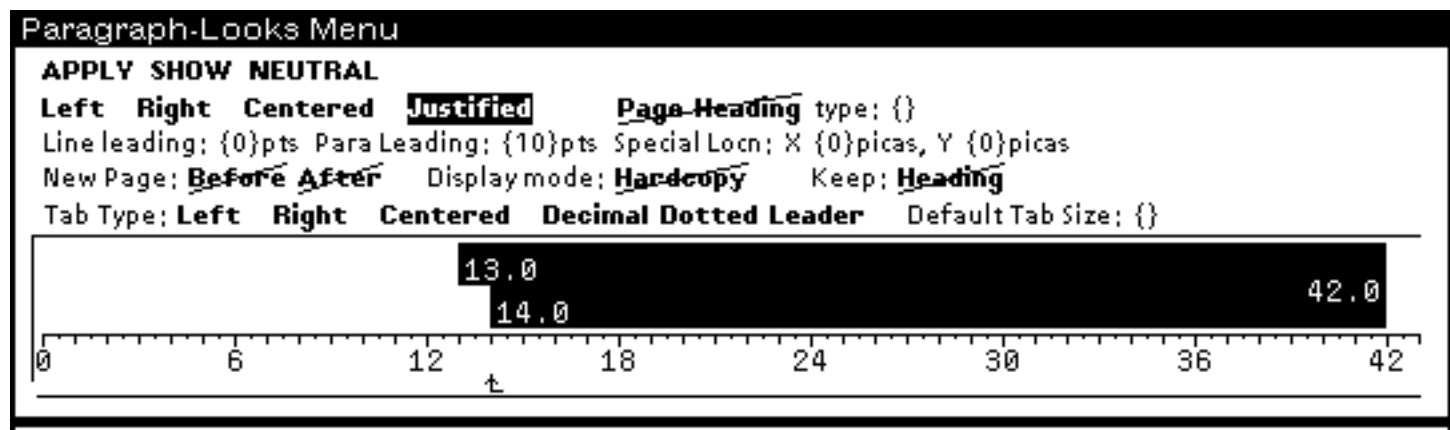
---

## Paragraph Formatting

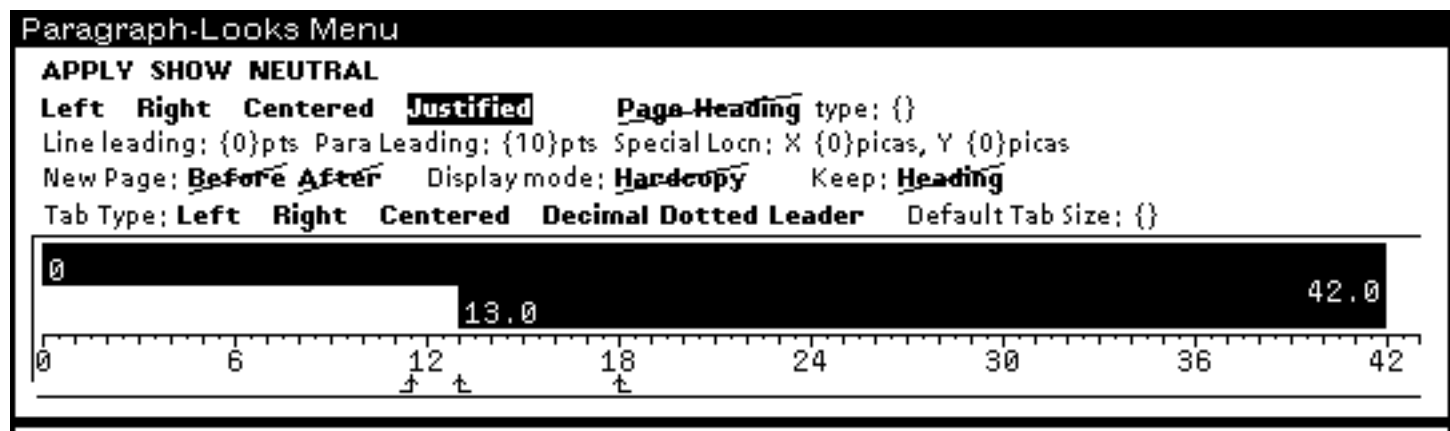
The text has the following format:



Bulleted lists have the following format:



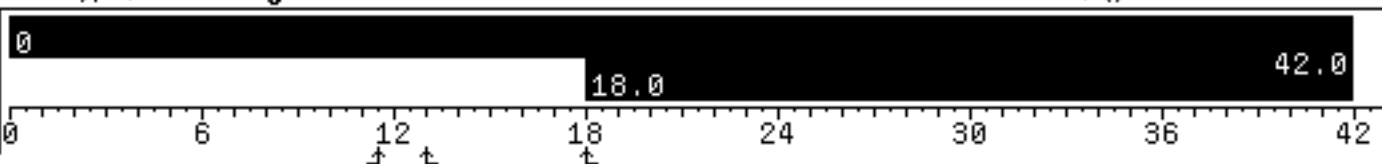
The template has the following format:



The Arguments section of the template has the second line start at 18 instead of 13.

**Paragraph-Looks Menu****APPLY SHOW NEUTRAL****Left Right Centered Justified Page-Heading** type: {}

Line leading: {}pts Para Leading: {}pts Special Locn: X {}picas, Y {}picas

New Page: **Before** ~~After~~ Display mode: **Hardcopy** Keep: **Heading**Tab Type: **Left Right Centered Decimal Dotted Leader** Default Tab Size: {}**Page Layout**

Page numbering varies with the chapter.

**Page Layout Menu****APPLY SHOW**For page: **First(&Default)** **Other Left** **Other Right**Starting Page #: {} Paper Size: **Letter** **Legal** **A4 Landscape**Page numbers: **No** **Yes** X: {} Y: {} Format: **123** **xiv** **XIV**Alignment: **Left** **Centered** **Right**

Text before number: {} Text after number: {}

Margins: Left {} Right {} Top {} Bottom {}

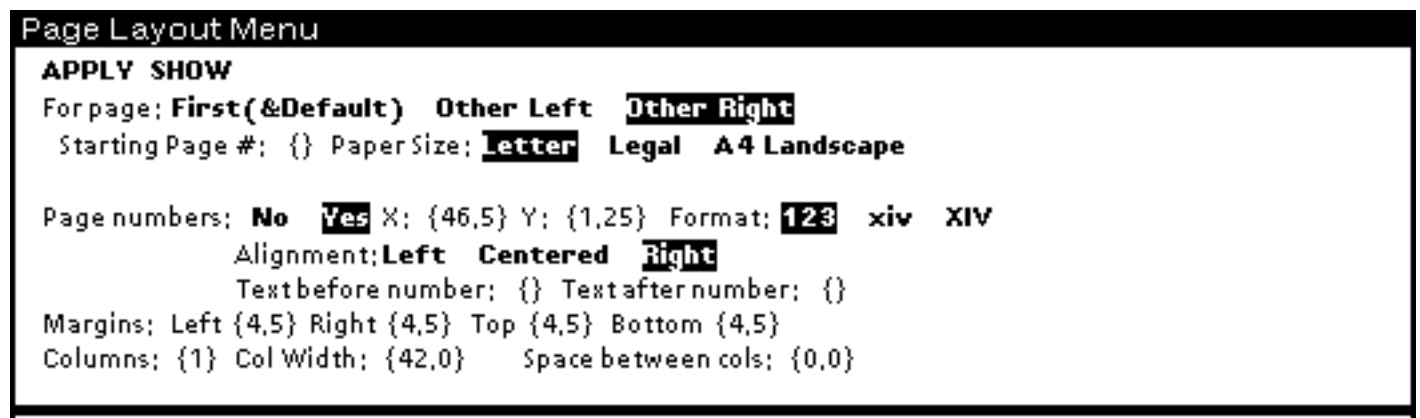
Columns: {} Col Width: {} Space between cols: {}

**Page Layout Menu****APPLY SHOW**For page: **First(&Default)** **Other Left** **Other Right**Starting Page #: {} Paper Size: **Letter** **Legal** **A4 Landscape**Page numbers: **No** **Yes** X: {} Y: {} Format: **123** **xiv** **XIV**Alignment: **Left** **Centered** **Right**

Text before number: {} Text after number: {}

Margins: Left {} Right {} Top {} Bottom {}

Columns: {} Col Width: {} Space between cols: {}



## Bitmaps, Graphs, and Sketches

---

To do SEdit and Inspector examples for the manual, you need to reset your FONTPROFILE and scale the resulting windows to 0.8.

- In your Interlisp Executive, enter (DV FONTPROFILE)
- Edit the FONTPROFILE to be as follows. (some of this is probably overkill, but it does eliminate any suprizes)

```
SEdit FONTPROFILE Package: INTERLISP
```

```
((DEFAULTFONT 1 (GACHA 12 BRR)
 (GACHA 10)
 (TERMINAL 10))
 (ITALICFONT 1 (HELVETICA 12 MIR)
 (GACHA 10 MIR)
 (MODERN 10 MIR))
 (BOLDFONT 2 (HELVETICA 12 BRR)
 (HELVETICA 10 BRR)
 (MODERN 10 BRR))
 (LITTLEFONT 3 (HELVETICA 10)
 (HELVETICA 8 MIR)
 (MODERN 10 MIR))
 (TINYFONT 6 (GACHA 10)
 (GACHA 8)
 (TERMINAL 8))
 (BIGFONT 4 (HELVETICA 14 BRR) NIL (MODERN 12 BRR))
 (MENUFONT 5 (HELVETICA 12))
 (COMMENTFONT 6 (HELVETICA 12)
 (HELVETICA 10)
 (MODERN 10))
 (TEXTFONT 7 (TIMESROMAN 12) NIL (CLASSIC 12)))
```

-- In your Interlisp Executive, enter  
(FONTPROFILE FONTPROFILE)  
(SEdit.RESET)

-- Make bitmaps of the resulting windows, and scale these bitmaps to 0.8

To get the pop-up menus (and their drag-through submenus) into a bitmap for using as an illustration:

--Move your type-in point to the exec window.

--Bring up your pop-up menu.

--control-G

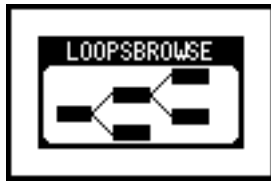
--When the menu comes up, select Mouse \*run. This will cause a break and spawn a new mouse process, so that the mouse continues to work.

--Move your type-in point to the tedit window.

--Shift-snap the menu image into the tedit window.

--Move the type-in point to the break window and type ^.





--To have your browser menus be a different font, type in your Interlisp Exec

```
(SETQ MENUFONT (FONTCREATE 'TERMNIAL 12 'BOLD))
```

--The global variable MENUFONT which is currently set to (HELVETICA 10 MRR).

--The menus that have already been created will still display this old font. Either recreate the browsers to force the menus to be recreated, or send a message to the browser

```
(_ browser ClearMenuCache)
```

to force that browser to recreate its menus.

## Special Notes and Cautions

---

Make sure you have changed the underscore to be a left arrow before loading and printing any files. To do this,

- Enter the following commands into your Executive:

```
(GETCHARBITMAP (CHARCODE _) '(MODERN 10 MRR))
(EDITBM IT)
```

- When the bitmap editor appears, delete the underscore and insert the following left arrow:

```
.....
.....
.....
.....
.....
...X....
...XX...
..XXXXXX.
...XX...
...X....
.....
.....
.....
.....
```

- Finally, enter the following commands into your Executive to store the pattern:

```
(PUTCHARBITMAP (CHARCODE _) '(MODERN 10 MRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(MODERN 10 BRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(TERMINAL 10 MRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(TERMINAL 10 BRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(TERMINAL 12 BRR) IT)
```

## A

**ActiveGaugeMixin** (Class) 4  
**ActiveHorizontalScale** (Class) 5  
**ActiveVerticalScale** (Class) 5  
**AlarmMixin** (Class) 6  
**AlphaNumeric** (Class) 6  
**Attach** (Method of Gauge) 16  
**Attached?** (Method of Gauge) 16

## B

**BoundedMixin** (Class) 7

## C

**ChangeFont** (Method of Gauge) 17  
**Close** (Method of Gauge) 17  
**copyMap** (Instance Variable of VirtualCopyContext) 29

## D

**Destroy** (Method of Gauge) 17  
**Detach** (Method of Gauge) 17  
**Dial** (Class) 8  
**DigiMeter** (Class) 8  
**DigiScale** (Class) 9

## G

**Gauge** (Class) 9  
gauge classes 3  
gauge methods 15  
**GaugeAV** (Class) 9  
**GaugeClasses** (Variable) 3  
**GAUGEFILES** (Variable) 3  
gauges 1  
**GET** (Relation) 23  
**GET CV** (Relation) 23

## H

**HorizontalScale** (Class) 10  
**HSGraphics** (Class) 11

## I

**IMPLEMENT** (Relation) 23  
**Instrument** (Class) 11

## L

**LCD** (Class) 11  
**LCDMixin** (Class) 12  
**LOADGAUGES** (Function) 2

## M

**MakeVirtualMixin** (Function) 28  
Masterscope 21  
**Meter** (Class) 12

## O

**OVERRIDE** (Relation) 24

## P

**PUT** (Relation) 23  
**PUT CV** (Relation) 23

## R

**Reset** (Method of Gauge) 17  
**RoundScale** (Class) 12

## S

**SelfScaleMixin** (Class) 13  
**SEND** (Relation) 22  
**SEND NOTSELF** (Relation) 22  
**SEND SELF** (Relation) 22  
**SetScale** (Method of Gauge) 17  
**Shape** (Method of Gauge) 18  
**ShapeToHold** (Method of Gauge) 18  
**SPECIALIZE** (Relation) 23  
SSHHorizontalScale 14  
SSVerticalScale 14  
**StraightScale** (Class) 14

## U

**UltimateOriginal** (Function) 29  
**Update** (Method of Gauge) 18  
**USE CV** (Relation) 24  
**USE IV** (Relation) 24  
**USE OBJECT** (Relation) 24

## V

**VerticalScale** (Class) 14  
virtual copies 27  
**VirtualCopy?** (Method of VirtualCopyMixin) 29  
**VirtualIVs** (Class Variable) 28  
**VSGraphics** (Class) 15

## %

**% copyMap%** (Instance Variable of VirtualCopyMixin) 29  
**% copyOf%** (Instance Variable of VirtualCopyMixin) 29

ntionally left blank]

[This page inte

---

## Description/Introduction

---

Gauges are an important part of the LOOPS user interface for both developers and end users. Gauges assist in understanding the dynamic nature of the programs. This is in contrast to the more typical case of debugging programs using static means. In the creation of user-friendly interfaces, you can use gauges to display, in analog or digital form, various data that may be changing. Also, by employing active gauges, you can provide a convenient way to interact with a system.

One of the features of gauges is the ease with which you can use them in a system. In more traditional languages, if you want to understand how a variable is changing over the course of a computation, you must make modifications in your program wherever you want to begin or end the examination of a variable. Given the capabilities of active values used by gauges, you need only attach or detach a gauge to the data that you are interested in monitoring.

The following types of gauges are available:

- Meter; a circular instrument that wraps around any number of times.
- Dial; a bounded dial, like an automobile speedometer.
- LCD; a gauge that uses the entire window to display a value.
- Scale; a horizontal or vertical display of a gauge.
- ActiveScale; a scale that allows you to change the gauge value.

Gauges are an example of the combination of programming capabilities within LOOPS. The different types of gauges are defined within the context of an inheritance lattice. This allows the more general functionality and variables to be allocated to more general gauge classes, with specific functionality placed in more restricted classes. You can also see the use of mixins to add a small amount of functionality to several different classes of gauges.

Note: Mixins are classes that are used only in conjunction with another class to create a subclass.

The methods within gauges are built upon both function calling and message sending. Gauges are "attached" to objects through the mechanism of active values. Since gauges are built upon the mechanism of active values, gauges can only be attached to data within objects. It is not possible to use gauges to monitor any arbitrary Lisp variable.

Prerequisites

The default font for gauges is Modern 10.

Installation/Loading Instructions

Gauges are divided among several different files to allow you to load only those objects and functions that you need. The table below lists the files to load for each type of gauge. The filecoms for each file will try to load any other required gauge files from **LOOPSLIBRARYDIRECTORY**. The file **GAUGES.DFASL** and either **GAUGEINSTRUMENTS.DFASL** or **GAUGEALPHANUMERICS.DFASL** will always be loaded; other files may also be loaded.

| Gauge        | File to load             |
|--------------|--------------------------|
| LCD          | GAUGEALPHANUMERICS.DFASL |
| METER        | GAUGEMETERS.DFASL        |
| DIAL         | GAUGEDIALS.DFASL         |
| SCALE        | GAUGESCALES.DFASL        |
| ACTIVE SCALE | GAUGEACTIVE.DFASL        |

Additionally, the file **GAUGESELFSCALEMIXIN.DFASL** can be loaded to add the class **SelfScaleMixin**, and **GAUGEALARMS.DFASL** can be loaded to add the class **AlarmMixin**.

To load the required files, first set the value of **LOOPSDIRECTORY** to include the directory where the gauges files are stored, then type the following expression in the Executive:

(LOAD 'FILENAME)

To load all of the gauges, load the file **GAUGELOADER** and then enter (LOADGAUGES). **GAUGELOADER** also sets the variables: **GAUGEFILES** and **GaugeClasses**.

(LOADGAUGES LDFLG SOURCES?FLG) [Function]

|            |                                                                                                                                                                                                                                                                                                         |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Loads all the gauges.                                                                                                                                                                                                                                                                                   |
| Behavior:  | Assumes that all of the gauge files are on the <b>LOOPSDIRECTORY</b> search path.<br><br>All the gauge files will be loaded based upon the settings of <i>LDFLG</i> and <i>SOURCES?FLG</i> . A <b>FILESLOAD</b> expression is built up and evaluated.                                                   |
| Arguments: | <i>LDFLG</i> Can be NIL, PROP, or SYSLOAD. See the <b>LDFLG</b> discussion under loading in the <i>Interlisp-D Reference Manual</i> .<br><br><i>SOURCES?FLG</i> Can be NIL or T. If NIL, this attempts to load the compiled files before trying to load the sources. If T, only the sources are loaded. |
| Returns:   | Used for side effect only.                                                                                                                                                                                                                                                                              |

**GAUGEFILES**

[Variable]

Behavior: Initialized to (GAUGEACTIVE GAUGEALARMS GAUGEALPHANUMERICS GAUGEBOUNDEDMIXIN GAUGEDIALS GAUGEDIGIMETER GAUGEDIGISCALE GAUGEINSTRUMENTS GAUGEMETERS GAUGES GAUGESCALES GAUGESELFSCALEMIXIN)

**GaugeClasses**

[Variable]

Behavior: Initialized to (GaugeAV ActiveGaugeMixin Gauge AlarmMixin BoundedMixin SelfScaleMixin)

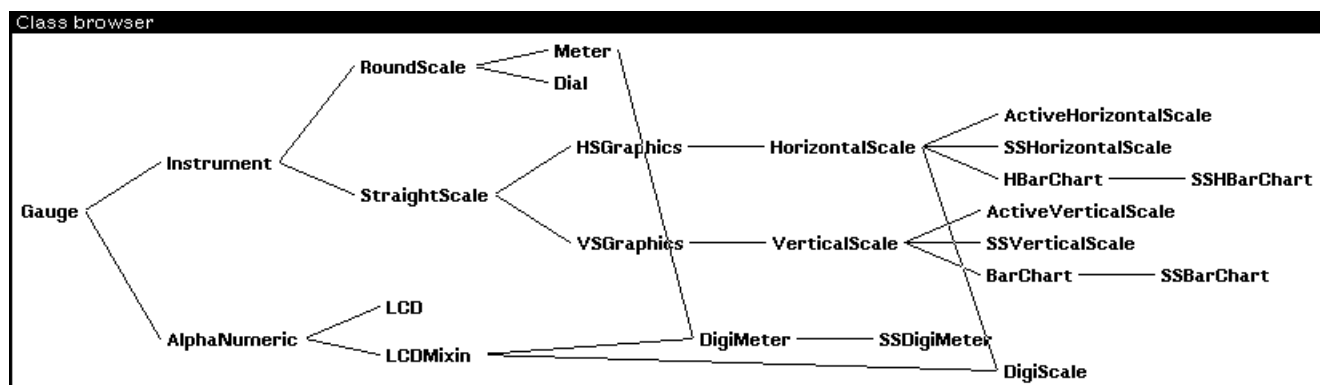
Call (Browse **GaugeClasses**) to open a browser of all of the gauge classes.

**Application/Module Functionality**

This section describes the gauge classes and methods.

**Gauge Classes**

This section describes the available gauges shown in the following browser.



Note: The browser does not include the optional mixin classes.

Within the class description of each class, the instance variables and class variables that are specializations only because they have different default values are not listed.

| Name                         | Type          | Description                                                                                                     |
|------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------|
| <b>ActiveGaugeMixin</b>      | AbstractClass | A gauge class that allows you to set the value of the variable being monitored with the cursor, via a SET menu. |
| <b>ActiveHorizontalScale</b> | Class         | An active gauge that displays the value on a horizontal scale.                                                  |
| <b>ActiveVerticalScale</b>   | Class         | An active gauge that displays its value on a vertical scale.                                                    |
| <b>AlarmMixin</b>            | AbstractClass | A mixin that adds alarm functionality to any gauge.                                                             |
| <b>AlphaNumeric</b>          | AbstractClass | A gauge that gives an alphanumeric display of a value.                                                          |

|                          |               |                                                                                                                                               |
|--------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>BarChart</b>          | Class         | A gauge that displays more than one <b>VerticalScale</b> side-by side.                                                                        |
| <b>BoundedMixin</b>      | AbstractClass | Creates a bounded scale for <b>displayVal</b> ; to be used as a mixin for instruments.                                                        |
| <b>Dial</b>              | Class         | A bounded dial, like an automobile speedometer.                                                                                               |
| <b>DigiMeter</b>         | Class         | A gauge that displays both an <b>LCD</b> and a meter.                                                                                         |
| <b>DigiScale</b>         | Class         | A gauge that displays both an <b>LCD</b> and a horizontal scale.                                                                              |
| <b>Gauge</b>             | AbstractClass | A class for objects that present a dynamic graphical image of a LOOPS value.                                                                  |
| <b>GaugeAV</b>           | Class         | An active value associated with a gauge.                                                                                                      |
| <b>HBarChart</b>         | Class         | A gauge that displays more than one <b>HorizontalScale</b> side-by side.                                                                      |
| <b>HorizontalScale</b>   | Class         | A labeled, bounded scale with a bar that fills to the right.                                                                                  |
| <b>HSGraphics</b>        | AbstractClass | Gauge that is displayed in the form of a single horizontal scale or bar.                                                                      |
| <b>Instrument</b>        | AbstractClass | A numeric gauge that is externally scaled by <b>inputLower</b> and <b>inputRange</b> and scaled internally by <b>lower</b> and <b>range</b> . |
| <b>LCD</b>               | Class         | Differs from <b>AlphaNumeric</b> in that the entire gauge window is the printing region.                                                      |
| <b>LCDMixin</b>          | AbstractClass | Computes print region differently from <b>LCD</b> .                                                                                           |
| <b>Meter</b>             | Class         | A circular instrument that wraps around any number of times.                                                                                  |
| <b>RoundScale</b>        | AbstractClass | Abstract Class for instruments with circular (arc) scales.                                                                                    |
| <b>SelfScaleMixin</b>    | AbstractClass | Provides for the gauge to rescale according to the reading.                                                                                   |
| <b>SSBarChart</b>        | Class         | A self-scaling version of <b>BarChart</b> .                                                                                                   |
| <b>SSDigiMeter</b>       | Class         | A self-scaling version of <b>DigiMeter</b> .                                                                                                  |
| <b>SSHBarChart</b>       | Class         | A self-scaling version of <b>HBarChart</b> .                                                                                                  |
| <b>SSHorizontalScale</b> | Class         | Gauge that is displayed in the form of a single scale or bar which rescales itself accordingly.                                               |
| <b>SSVerticalScale</b>   | Class         | Gauge that is displayed in the form of a single vertical scale or bar which rescales itself accordingly.                                      |
| <b>StraightScale</b>     | AbstractClass | Abstract Class for instruments with straight scales.                                                                                          |
| <b>VSGraphics</b>        | AbstractClass | Gauge that is displayed in the form of a single vertical scale or bar.                                                                        |
| <b>VerticalScale</b>     | Class         | Gauge that is displayed in the form of a single vertical scale or bar.                                                                        |

---

**ActiveGaugeMixin**

[Class]

Description: A gauge class that allows you to set the value of the variable being monitored with the cursor, via a **SET** menu.

MetaClass: AbstractClass

Supers: Object

Class Variables: None.

Instance Variables: **cursor** The cursor to use when changing the scale; the default is NIL.

**ActiveHorizontalScale**

[Class]

Description: An active gauge that displays the value on a horizontal scale. This gauge shows the value of the data it is connected with and allows you to change that data with the gauge.

MetaClass: Class

Supers: ActiveGaugeMixin, HorizontalScale

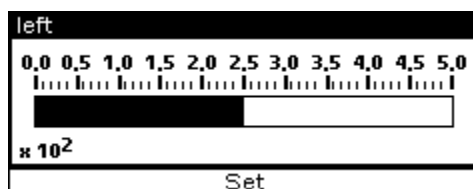
Class Variables: None.

Instance Variables: **cursor** Cursor to use when changing the scale; its property **:initform** is set to **HorizontalAGCursor**.

Example: These gauges have an attached menu at the bottom of the gauge. When you position the cursor over this menu and press a mouse button, the cursor changes to the following shape:



While the left button is held down, the system tracks movements of the cursor and changes the value that the gauge is monitoring.

**ActiveVerticalScale**

[Class]

Description: Similar to **ActiveHorizontalScale**, except that a vertical scale is used.

MetaClass: Class

Supers: ActiveGaugeMixin, VerticalScale

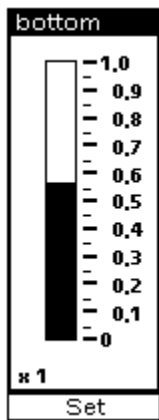
Class Variables: None.

Instance Variables: **cursor** Cursor to use when changing the scale; its **:initform** property is set to **VerticalAGCursor**.

Example: Similar to **ActiveHorizontalScale**. When setting, the cursor changes to the following shape:







---

**AlarmMixin** [Class]

Description: A mixin that adds alarm functionality to any gauge. An alarm is defined as warning object that is set off when the value being monitored falls outside of the specified range. The gauge flashes and stays inverted when the alarm is tripped.

---

**CAUTION**

When a new class of gauges is created that will use the properties of **AlarmMixin**, **AlarmMixin** should be the first class on the Supers list of the new class. This guarantees that the **AlarmMixin.Set** method is invoked.

---

- MetaClass: AbstractClass
- Supers: Object
- Class Variables: MiddleButtonItem
- Instance Variables:
- lowTripPoint** Alarm is triggered when reading goes below this point.
  - hiTripPoint** Alarm is triggered when reading goes above this point.
  - flashNumber** Number of times alarm will flash when it is tripped.
  - flashInterval** Interval in milliseconds between flashes.

---

**AlphaNumeric** [Class]

- Description: This class contains some of the methods and data for the LCD classes. These gauges can display any type of character, letters, or numbers.
- MetaClass: AbstractClass
- Supers: Gauge
- Class Variables: None.

Instance Variables: **precision** Number of characters displayed in the reading. The default value is 5.

**BarChart**

[Class]

Description: A gauge that can display more than one **VerticalScale** at once, side-by side.

MetaClass: Class

Supers: VerticalScale

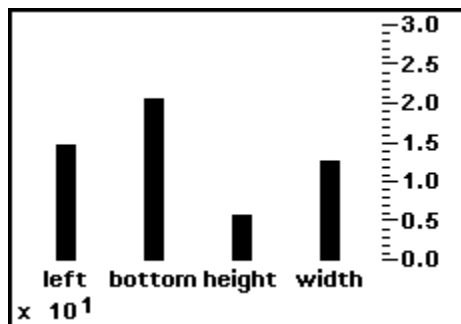
Class Variables: None.

Instance Variables: **maxLabelWidth**  
Maximum width of labels on each bar. Default value is 0 which means no limit.

**scaleLeft**  
Offset within the gauge window from the left for the leftmost bar. Default value is 3.

**scaleBottom**  
Offset within the gauge window from the bottom for all the bars. Default value is 30.

Example: Here is a **BarChart** showing the size and shape of a window. It is displaying the values 15, 21, 13, and 6.

**BoundedMixin**

[Class]

Description: This mixin is a super of the scale classes and **Dial**. If a gauge that has **BoundedMixin** as a super class tries to display a new setting that is outside of the range of the gauge, the gauge will display the minimum or maximum value as appropriate and place a "???" in the window.

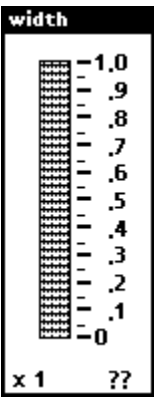
MetaClass: AbstractClass

Supers: Object

Class Variables: None.

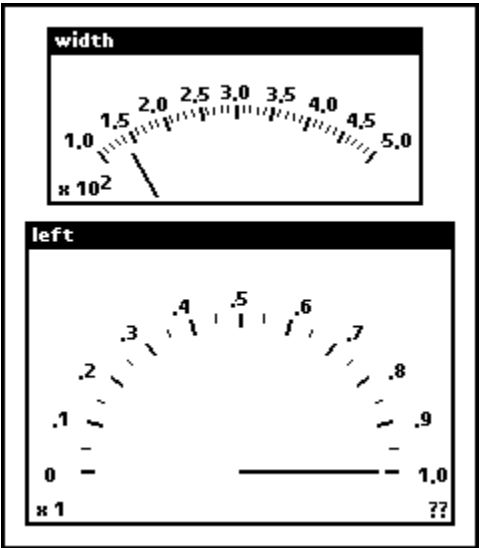
Instance Variables: None.

Example: Here is a vertical scale that displays a reading greater than its maximum.



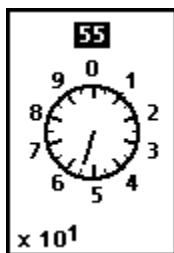
Dial [Class]

- Description: A bounded dial, like an automobile speedometer.
- MetaClass: Class
- Supers: BoundedMixin, RoundScale
- Class Variables: None.
- Instance Variables: This class specializes the same instance variables as **RoundScale**.
- Example: The angle of the arc changes with the shape of the window.



DigiMeter [Class]

- Description: A gauge that combines both a meter and an LCD.
- MetaClass: Class
- Supers: Meter, LCDMixin
- Class Variables: None.
- Instance Variables: **spaceForLCD**  
Vertical space required by LCD within the gauge. Defaults to 30.
- Example: This **DigiMeter** is displaying 55.

**DigiScale**

[Class]

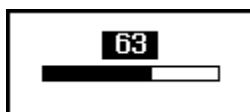
Description: A gauge that combines both a horizontal scale having no ticks and an LCD.

MetaClass: Class

Supers: HorizontalScale, LCDMixin

Class Variables: None.

Example: This **DigiScale** is displaying 63 with its scale set from 0 to 100.

**Gauge**

[Class]

Description: A class for objects that present a dynamic graphical image of a Xerox LOOPS value. This class provides most of the methods for using gauges.

MetaClass: AbstractClass

Supers: Window

Class Variables: **LeftButtonItem**  
Menu options associated with the left mouse button.

**MiddleButtonItem**  
Menu options associated with the middle mouse button.

Instance Variables: **reading** External value of reading. The default value is 0.

**containedInAV**  
Active value that connects the gauge to the data it is monitoring. It should be an instance of the class **GaugeAV**.

**font** Font that is used by a gauge; default value is (Modern 10).

**width** Width of a gauge; has property **min**, which specifies the minimum width for a gauge.

**height** Height of a gauge; has property **min**, which specifies the minimum height for a gauge.

**GaugeAV**

[Class]

Description: An active value that is associated with a gauge.

MetaClass: Class

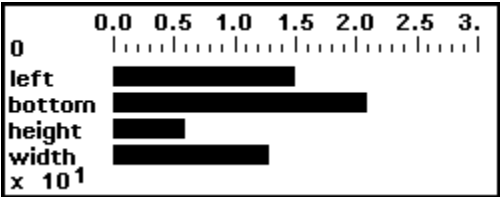
Supers: LocalStateActiveValue

|                     |                 |                                                                      |
|---------------------|-----------------|----------------------------------------------------------------------|
| Class Variables:    | None.           |                                                                      |
| Instance Variables: | <b>gauge</b>    | The gauge connected to this active value.                            |
|                     | <b>object</b>   | The object containing the variable associated with the active value. |
|                     | <b>propName</b> | The property name of the associated variable.                        |
|                     | <b>type</b>     | Data type of the associated variable.                                |
|                     | <b>varName</b>  | Name of the associated variable.                                     |

HBarChart

[Class]

|                     |                                                                                                                    |                                                                                        |
|---------------------|--------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| Description:        | A gauge that can display more than one <b>HorizontalScale</b> at once, side-by-side.                               |                                                                                        |
| MetaClass:          | Class                                                                                                              |                                                                                        |
| Supers:             | HorizontalScale                                                                                                    |                                                                                        |
| Class Variables:    | None.                                                                                                              |                                                                                        |
| Instance Variables: | <b>maxLabelWidth</b>                                                                                               | Maximum width of labels on each bar. Default value is 0 which means no limit.          |
|                     | <b>scaleLeft</b>                                                                                                   | Offset within the gauge window from the left for the leftmost bar. Default value is 3. |
| Example:            | Here is an <b>HBarChart</b> showing the size and shape of a window. It is displaying the values 15, 21, 13, and 6. |                                                                                        |



HorizontalScale

[Class]

|                     |                                                                      |  |
|---------------------|----------------------------------------------------------------------|--|
| Description:        | A labeled, bounded scale with a bar that fills to the right.         |  |
| MetaClass:          | Class                                                                |  |
| Supers:             | HSGraphics                                                           |  |
| Class Variables:    | None.                                                                |  |
| Instance Variables: | None.                                                                |  |
| Example:            | This <b>HorizontalScale</b> is reading 350 on a scale from 0 to 500. |  |



HSGraphics

[Class]

|                     |                                                                                                                                                                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description:        | This class provides some of the methods for displaying the graphics of a horizontal scale.                                                                                                                                                                                                                          |
| MetaClass:          | AbstractClass                                                                                                                                                                                                                                                                                                       |
| Supers:             | StraightScale                                                                                                                                                                                                                                                                                                       |
| Class Variables:    | None.                                                                                                                                                                                                                                                                                                               |
| Instance Variables: | <b>scaleBottom</b> Bottom edge of scale in pixels. The default value is 10.<br><b>scaleLeft</b> Left edge of scale in pixels. The default value is 12.<br><b>scaleWidth</b> Width of inside of scale in pixels. The default value is 120.<br><b>scaleHeight</b> Height of scale in pixels. The default value is 15. |

**Instrument**

[Class]

|                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description:        | A class that provides additional methods and data for gauges that display only numerical data. This data is externally scaled by <b>inputLower</b> and <b>inputRange</b> , and scaled internally by <b>lower</b> and <b>range</b> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| MetaClass:          | AbstractClass                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Supers:             | Gauge                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Class Variables:    | None.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| Instance Variables: | <b>ticks</b> Scale marks on the instrument; value is a number or NIL; <b>smallTicks</b> property indicates the number of smaller ticks between each large tick.<br><b>displayVal</b> Internal value relative to instrument.<br><b>range</b> Range for internal <b>displayVal</b> .<br><b>inputRange</b> Range for external reading.<br><b>lower</b> Lower bound for internal <b>displayVal</b> .<br><b>inputLower</b> Lower bound for external reading.<br><b>brushWidth</b> Scale factor for width of ticks, rays, and circles in pixels.<br><b>labels</b> The labels that will be displayed on the gauge.<br><b>labelScale</b> A dotted pair representing the sign and exponent of a reading.<br><b>spaceForLabelScale</b> Extra vertical space to display scale label. |

**LCD**

[Class]

|                     |                                                                                      |
|---------------------|--------------------------------------------------------------------------------------|
| Description:        | Differs from <b>LCDMixin</b> in that the entire gauge window is the printing region. |
| MetaClass:          | Class                                                                                |
| Supers:             | AlphaNumeric                                                                         |
| Class Variables:    | None.                                                                                |
| Instance Variables: | None.                                                                                |

Example: This **LCD** is displaying the string "Mumble", and has been **Shaped**to 120 x 60.



**LCDMixin** [Class]

Description: Computes printing region differently from LCD so that an LCD may be added into another window.

MetaClass: AbstractClass

Supers: AlphaNumeric

Class Variables: None.

Instance Variables: **precision** Number of characters displayed in the reading; the default value is 3. Its property is **readingRegion**; the default value is NIL.

**readingY** Y position of bottom of reading. The default value is 7.

**Meter** [Class]

Description: A circular instrument that wraps around any number of times. It displays a sign and exponent in the lower left corner of its window.

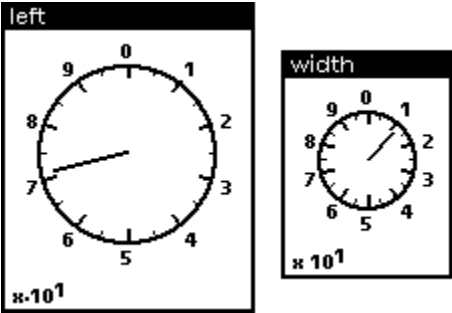
MetaClass: Class

Supers: RoundScale

Class Variables: None.

Instance Variables: This class specializes the same instance variables as **RoundScale**.

Example: The **Meter** on the left is displaying a negative value.



**RoundScale** [Class]

Description: Abstract Class for instruments with circular (arc) scales.

MetaClass: AbstractClass

Supers: Instrument

Class Variables: None.

Instance Variables: **needleLength** Radius of needle in pixels. The default value is 15.

|               |                                                                                                                               |
|---------------|-------------------------------------------------------------------------------------------------------------------------------|
| <b>radius</b> | Radius of arc in pixels. The default value is 10.                                                                             |
| <b>xc</b>     | x-coordinate window coordinate of center of arc. (See <b>DRAWARC</b> in the <i>Xerox Lisp Release Notes, Lyric Release</i> .) |
| <b>yc</b>     | y-coordinate window coordinate of center of arc. (See <b>DRAWARC</b> in the <i>Xerox Lisp Release Notes, Lyric Release</i> .) |

### **SelfScaleMixin** [Class]

|                     |                                                                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description:        | Provides for the gauge to rescale according to the reading.                                                                                                   |
| MetaClass:          | AbstractClass                                                                                                                                                 |
| Supers:             | Object                                                                                                                                                        |
| Class Variables:    | None.                                                                                                                                                         |
| Instance Variables: | <b>lowScaleFactor</b><br>Rescales if reading shrinks so that it will fit more than <b>lowScaleFactor</b> times in <b>inputRange</b> . The default value is 5. |

### **SSBarChart** [Class]

|                     |                                             |
|---------------------|---------------------------------------------|
| Description:        | A self-scaling version of <b>BarChart</b> . |
| MetaClass:          | Class                                       |
| Supers:             | BarChart                                    |
| Class Variables:    | None.                                       |
| Instance Variables: | None.                                       |

### **SSDigiMeter** [Class]

|                     |                                              |
|---------------------|----------------------------------------------|
| Description:        | A self-scaling version of <b>DigiMeter</b> . |
| MetaClass:          | Class                                        |
| Supers:             | DigiMeter                                    |
| Class Variables:    | None.                                        |
| Instance Variables: | None.                                        |

### **SSHBarChart** [Class]

|                     |                                              |
|---------------------|----------------------------------------------|
| Description:        | A self-scaling version of <b>HBarChart</b> . |
| MetaClass:          | Class                                        |
| Supers:             | HBarChart                                    |
| Class Variables:    | None.                                        |
| Instance Variables: | None.                                        |

### **SSHorizontalScale** [Class]

|              |                                                                                                            |
|--------------|------------------------------------------------------------------------------------------------------------|
| Description: | Gauge that is displayed in the form of a single horizontal scale or bar which rescales itself accordingly. |
|--------------|------------------------------------------------------------------------------------------------------------|



MetaClass: Class  
Supers: VerticalScale  
Class Variables: None.  
Instance Variables: None.

---

**SSVerticalScale** [Class]

Description: Gauge that is displayed in the form of a single vertical scale or bar which rescales itself accordingly.  
MetaClass: Class  
Supers: HorizontalScale  
Class Variables: None.  
Instance Variables: None.

---

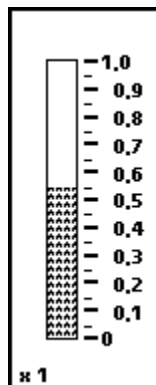
**StraightScale** [Class]

Description: Abstract class for instruments with straight scales.  
MetaClass: AbstractClass  
Supers: BoundedMixin, Instrument  
Class Variables: None.  
Instance Variables: **shade** Shade of bar; numeric value from 0 to 65535. The default value is 65535, which is BLACKSHADE.

---

**VerticalScale** [Class]

Description: Gauge that is displayed in the form of a single vertical scale or bar.  
MetaClass: Class  
Supers: VSGraphics  
Class Variables: None.  
Instance Variables: None.  
Example: This **VerticalScale** is displaying the value .55 and has its **Shade** set to 1258.



**VSGraphics****[Class]**

|                     |                                                                                                                                                                                                                                                                                                                     |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Description:        | Similar to <b>HSGraphics</b> but for vertical scales.                                                                                                                                                                                                                                                               |
| MetaClass:          | AbstractClass                                                                                                                                                                                                                                                                                                       |
| Supers:             | StraightScale                                                                                                                                                                                                                                                                                                       |
| Class Variables:    | None.                                                                                                                                                                                                                                                                                                               |
| Instance Variables: | <b>scaleBottom</b> Bottom edge of scale in pixels. The default value is 12.<br><b>scaleLeft</b> Left edge of scale in pixels. The default value is 15.<br><b>scaleWidth</b> Width of inside of scale in pixels. The default value is 15.<br><b>scaleHeight</b> Height of scale in pixels. The default value is 120. |

**Gauge Methods**

This section describes the available methods and functions which are used to manipulate gauges. In many cases, a particular gauge class specializes a method defined in the class **Gauge**. In this case, the specialized method definition is not explicitly defined; instead, this is noted in the Specializes/Specializations field of the description.

| Name               | Type   | Description                                                                  |
|--------------------|--------|------------------------------------------------------------------------------|
| <b>Attach</b>      | Method | Connects a gauge to an object.                                               |
| <b>Attached?</b>   | Method | Determines what the gauge is attached to.                                    |
| <b>ChangeFont</b>  | Method | Sets the gauge's instance variable <b>font</b> and updates the gauge.        |
| <b>Close</b>       | Method | Detaches the gauge and closes the window.                                    |
| <b>Destroy</b>     | Method | Destroys the gauge, detaching it first.                                      |
| <b>Detach</b>      | Method | Detaches the gauge from the variable it is attached to.                      |
| <b>Reset</b>       | Method | Resets the gauge's instance variable <b>reading</b> .                        |
| <b>SetScale</b>    | Method | Sets the scale for the gauge.                                                |
| <b>Shape</b>       | Method | Sweeps a new region.                                                         |
| <b>ShapeToHold</b> | Method | Shapes the gauge window to its smallest possible size.                       |
| <b>Update</b>      | Method | Reinitializes the gauge and its display window to reflect the current state. |

(← *self* **Attach** *obj varName propName type xOrPos y*)

**[Method of Gauge]**

|            |                                                                                                                                                                                                                                                                                                                    |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Connects a gauge to an object.                                                                                                                                                                                                                                                                                     |
| Behavior:  | Displays the gauge on the screen and associates that gauge with the variable <i>varName</i> of <i>obj</i> . If <i>propName</i> is specified, the gauge will monitor the variable's property. If <i>xOrPos</i> and <i>y</i> are not specified, a small box will appear which must be positioned to place the gauge. |
| Arguments: | <i>obj</i> A pointer to the object to which the gauge is to be attached.                                                                                                                                                                                                                                           |

|                 |                                                                                                                                                                                                                                                                        |
|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>varName</i>  | The name of the instance variable, class variable, or method to which the gauge is to be attached.                                                                                                                                                                     |
| <i>propName</i> | If non-NIL, the gauge will be attached to this property.                                                                                                                                                                                                               |
| <i>type</i>     | One of IV, CV, or METHOD, within the object being connected to the gauge. If NIL, it defaults to IV.                                                                                                                                                                   |
| <i>xOrPos</i>   | A numerical value to specify where, in screen coordinates, the gauge will be placed on the display. If NIL, you are asked to place the gauge on the screen. This can be a number to specify the x coordinate or a position. If it is a number, also specify <i>y</i> . |
| <i>y</i>        | If <i>xOrPos</i> is not a position, this specifies the y coordinate in screen coordinates for the gauge.                                                                                                                                                               |

Returns: *self*

Specializations: **StraightScale.Attach** has an additional *shade* argument so that the shade of the scale may be specified at the time the gauge is attached. The following shows the argument list for this method:

```
(_ ($ instance OfHorizontalScale) Attach obj varName shade propName type
xOrPos y)
```

The **Attach** methods for **BarChart**, **HBarChart**, and their subclasses take an additional *label* argument. If no *label* argument is given, the bar is labeled with *varName*. The *label* argument comes last, as follows:

```
(_ ($ instance OfBarChart) Attach obj varName propName propName type
xOrPos y label)
```

---

(← *self* **Attached?** *don'tPrintFlg*) [Method of Gauge]

---

Purpose: Determines what a gauge is attached to.

Behavior: If *don'tPrintFlg* is non-NIL this returns the value of the gauge instance variable **containedInAV**. If *don'tPrintFlg* is NIL, the **object** and the **varName** the gauge is attached to will be printed in an attached window.

Arguments: *don'tPrintFlg*  
Suppresses displaying what the gauge is attached to.

Returns: NIL

---

(← *self* **ChangeFont** *newFont*) [Method of Gauge]

---

Purpose/Behavior: Sets the gauge's instance variable **font** to *newFont* and updates the gauge. If the gauge is too small for *newFont*, it is reshaped.

Arguments: *newFont* A font in which to display the gauge's text.

Returns: Previous value of **font**.

---

(← *self* **Close**) [Method of Gauge]

---

Purpose/Behavior: Detaches the gauge and closes the window.

Returns: CLOSED

---

(← *self* **Destroy**) [Method of Gauge]

---

Purpose/Behavior: Destroys the gauge, detaching it first before closing the window.

Returns: NIL

(← *self Detach*)

[Method of Gauge]

Purpose/Behavior: Detaches the gauge from the variable to which it is attached. This prints in an attached window that the gauge is being detached, and deletes all of the links connecting the gauge, active value, and object being monitored. Does not close the window.

Returns: NIL

(← *self Reset newReading*)

[Method of Gauge]

Purpose/Behavior: Sets the gauge's instance variable **reading** to *newReading* and updates the gauge. If the gauge is too small for *newReading* and it is **SelfScaling**, it is reshaped.

Arguments: *newReading* Sets the instance variable **reading** to *newReading*, and updates the gauge without going through any intermediate steps.

Returns: NIL if gauge is **AlphaNumeric** or **RoundScale**; otherwise *self*.

Specializations: **Alphanumeric.Reset**, **RoundScale.Reset**

Example: The following example causes the LCD to be redisplayed with the *newReading*:

```
13_(_ ($ lcd1) Reset "New Title")
```

(← *self SetScale min max*)

[Method of Gauge]

Purpose/Behavior: Sets the scale for the gauge; computes the new scale values and redisplayes if necessary.

Arguments: *min* Lowest value on scale.

*max* Highest value on scale.

Returns: *self*

(← *self Shape newRegion noUpdateFlg*)

[Method of Gauge]

Purpose/Behavior: If *newRegion* is NIL, you are prompted to sweep out a region which has a minimum sized based upon a **min** property of **IV width** and **height:;min**. If *newRegion* is non-NIL, it is first checked to guarantee that it is at least as large as **width:;min** by **height:;min**.

Arguments: *newregion* List specifying the external coordinates of the window in which the gauge is displayed; list is of the form (left, bottom, width, height).

*noUpdateFlg* If NIL, reshapes the gauge.

Returns: NIL

Specializes: Window

Specializations: **LCD**, **Meter**, **DigiMeter**. **Meter.Shape** has an extra argument *ExtraSpaceFlg*. If T, this will allow you to shape a fairly arbitrary region for the gauge; if NIL, the meter is constrained to be close to a square. This latter behavior is what the user sees when trying to shape the meter from the window menu.

**BarChart**, **HBarChart**, and their subclasses can only be freely **Shaped** in the direction their bars run (i.e., **BarCharts** can be **Shaped** vertically and **HBarCharts** can be **Shaped** horizontally). Their size along the other dimension is fixed by the number of values attached to the chart.

Example: This example reshapes the gauge to a location where the lower left corner is at (10,100) a width of 50 and a height of 150.

```
14_(_ ($ lcd1) Shape '(10 100 50 150))
```

---

(← *self* **ShapeToHold**)

[Method of Gauge]

Purpose/Behavior: Shapes the gauge window to its smallest possible size based on **width:;min** and **height:;min** and redisplay the gauge.

Returns: NIL

Specializations: **LCD.Shape**

---

(← *self* **Update**)

[Method of Gauge]

Purpose/Behavior: Reinitializes the gauge and its display window to reflect the current state.

Returns: *self*

Categories: Window

---

## Examples

The typical use pattern for a gauge is to first create it, set the scale to the appropriate value, and attach it to the desired data.

To attach a horizontal scale to a Xerox LOOPS window, **w1**, first enter

```
15_(_ ($ Window) New 'w1)
#,(($& HorizontalScale (|OZW0.1Y:.;h.Qm:| . 495))
```

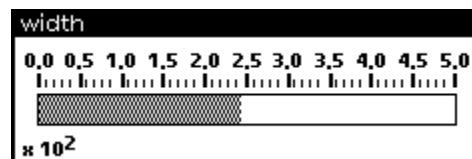
```
16_(_ ($ HorizontalScale) New 'hs1)
#,(($& HorizontalScale (|OZW0.1Y:.;h.Qm:| . 496))
```

```
17_(_ ($ hs1) SetScale 0 500)
NIL
```

Now make the connection.

```
18_(_ ($ hs1) Attach ($ w1) 'width GRAYSHADE)
#,(($& HorizontalScale (|OZW0.1Y:.;h.Qm:| . 496))
```

The following gauge appears and you are prompted to place it .



The title of the gauge shows the instance variable being monitored.

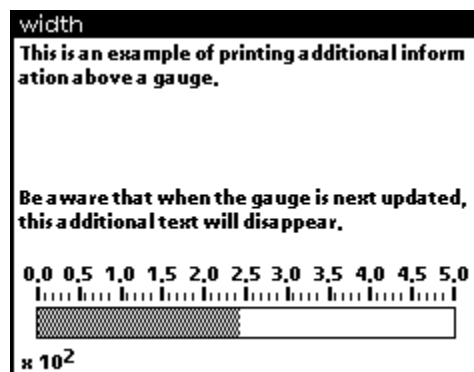
Gauges can be shaped larger. The graphics used to display scales do not change; extra white space is added to the top or right. You can use this space to print additional information, as follows:

```
19 _ (MOVETOUPPERLEFT (@ ($ hs1) window))
 {WINDOW}#372,7104
```

```
20 (PRIN1 "This is an example of printing additional
information above a gauge.
```

Be aware that when the gauge is next updated, this additional text will disappear." (@ (\$ hs1) window))  
 "This is an example of printing additional information above a gauge.

Be aware that when the gauge is next updated, this additional text will disappear."



## Limitations

When a font is changed, a gauge occasionally needs to be updated to be correctly displayed.

Instruments can have only floating point numbers for labels, and cannot have integers.

[This page intentionally left blank]

---

## Description/Introduction

---

Masterscope has been modified to provide for analysis of files created under the Koto or Lyric/Medley release of Xerox LOOPS. A full explanation of Masterscope can be found in the *Xerox Lisp Library Modules Manual, Lyric and Medley Releases*. In addition to the relations explained there, Xerox LOOPS defines the relations described in this chapter.

Note: Masterscope data base files created under Buttress Loops will not function properly in this release. Those data base files will have to be recreated.

---

## Installation/Loading Instructions

---

- Load MASTERSCOPE from your Lyric/Medley library floppies according to its loading instructions. This should load the compiled files MASTERSCOPE, MSANALYZE, and MSPARSE.
- Load LOOPSMS.DFASL from wherever you installed the Xerox LOOPS Library Modules. This should load versions of MASTERSCOPE and MSPARSE that extend Masterscope to handle Xerox LOOPS constructs.

---

## Relations

---

Xerox LOOPS defines the following relations:

| Name                | Type     | Description                                                                                             |
|---------------------|----------|---------------------------------------------------------------------------------------------------------|
| <b>SEND</b>         | Relation | Collects all places where the method is sent.                                                           |
| <b>SEND SELF</b>    | Relation | Collects all places where the method is sent to <i>self</i> .                                           |
| <b>SEND NOTSELF</b> | Relation | Collects all places where the method is sent to an object other than <i>self</i> .                      |
| <b>GET</b>          | Relation | Locates all places where the value of an instance variable is retrieved.                                |
| <b>GET CV</b>       | Relation | Locates all places where the value of a class variable is retrieved.                                    |
| <b>PUT</b>          | Relation | Locates all places where the value of an instance variable is set.                                      |
| <b>PUT CV</b>       | Relation | Locates all places where the value of a class variable is set.                                          |
| <b>IMPLEMENT</b>    | Relation | Locates all methods that specialize the given selector.                                                 |
| <b>SPECIALIZE</b>   | Relation | Locates all methods that specialize the given selector and use <b>_Super</b> in the body of the method. |



|                   |          |                                                                                                                              |
|-------------------|----------|------------------------------------------------------------------------------------------------------------------------------|
| <b>OVERRIDE</b>   | Relation | Locates all methods that specialize the given selector and do not use <b>_Super</b> in the body of the method.               |
| <b>USE IV</b>     | Relation | Used with an instance variable name to locate all places where the instance variable is used in a <b>GET</b> or <b>PUT</b> . |
| <b>USE CV</b>     | Relation | Used with a class variable name to locate all places where the class variable is used in a <b>GET</b> or <b>PUT</b> .        |
| <b>USE OBJECT</b> | Relation | Used with an object name to locate all places where the object is used.                                                      |

---

**SEND**

[Relation]

Purpose/Behavior: Used between method names and selectors to collect all places where the method is sent. For example, the form

```
. WHO IS SENT BY 'Helicopter.Move
```

works, but

```
. WHO IS SENT BY Move
```

does not work.

Example: The following command allows you to edit all code that sends the message **New**.

```
. EDIT ALL WHO SEND New
```

---

**SEND SELF**

[Relation]

Purpose/Behavior: Used between method names and selectors to collect all places where the method is sent to *self*. Places where

```
(← self methodName)
```

is found are collected, while places where

```
(← otherInstance methodName)
```

is found are not.

Example: The following command allows you to edit all code that sends the message **Clear** to *self*.

```
. WHO SENDS SELF Clear
```

---

**SEND NOTSELF**

[Relation]

Purpose: Same as **SEND SELF**, except the only places where the message is sent to an object other than *self*.

Example: The following allows you to edit all code that sends the message **Clear** to any instance other than *self*.

```
. SHOW ALL WHO SEND NOTSELF Clear
```

---

**GET** [Relation]

Purpose: Used with an instance variable name to locate all places where the value of the instance variable is retrieved. This relation can be used along with the **SELF** and **NOTSELF** modifiers.

Example: This command allows you to edit all code that gets the value of the instance variable **width** from an instance other than self and the value of the instance variable **height** from *self*.

```
. SHOW ALL WHO GET NOTSELF width AND GET SELF height
```

---

**GET CV** [Relation]

Purpose: Same as **GET**, except that **GET CV** locates places where the value of the class variable is retrieved. This relation can be used with the **SELF** and **NOTSELF** modifiers.

Example: This command allows you to edit all code that accesses the value of the class variable **height** of *self*.

```
. SHOW ALL WHO GET CVSELF height
```

---

**PUT** [Relation]

Purpose: Used with an instance variable name to locate all places where the value of the instance variable is set. This relation can be used along with the **SELF** and **NOTSELF** modifiers.

Example: This command allows you to edit all code that sets the value of the instance variable **width**.

```
. EDIT ANY WHO PUT width
```

---

**PUT CV** [Relation]

Purpose: Same as **PUT**, except locates places where a specified class variable is set. This relation can be used along with the **SELF** and **NOTSELF** modifiers.

Example: This command list all the sections of code that set the value of the class variable **width** for an instance other than *self*.

```
. WHO PUTS CV NOTSELF width
```

---

**IMPLEMENT** [Relation]

Purpose: Used with a method name to locate all methods that specialize the given selector.

Example: This returns a list of classes where the method **Clear** is defined.

```
. WHO IMPLEMENTS Clear
```

---

**SPECIALIZE** [Relation]

Purpose: Used with a method name to locate all methods that specialize the given selector and use **\_Super** in the body of the method.

Example: This command allows you to edit all the methods that are specializations of **Clear** and use the **\_Super** form.

```
. EDIT ANY WHO SPECIALIZE Clear
```

---

**OVERRIDE****[Relation]**

Purpose: Like **SPECIALIZE** above, except it locates all methods that specialize the given selector and **\_Super** is not used in the body of the method.

Example: This command allows you to edit all the specializations of **Clear** that do not make use of the **\_Super** form.

```
. EDIT ALL WHO OVERRIDE Clear
```

---

**USE IV****[Relation]**

Purpose: Used with an instance variable name to locate all places where the instance variable is used in a **Get** or **Put**. It is equivalent to using the relation form of **GET IVName** or **PUT IVName**.

Example: This command allows you to edit all code that either sets or accesses the instance variable **width**.

```
. EDIT ANY WHO USE THE IV width.
```

---

**USE CV****[Relation]**

Purpose: Used with a class variable name to locate all places where the class variable is used in a **Get** or **Put**. It is equivalent to using the relation form: **GET CV CVName OR PUT CV CVName**.

Example: This command allows you to edit all code where the class variable **commonWindow** is either set or accessed.

```
. EDIT ANY WHO USE THE CV commonWindow
```

---

**USE OBJECT****[Relation]**

Purpose Uses an object name to locate all places where the object is used.

Example This command returns a list of all code where the object **Window** is used.

```
. WHO USES THE OBJECT Window??
```

---

**Limitations**

---

Masterscope has several limitations:

- Names of methods must be quoted when used with Masterscope; for example, the method name `Helicopter.Move` must be entered as `'Helicopter.Move'`.
- The following expression will not find a call to **GetValue** when in a method :

```
. WHO CALLS GetValue
```

Masterscope does not record calls to **GetValue** and **PutValue** explicitly; it records them under the Get- relation along with calls of the form

```
(_ foo Get 'bar)
```

Similarly, the following functions are recorded under relations instead of their names:

|                      |        |
|----------------------|--------|
| <b>GetClassValue</b> | Get CV |
| <b>PutClassValue</b> | Put CV |
| <b>GetClassIV</b>    | Get IV |
| <b>PutClassIV</b>    | Put IV |

If you want to find the explicit calls to Get/PutValue, use

```
. WHO GETS ANY AND NOT SENDS Get
```

- similar accessors are accessing instance variables; i.e.,

```
(GetValue foo 'bar)
```

records an access to the instance variable **bar**. This is not necessarily the case; **bar** could also be a class variable.

- The methods and functions that create class and instance variables populate the appropriate **PUT NOTSELF** relations. For example, a function that does

```
(_($ foo) AddCV 'bar)
```

will be found by the query

```
. WHO PUTS CV NOTSELF 'bar
```

An exception occurs with the generalized **Add** and **Delete** method. For example,

```
($ foo) Add 'IV 'bar)
```

will not be noticed as accessing the instance variable **bar**.

Also, the templates for methods and functions that accept property lists generally only notice the first property. For example,

```
((_($ foo) NewWithValues ' (bar baz chain link
sausage)))
```

notices **baz** as a property, not a link.

[This page intentionally left blank]

# Writer's Notes -- Production Details

---

This file includes notes on the production of *Xerox LOOPS Library Modules Manual*, Lyric Beta Release. This manual is packaged with the *Xerox LOOPS Release Notes* and *Xerox LOOPS Reference Manual* to form one binder.

Writer: Raven Kontur Brewster

Printing Date: >>DA<< >>MO<< 1988

## Files Needed

---

To edit or print the manual, make sure you have the following files loaded:

IMTOOLS  
SKETCH  
GRAPHER

## Fonts Used

---

{ERIS}<LISP>FONTS>

Modern font  
18-point bold  
14-point bold  
12-point bold  
10-point regular  
10-point italic  
10-point bold

Terminal font  
10-point regular

## Printing Information

---

The manual was printed under a Lyric sysout on the Tsunami printer.

## Artwork

---

- The cover page for the binder is in the file {ERIS}<Doc>Loops>Lyric>Beta>BinderCover.tedit.

## Special Notes and Cautions

---

If you bring the file into a TEdit window to print it, you must first make sure your underscore character is redefined as a left arrow. See the file on conventions for details. This restriction does not apply if you use the Hardcopy option from the File Browser.



---

## Description/Introduction

---

In many knowledge-based systems, it is useful to represent knowledge as interconnected sets of instances. A virtual copy mechanism allows a network of instances to be viewed as a prototype which can be copied. The copy of the prototype is virtual in that the contents of each instance is not completely copied at creation time. Instead, it inherits default values from the prototype (also called the original), thus continuing to share the parts not modified in the copy. The copied network is virtual also in the sense that only those instances needed in the processing are copied.

A virtual copy of an object in the prototype network has the following properties:

- It responds to at least the same set of messages as the prototype object and in the same way; that is, a copy has the same procedural behavior that is defined for the prototype.
- A copy inherits variables and their values from the prototype, and continues to do so until an explicit change is made in the copy. At that point, the new value is stored in the copy and it stops tracking the prototype for that variable. A fetch operation on a value that is not stored locally either finds or creates a virtual copy of the value obtained from the prototype.

---

## Installation/Loading Instructions

---

The implementation of virtual copies is contained in the file `LOOPSCOPY.LCOM`. No other files are necessary.

---

## Application /Module Functionality

---

A network of instances is tied together through the values of instance variables within each of the instances. Assume an object A has an instance variable x, the value of which is the object B. A virtual copy of A will also have an instance variable named x. The value of x in the copy will point to B if B is a shared object, or x may point to a copy of B if it is to be virtual. Changing the value of x in the copy will not change the value in the original.

---

## Overview of Operation

---

By default, virtual copies share instance variables. This means that changing the value of an instance variable in the original will be tracked by the copy.

Virtual copies are implemented with two additional classes:

- **VirtualCopyMixin**

The class **VirtualCopyMixin** is a subclass of Tofu which contains two instance variables:



- % **copyMap**%
- % **copyOf**%

(These unusual names are used to avoid conflicts with any other instance variable names users may create.) This class contains several methods, most of which are required to implement virtual copies and are not used by a programmer.

Printing a virtual copy instance is a specialization of how regular instances are printed. All instances print as #,(\$& <class-name> UID). The class of a virtual copy is a dynamic mixin of the class **VirtualCopyMixin** and the class of the original object (see the *Xerox LOOPS Reference Manual* for more information on mixins). The virtual copy print function adds the name or unique identifier (UID) of the original object. For example,

```
#,($& (VirtualCopyMixin Container1) (JFW0.0X:.aF4.R>8 . 3) c1)
```

is a copy of the object named **c1**.

- **VirtualCopyContext**

The class **VirtualCopyContext** has no methods and only one instance variable, **copyMap**. Instances are used as an argument for calls to **MakeVirtualMixin**.

Since copies can be made of copies, you often need to determine the original object of a chain of copies with the **UltimateOriginal** function.

---

## Operands

This section describes the functions, methods, class variables, and instance variables that operate on virtual copies.

---

### VirtualIVs

[Class Variable]

**Purpose/Behavior:** Helps specify a class whose instances may be made into virtual copies. The value of this class variable should be either the symbol **ALL**, or a list of instance variables contained within instances of the class. If the value is **ALL**, all objects pointed to by any of the instance variables will be copied. If the value is a list of instance variables, only the instance variables on this list will have their values copied. Other instance variable values will be shared between the copy and the original.

---

### (**MakeVirtualMixin** *x copyContextObj*)

[Function]

**Purpose:** Creates a virtual copy of an object.

**Behavior:** Creates a dynamic mixin class combining the classes **VirtualCopyMixin** and the class of *x*. An instance of this resulting class is created and it is returned.

**Arguments:** *x*                      An object to be copied; must have the class variable **VirtualIVs** as described above.

*copyContextObj*

Usually **NIL**; used internally by **MakeVirtualMixin** when it calls itself. It can be an instance of **VirtualCopyContext** if you are creating an instance that is intended to be part of a currently existing network of copies starting from another entry point. See description in **Limitations** below for a further explanation of this point.

**Returns:** An object that is a copy of *x*.

Example: Refer to the section, "Example."

**% copyMap%** [Instance Variable of VirtualCopyMixin]

Purpose/Behavior: A mapping of original nodes (which are objects) in a network to the copied nodes. This map is stored in an instance of the class **VirtualCopyContext**.

**% copyOf%** [Instance Variable of VirtualCopyMixin]

Purpose/Behavior: Within an instance that is a copy, the value of this instance variable is a pointer to the object that was copied.

**(← self VirtualCopy?)** [Method of VirtualCopyMixin]

Purpose: Determines if an object is a virtual copy.

Returns: *self*

Categories: Object, VirtualCopyMixin

**copyMap** [Instance Variable of VirtualCopyContext]

Purpose/Behavior: The value of this instance variable is a list of dotted pairs. The CAR of each pair is the original; the CDR, the copy.

**(UltimateOriginal self)** [Function]

Purpose: Determines what an object is ultimately copying.

Behavior: If *self* is not a virtual copy, *self* is returned.

If *self* is a virtual copy, this recurses through the value of the instance variable **% copyOf%** until it finds the original and returns it.

Arguments: *self* A Xerox LOOPS object.

Returns: *self* or what is at the top of *self*'s copy chain.

## Example

Create a class called **test** and edit it as shown.

```
44_(_ ($ Class) New 'test)
#,($C test)

45_(ED 'test)
```

```
SEdit #,($C test) Package: INTERLISP
((MetaClass Class Edited%: ; Edited 11-Dec-87
 ; 16:50 by
)
 (Supers Object)
 (ClassVariables (VirtualIVs (atomCopy listCopy objCopy)))
 (InstanceVariables (atom NIL)
 (atomCopy NIL)
 (list NIL)
 (listCopy NIL)
 (obj NIL)
 (objCopy NIL))
 (MethodFns))
```

Create an instance called **t0** of this class and inspect it.

```
46_(_ ($ test) New 't0)
#,($& test (N↑W0.1Y%.:;h.Lh9 . 556))

47_(_ ($ test)
 NewWithValues
 (BQUOTE ((atom 1)
 (atomCopy 2)
 (list (a b c))
 (listCopy (A B (\, (_ ($ test) New (QUOTE t1))))))
 (obj (\, (_ ($ test) New (QUOTE t2))))
 (objCopy (\, (_ ($ test) New (QUOTE t3))))))
#,($& test (N↑W0.1Y%.:;h.Lh9 . 560))

48_(_ IT SetName 't0)
#,($& test (N↑W0.1Y%.:;h.Lh9 . 560))

49_(INSPECT IT)
{WINDOW}#52,51234
```

```
All Values of test ($ t0).
atom 1
atomCopy 2
list (a b c)
listCopy (A B #,($ t1))
obj #,($ t2)
objCopy #,($ t3)
```

Make a copy called **t0copy** and inspect it.

```
57_(_ (MakeVirtualMixin ($ t0))
 SetName
 (QUOTE t0copy))
#,($& (VirtualCopyMixin test) N↑W0.1Y%.:;h.Lh9 . 562)
```

```
58_ (INSPECT IT)
{WINDOW}#53,10150
```

```
All Values of (VirtualCopyMixin test) ($ t0copy).
atom NIL
atomCopy NIL
list NIL
listCopy NIL
obj NIL
objCopy NIL
| copyOf | #,($ t0)
| copyMap | #,($& VirtualCopyContext (N+V0.1Y%.:;h.Lh9 . 563))
```

Make the following changes to **t0** and then reinspect **t0copy**.

```
60_ (for iv in '(atom atomCopy list listCopy obj objCopy)
as val in (LIST 11 22 '(a b c d) '(A B C) ($ t3) ($ t1))
do (PutValue ($ t0) iv val]
NIL
```

```
61_ (INSPECT IT)
{WINDOW}#53,10152
```

```
All Values of (VirtualCopyMixin test) ($ t0copy).
atom 11
atomCopy 22
list (a b c d)
listCopy (A B C)
obj #,($ t3)
objCopy #,($& (VirtualCopyMixin test) (N+V0.1Y%.:;h.Lh9 . 565) t1)
| copyOf | #,($ t0)
| copyMap | #,($& VirtualCopyContext (N+V0.1Y%.:;h.Lh9 . 566))
```

The copied instance variables have not changed since they do not track changes in the original object.

## Limitations

Some subtle issues are involved in building and using prototype structures so that the structure is preserved in the copied network. These involve how the network is typically traversed.

A general constraint is that all the links to any shared node in the prototype either all be marked as virtual variables, or none of them are. If they are all marked, then a single copy will be made and used. If none are, then the original object from the prototype will be used. Sharing with the prototype can be useful if this object is a repository for standard information that is independent of context. However, if this constraint is violated, the topology of the virtual copy will be different from that of the prototype.

In the simplest situation the network has a single entry node. In this case, a copy-map (see the section "Operands") can be created when the entry node object is first copied. After that all values are copied using this copy-map. The mechanism works well in this situation, even if there is sharing and there are cycles within the network.

At the other extreme, networks can have arbitrary connectivity, including multiple entries from outside the network, for example, from other networks or

non-objects. In this case, the following constraints are necessary to ensure correctness of the virtual copy mechanism.

The first constraint states that all access to the network must start through a copy of one of the nodes in the prototype. This condition is necessary because the criteria for copying are contained in the links from one object to another, not in the objects themselves, and a shared node could not specify a link to a node to be copied. This constraint ensures that all accesses from the outside will be copied if and only if that object would have been copied because of an internal link. Otherwise, an analogous situation would occur in which you could either reach a copy or the original node of the prototype itself depending upon which path you follow when the paths lead to the same node in the prototype.

The final constraint requires that all entries to the network should be passed the same copy-map if they are to share structure. The underlying concern in imposing these constraints is that a network be always copied the same way to maintain its topology regardless of where you start.

Suppose you want to make a virtual copy of a virtual copy, that is, to use a virtual copy of a network as a prototype itself. This is very useful if you are using a network to hold the state of a partial design and you want to try two alternative continuations of the design. Some hidden costs are associated with such multiple-level virtual copies.

Suppose further that a network N1 is used as a prototype and you make a virtual copy, N1-VC. Furthermore, N1-VC-VC is defined to be a copy of N1-VC. Values missing from N1-VC-VC are found in the corresponding object of N1-VC. If the value is missing there, the process recurs, and N1 is examined. If the value is to be a virtual copy, then this process will add a virtual copy in N1-VC, and then a second level copy in N1-VC-VC. This is necessary to preserve the semantics presented, but implies that many levels of virtual copy cannot easily do inexpensive incremental searches of a network.

---

## References

---

Mittal, S. , Bobrow, D. G., and Kahn, K. *Virtual Copies, Between Classes and Instances*. ACM OOPSLA-86 Conference Proceedings, Portland, Oregon, 1986.

This directory contains all the documentation for Xerox Loops in all its various incarnations.  
Directories are as follows:

{ERIS}<Doc>LOOPS> - most general information  
{ERIS}<Doc>loops>ProductionSpecs> - specs for all versions of Koto LOOPS

{ERIS}<Doc>Loops>Koto>Final - Product Release for Koto LOOPS (Oct 87)  
{ERIS}<Doc>Loops>Lyric>Alpha - Alpha Lyric LOOPS (Jan 88)

Most directories have the following subdirectories:

Ref> - Reference Manual  
RelNote> - Release Notes  
LibMod> - Library Modules Manual (Lyric Release)  
LibPkg> - Library Packages Manual (Koto Release)  
UserMod> - Users' Modules Manual (Lyric Release)  
UserPkg> - Users' Packages Manual (Koto Release)

And even further in the diectory maze,  
X-Index> - has IMPTR files and resulting index  
Z-ReleaseInfo> - holds more details on conventions and production details.

Enjoy,  
Raven Brewster

This appendix describes how to install the LOOPS System files, Library Modules files, and Users' Modules files on the Sun Workstation.

---

## Overview of the Distribution Kit

---

The distribution kit for LOOPS on the Sun consists of a single ¼-inch tape cartridge. It contains the complete release in "tar" format and creates appropriate directories when its contents are extracted.

---

## Preparation

---

Preparing to install LOOPS requires that the Medley release of Lisp is already installed and that adequate file space is available.

Before installing LOOPS, remember that

- the Medley 1.0, 1.1 or 1.2 release of Lisp must already be installed on your Sun Workstation;
- the complete LOOPS distribution requires about 1.2 MBytes of file space.

---

## Installation

---

The software installation procedure shows the steps required for installing the Medley LOOPS software on a Sun Workstation with Medley already installed. Examples are given where appropriate. Only those users who are system administrators and have **root** privileges can install the LOOPS, Medley release.

Before starting software installation, remember that the LOOPS software requires about 1.2 MBytes of file space.

1. Log in under your username.

```
login yourname
```

```
prompt%
```

where **yourname** is replaced by your username.

2. Check for available space with the **df** command:

```
prompt% df
```

| Filesystem | kbytes | used | avail  | capacity | Mounted on |
|------------|--------|------|--------|----------|------------|
| /dev/xy0a  | 7437   | 5470 | 1223   | 82%      | /          |
| /dev/xy0h  | 148455 | 4900 | 128709 | 96%      | /usr/misc  |

3. Determine if you need to run **su** to make a directory for the distribution. If so, type in **su**:

```
prompt% su
```

4. Make a directory for the distribution. This directory should be named **/usr/local/lde/loops**. If you have enough space on the file system containing **/usr/local/lde**, then

```
prompt# mkdir /usr/local/lde/loops/
```

If you don't have enough space on **/usr/local/lde**, go to step 6.

5. Make yourself owner of this directory:

```
prompt# /etc/chown yourname /usr/local/lde/loops/
```

where **yourname** is your username.

6. If you don't have space on the file system which contains **/usr/local/lde**, but do have space somewhere else, for instance on **/usr1**, then make the directory there and link **/usr/local/lde/loops** to it:

```
prompt# mkdir /usr1/loops
```

```
prompt# /etc/chown yourname /usr/usr1/loops
```

```
prompt# ln -s /usr1/loops /usr/local/lde/loops
```

7. If you ran **su**, leave the privileged shell by typing:

```
prompt% exit
```

8. Insert the ¼-inch cartridge tape, containing the LOOPS software, in the drive.

9. Connect to **/usr/local/lde/loops**:

```
prompt# cd /usr/local/lde/loops
```

10. Load the Medley software from tape. Indicate the appropriate device abbreviation for your tape by replacing **xx** in the example below with

**ar** for the Archive drive,

**st** for a SCSI tape drive.

This example shows the command entry sequence:

```
prompt# tar xvpf /dev/rxx0
```

As the software is loaded (a process that takes some time) the system prints a series of lines in the following form:

```
x ./system/LOOPS., 28552 bytes, 56 tape blocks
```



The x at the beginning of the line indicates that the file is being extracted from the tape.

This creates directories named:

/usr/local/lde/loops/system/

/usr/local/lde/loops/library/

/usr/local/lde/loops/users/

This is a good time to set the protection of the extracted directories and files so that the work group using LOOPS has at least read access to them.

11. Boot Medley Lisp.
12. Open an Interlisp Executive Window.
13. Make certain the time is set correctly.
14. Set the **DIRECTORIES** and **DISPLAYFONTDIRECTORIES** variables appropriately so the sysout can find your Lyric/Medley library and font files.
15. Make the LOOPS System directory your connected directory:

```
CONN {DSK}/usr/local/lde/loops/system/
```

16. Enter the following into your Exec:

```
LOAD (LOOPS)
```

A menu appears that looks like this:

```

Loops system
Install from distribution
Load into sysout

```

17. Select the menu option **Install from distribution**.

The following menu appears:

| Loops directories       | Click here when           |
|-------------------------|---------------------------|
| LOOPSDIRECTORY          | {DSK}<LISPFILES>LOOPS>    |
| LOOPSLIBRARYDIRECTORY   | {DSK}<LISPFILES>LOOPS>LIB |
| LOOPUSERSDIRECTORY      | {DSK}<LISPFILES>LOOPS>USE |
| LOOPUSERSRULESDIRECTORY | {DSK}<LISPFILES>LOOPS>RUL |

This menu shows the current (or default, if unset) values of the variables LOOPS examines when it loads things.

If you have installed LOOPS under /usr/local/lde/loops/ click the mouse on the menu items to set these directories to point where the tape was unloaded:

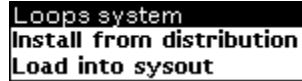
```

LOOPSDIRECTORY {dsk}/usr/local/lde/loops/system/
LOOPSLIBRARYDIRECTORY {dsk}/usr/local/lde/loops/library/
LOOPUSERSDIRECTORY {dsk}/usr/local/lde/loops/users/
LOOPUSERSRULESDIRECTORY {dsk}/usr/local/lde/loops/users/

```

As the last installation step, the installation tool automatically modifies the file LOOPSSITE, writes it out to the value of the variable **LOOPSDIRECTORY**, and compiles it.

When this step is finished, the first menu reappears:



```
Loops system
Install from distribution
Load into sysout
```

18. Select the menu option **Load into sysout** to load LOOPS into your system. The following menu appears:



```
Load Which?
Loops
Loops Masterscope
Gauges
LoopsBackwards
VirtualCopy
```

19. Select **LOOPS** from the menu to load LOOPS from the location where you installed it.

Once LOOPS is loaded the LOOPS System menu reappears. To load one of the other LOOPS library or Users' modules, select the appropriate name in the Load Which? menu.

20. Position your mouse cursor anywhere on the screen except for the Load Which? menu, then press the left mouse button to exit the installation procedure.

Medley LOOPS is now installed on your Sun Workstation.

---

## Loading After Installation

---

This section describes how to reload LOOPS into a newly started Lisp sysout after LOOPS has been previously installed.

1. Start up Medley on your Sun Workstation.
2. Open an INTERLISP Exec window.
3. Make sure **DIRECTORIES** points to a directory containing **GRAPHER.LCOM**, and **DISPLAYFONTDIRECTORIES** points to a directory containing the Helvetica display font files from your Lisp distribution kit.

4. Connect to the directory containing the LOOPS system files:

```
(CNDIR ' {DSK} /USR/LOCAL/LDE/LOOPS/SYSTEM/)
```

5. Load LOOPS loader program:

```
(FILESLOAD LOADLOOPS)
```

6. Run the LOOPS loader program:

```
(LOADLOOPS)
```

This procedure loads only the LOOPS system files. Please see the manuals describing the LOOPS Library and Users' Modules for their loading procedures.

---

**CAUTION**

LOOPS uses the new compiler and its macrolet facilities. When LOOPS is loaded, it sets your **\*DEFAULT-CLEANUP-COMPILER\*** to **'CL:COMPILE-FILE**. More information on this cleanup flag and the new compiler is available in the *Lisp Release Notes*, in your Medley Lisp kit.

---

[This page intentionally left blank]

## DOCUMENT UPDATE SHEET

---

**Document Name:**    *LOOPS Manual*

---

---

**Document Number:**   *310000*

---

| DOC.<br>VERSION | RELEASE<br>DATE | REPLACE<br>PAGES | INSERT<br>PAGES | INSTRUCTIONS/<br>NOTES                                                                                              |
|-----------------|-----------------|------------------|-----------------|---------------------------------------------------------------------------------------------------------------------|
| Lyric/Medley    | Oct., 1988      | NA               | NA              | Please read the Errata Sheet, accompanying this release material, for last minute release notes.                    |
| Lyric/Medley    | Oct., 1988      | NA               | NA              | The Lyric/Medley LOOPS documentation contains numerous references to Xerox LOOPS. Xerox LOOPS is now known as Envos |
| LOOPS.          |                 |                  |                 |                                                                                                                     |
| Medley          | Oct., 1988      | NA               | A-1-A-4         | Add <i>Appendix A, Sun Installation Procedure</i> , to your <i>LOOPS Release Notes</i> .                            |

BACK COVER

FONT

*Xerox Corporation*  
*AIS Administration*  
*250 North Halstead St.*  
*5910-432*  
*800-824-6449 (Calif.)*  
*800-228-5325 (U.S.)*  
*Pasadena, California 91107*

REGULAR or  
TRIUMVIRAT  
BLACK  
ITALIC

COVER TEXT:

NOTE: USE APPROPRIATE POINT SIZES TO ACHIEVE EFFECT  
SHOWN BELOW. USE XEROX STANDARDS FOR <<8-1/2x11>> BINDER COVERS

FONT

ARTIFICIAL INTELLIGENCE SYSTEMS  
XEROX LOOPS

HELVETICA  
BOLD ITALIC

RELEASE NOTES  
REFERENCE MANUAL  
LIBRARY PACKAGES MANUAL  
  
RELEASE 1.0

REGULAR or  
TRIUMVIRATE  
BLACK  
ITALIC

SPINE (3 INCH) :

ARTIFICIAL INTELLIGENCE SYSTEMS  
XEROX LOOPS  
  
RELEASE NOTES  
REFERENCE MANUAL  
LIBRARY PACKAGES MANUAL  
  
RELEASE 1.0

HELVETICA  
BOLD ITALIC

REGULAR or  
TRIUMVIRATE  
BLACK  
ITALIC

Subject: Software Configuration: Xerox LOOPS Documentation, Release 1.0  
To: Fournier.pasa

I. This is a request to configure for production the artwork for the following Documentation Kit.

Kit: Xerox LOOPS, Release 1.0

For AICCB use only. Xerox Part Number: >>12R number<<  
Internal Part Number: >>our number<<

## **DOCUMENTATION SET**

**# of Books in Set: 4**

**Volume Names:** *Xerox LOOPS Reference Manual*  
*Xerox LOOPS Release Notes*  
*Xerox LOOPS Library Packages Manual*  
*Xerox LOOPS Users' Packages Manual*

II. This is a request for artwork for the Documentation Kit.

## **BINDING**

**Volume Names:** *Xerox LOOPS Release Notes*  
*Xerox LOOPS Reference Manual*  
*Xerox LOOPS Library Packages Manual*

**Book Size:** 8-1/2 X 11

**Binder Size:** 3 inch

**Binding Type:** vinyl

**Book Cover Material:**

**Other Binding Specifications:** plastic sleeves for slipin on front, spine, and back

**Binder Rings:**

**Sheet Lifters:** yes

**Inside Pocket:**

**Binder Colors:** PMS Gray 422-C

**Special Instructions:** Bind all manuals in one binder

## **BINDING**

**Volume Names:** *Xerox LOOPS Users' Packages Manual*

**Book Size:** 8-1/2 X 11

**Binder Size:** 1 inch

**Binding Type:** vinyl



**Book Cover Material:****Other Binding Specifications:** plastic sleeves for slip in on front, spine, and back**Binder Rings:****Sheet Lifters:****Inside Pocket:****Binder Colors:** PMS Gray 422-C**Special Instructions:**

Please see the following files for actual cover artwork specifications:

```
{ERINYES}<doc>LoopsRef>ProductionSpecs>BackCover.Sketch;2
{ERINYES}<doc>LOOPSREF>PRODUCTIONSPECS>ComboCoverSpine.sketch;1
{ERINYES}<doc>LOOPSREF>PRODUCTIONSPECS>UsersCoverSpine.sketch;2
```

**TABS**

Tab Set Part Number: &gt;&gt;For AICCB use: Part Number&lt;&lt;

Please see the following files for the tab specifications:

```
{ERINYES}<doc>LOOPSREF>PRODUCTIONSPECS>TabSample1a.Sketch;3
{ERINYES}<doc>LOOPSREF>PRODUCTIONSPECS>TabSample1B.Sketch;6
{ERINYES}<doc>LOOPSREF>PRODUCTIONSPECS>TABSAMPLE1C.Sketch;2
{ERINYES}<doc>LOOPSREF>PRODUCTIONSPECS>TabSample2A.sketch;3
{ERINYES}<doc>LOOPSREF>PRODUCTIONSPECS>TabSample2B.sketch;3
```

COVER TEXT:

NOTE: USE APPROPRIATE POINT SIZES TO ACHIEVE EFFECT  
SHOWN BELOW. USE XEROX STANDARDS FOR <<8-1/2x11>> BINDER COVERS

FONT

ARTIFICIAL INTELLIGENCE SYSTEMS

XEROX LOOPS

HELVETICA  
BOLD ITALIC

XEROX LOOPS RELEASE NOTES  
XEROX LOOPS REFERENCE MANUAL  
XEROX LOOPS LIBRARY PACKAGES MANUAL

REGULAR or  
TRIUMVIRATE  
BLACK  
ITALIC

KOTO RELEASE

SPINE (3 INCH) :

ARTIFICIAL INTELLIGENCE SYSTEMS

XEROX LOOPS

XEROX LOOPS RELEASE NOTES  
XEROX LOOPS REFERENCE MANUAL  
XEROX LOOPS LIBRARY PACKAGES MANUAL

HELVETICA  
BOLD ITALIC

KOTO RELEASE

TAB TEXT

POINT SIZE:8

FONT: OPTIMA

TABS:

XEROX LOOPS RELEASE NOTES

XEROX LOOPS REFERENCE MANUAL

XEROX LOOPS LIBRARY PACKAGES MANUAL

TABS FOR LOOPS (3" BINDER)

TAB COLOR: PMS GRAY 422-C

BANK 1

XEROX LOOPS RELEASE NOTES

BANK 2

XEROX LOOPS REFERENCE MANUAL

SUBTABS: See following page

BANK 3

XEROX LOOPS LIBRARY PACKAGES MANUAL

## **PACKAGING**

### **DOCUMENTATION SET**

**# of Books in Set:** 4

**Volume Names:** Xerox LOOPS Reference Manual  
Xerox LOOPS Release Notes  
Xerox LOOPS Library Packages Manual  
Xerox LOOPS Users' Packages Manual

### **BINDING**

**Volume Names:** Xerox LOOPS Release Notes  
Xerox LOOPS Reference Manual  
Xerox LOOPS Library Packages Manual

**Book Size:** 8-1/2 X 11

**Binder Size:** 3 inch

**Binding Type:**

**Book Cover Material:**

**Other Binding Specifications:**

**Binder Rings:**

**Sheet Lifters:**

**Inside Pocket:**

**Binder Colors:** PMS Gray 422-C

**Special Instructions:** Bind all manuals in one binder

### **BINDING**

**Volume Names:** Xerox LOOPS Users' Packages Manual

**Book Size:** 8-1/2 X 11

**Binder Size:** 1 inch

**Binding Type:**

**Book Cover Material:**

**Other Binding Specifications:**

**Binder Rings:**

**Sheet Lifters:**

**Inside Pocket:**

***Binder Colors:*** PMS Gray 422-C

***Special Instructions:***

## **FLOPPY DISKS**

***Floppy Disk Packaging:***

***Label Specifications:***

***Special instructions:***

## **PRINTING**

***Printing Method:***

***Paper Weight:***

***Paper Size:***

***Exceptions (e.g., oversize diagrams):***

***Special Instructions:***

## **LOOPS PART NUMBERS**

**3102466**      LOOPS Manual (Reference, Release Notes, Installation Guide)

**3102477**      LOOPS Users' Packages

# SUBTABS FOR LOOPS (3" BINDER)

COLOR OF TABS: WHITE

BANK 1

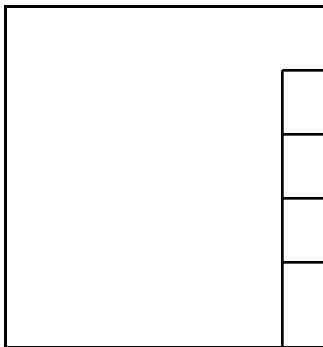
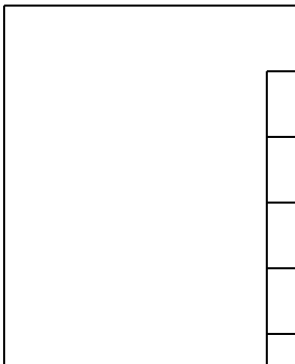


Table of Contents

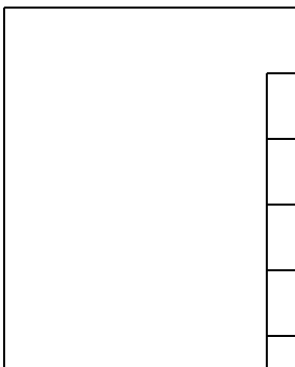
1. Introduction
2. Instances
3. Classes
4. Metaclasses

BANK 2



5. Active Values
6. Methods
7. Message  
Sending Forms
8. Iterative  
Statements
9. Miscellaneous

BANK 3



10. Browsers
11. Errors  
and Breaks
12. Breaking  
and Tracing
13. Editing
14. File Package

Continued on next page



SUBTAB TEXT

POINT SIZE:

FONT:

SUBTABS:

|                                                   |                              |                             |                             |     |
|---------------------------------------------------|------------------------------|-----------------------------|-----------------------------|-----|
| Table of Contents<br>Metaclasses                  | 1. Introduction              | 2. Instances                | 3. Classes                  | 4.  |
| 5. Active Values<br>Miscellaneous                 | 6. Methods                   | 7. Message<br>Sending Forms | 8. Iterative<br>Statements  | 9.  |
| 10. Browsers<br>File Package                      | 11. Errors<br>and Breaks     | 12. Breaking<br>and Tracing | 13. Editing                 | 14. |
| 15. Masterscope<br>User Input/<br>Output Packages | 16. Performance<br>Issues    | 17. Processes               | 18. Reading<br>and Printing | 19. |
| 20. LOOPS Windows                                 | A. Previous<br>Active Values | Glossary                    | Index                       |     |

COVER TEXT:

NOTE: USE APPROPRIATE POINT SIZES TO ACHIEVE EFFECT  
SHOWN BELOW. USE XEROX STANDARDS FOR <<8-1/2x11>> BINDER COVERS

FONT

ARTIFICIAL INTELLIGENCE SYSTEMS  
XEROX LOOPS

HELVETICA  
BOLD ITALIC

USERS' PACKAGES MANUAL  
KOTO RELEASE

REGULAR or  
TRIUMVIRATE  
BLACK  
ITALIC

SPINE (1 INCH) :

ARTIFICIAL INTELLIGENCE SYSTEMS  
XEROX LOOPS  
USERS' PACKAGES MANUAL  
KOTO RELEASE

HELVETICA  
BOLD ITALIC

REGULAR or  
TRIUMVIRATE  
BLACK  
ITALIC

The two big things about Xerox LOOPS are:

- You have to change your underscore to a left arrow.
- You have to change your FONTPROFILE if you want to any examples of menus.

See the conventions file for details.

{ERIS}<Doc>LOOPS>\*>Conventions.tedit

---

## ***Annotated Values and Active Values***

In the previous chapters, IVs, CVs, and their properties have been treated as passive entities without structure. ***Annotated values*** are a way of associating behavior and annotations with variables. In keeping with the object oriented programming style of LOOPS, these annotations are objects. Annotation objects are called ***active values***. When a variable containing an annotated value is accessed, a message is sent to the active value. This mechanism is dual to the notion of messages: messages are a way of telling objects to perform operations, which can change their variables as a side effect; active values are a way of accessing variables, which can send messages as a side effect.

This chapter first describes the structure and implementation of annotated values. Functions for explicitly dealing with annotated values are documented. Then the class `ActiveValue` is introduced and the standard protocol for active values is described. Next, the standard subclasses of `ActiveValue` are explained.

### **5.1. Annotated Values**

LOOPS defines a new INTERLISP data type called `annotatedValue`. Each `annotatedValue` contains a single field. This field contains an object, the annotated value's active value. The standard variable access functions described in previous chapters (`GetValue`, `PutValue`, `GetClassValue`, `PutClassValue`) treat values that are annotated values specially. `GetValue` and `GetClassValue` do not return the annotated value. Instead, they send the contained active value a message, and return the result of that message. Similarly, if the current value of a variable is an annotated value, `PutValue` and `PutClassValue` operate by sending the contained active value a message.

`type? annotatedValue value` [Macro]

Returns true if *value* is an annotated value, false otherwise. This is the standard way to test to see if a value is an annotated value.

`create annotatedValue annotatedValue ← object` [Macro]

Creates a new annotated value with active value *object*. No checking of *object* is performed.

`fetch annotatedValue of value` [Macro]

Returns the active value contained in the annotated value *value*. If *value* is not an annotated value, generates an error.

`replace annotatedValue of value with object` [Macro]

Replaces the active value contained in the annotated value *value* with *object*. If *value* is not an annotated value, generates an error. No checking of *object* is performed.

`←AV av selector . args` [Macro]

←AV is a message sending form that can be used with annotated values. (←AV *av selector . args*)  
→ (← (fetch annotatedValue of *av*) *selector . args*) .

AnnotatedValue [Class]

Sometimes people forget to extract the active value from an annotated value, and they end up trying to use an annotated value as an object. Using the `LispDataType` feature, LOOPS takes care of this for you. Annotated values are considered to belong to the LOOPS class `AnnotatedValue`. If you send a message to an annotated value, the behavior is found in the class `AnnotatedValue`. There, the method for `MessageNotUnderstood` forwards the message off to the contained active value. Similarly, if you attempt to get an IV from an annotated value, the get ends up happening to the wrapped active value.

## 5.2. The Abstract Class ActiveValue

Active values follow a standard protocol that allow them to be used inside of annotated values.

In the description of methods for active values, the arguments *containingObj*, *varName*, *propName*, and *type* are used to describe the variable containing the active value. *type* is one of IV, CV, or NIL : a *type* of IV or NIL indicates that the variable is an instance variable or an instance variable property of *containingObj*; a *type* of CV indicates a class variable or class variable property of *containingObj*. If *propName* is NIL, the variable is either an IV or a CV, otherwise it is an IV or CV property with name *propName*. *containingObj* is the instance or class that contains the variable.

ActiveValue [Abstract class]

The class `ActiveValue` captures the protocol followed by all active value objects. `ActiveValue` is an abstract class, so you cannot make instances of `ActiveValue`. Specializations of `ActiveValue` need to specialize the `GetWrappedValueOnly` and `PutWrappedValueOnly` methods. Methods that you want to specialize include `AVPrintSource`, `GetWrappedValue`, `PutWrappedValue`, `WrappingPrecedence`, and `CopyActiveValue`.

### 5.2.1 Displaying Annotated Values

← *self* AVPrintSource [ActiveValue method]

An annotated value determines how it will print out by sending the AVPrintSource message to the its active value. This message returns a form suitable for use by the INTERLISP function DEFPRINT. The result should be a pair of the form (*item1* . *item2*). *item1* will be printed using PRIN1, and then *item2* will be printed by PRIN2 (see the IRM description of DEFPRINT for more details).

The default method in ActiveValue returns the list

```
("#." $AV className avNames (ivName value propName value ...) (ivName ...)...)
```

which will cause the annotated value to print out as

```
#.($AV className avNames (ivName value propName value ...) (ivName ...)...) .
```

*className* is the name of the class of the active value. *avNames* is a list of names of *self*; the last element of *avNames* is the uid of *self*. The lists (*ivName value propName value ...*) describe the state of the IVs of the active value. Note that the uid of the active value is included in the printed form, so the identity of the active value object can be recovered. In this way, different annotated values can share the same active value, and have this sharing maintained across a dump/load-up.

\$AV *className avNames* . *ivForms* [Special Form]

\$AV is used to reconstruct a dumped annotated value. It returns a new annotated value whose active value is reconstructed from the *avNames* and *ivForms*.

### 5.2.2 Fetching and Replacing Wrapped Values

← *self* GetWrappedValue *containingObj varName propName type* [ActiveValue method]

The GetWrappedValue message provides a way to perform arbitrary actions when a variable is read. When GetValue (or GetClassValue) finds an annotated value in an instance, it does not return the annotated value. Instead, it sends the contained active value the GetWrappedValue message and returns the result of this message.

The default method in ActiveValue sends the message GetWrappedValueOnly to *self*. If this value is an annotated value, it is triggered by sending it the GetWrappedValue message, and the result is returned; otherwise the value is returned with no further processing.

← *self* GetWrappedValueOnly [ActiveValue method]

Returns the immediate "local state" of the variable that is wrapped by the active value *self*. If this local state is a nested active value, it is not triggered. The default implementation causes an error by calling SubclassResponsibility.

```
← self PutWrappedValue containingObj varName
 newValue propName type [ActiveValue method]
```

The PutWrappedValue message provides a way to perform arbitrary actions when a variable is set. When PutValue (or PutClassValue) attempts to replace an annotated value, it instead sends the contained active value the PutWrappedValue message.

The default method in ActiveValue checks to see if the current value is a nested active value by sending the GetWrappedValueOnly message to *self*. If the result is an annotated value, PutWrappedValue forwards the message on the the nested active value; otherwise it sends the message PutWrappedValueOnly to *self* and returns the result.

```
← self PutWrappedValueOnly newValue [ActiveValue method]
```

Replaces the immediate "local state" of the variable that is wrapped by the active value *self*. The current local state is replaced. If the current value is a nested active value, it is not triggered. The default implementation causes an error by calling SubclassResponsibility.

### 5.2.3 Inheriting Active Values

Typical implementations of PutWrappedValue store the new value in the active value. However, if the active value is shared among different instances all these instances would see this change. In particular, if the active value is inherited from the class of the instance, all other instances of the class would see this change. This behavior is usually not desired. The GetWrappedValue method of active values is also free to alter the internal state of the active value, causing the same problem. To get around this problem, the annotated value is first copied, and this copy is stored in the instance. The CopyActiveValue method implements this copying. When GetValue or PutValue finds no local value, it first checks to see if the current value is an annotated value inherited from the class. If it is, it sends CopyActiveValue to the active value, and stores the result in the instance. The put or get then proceeds.

```
← self CopyActiveValue annotatedValue [ActiveValue method]
```

*annotatedValue* is an annotated value that surrounds *self*. CopyActiveValue should return a copy of *annotatedValue*, containing a copy of *self*. It is possible, and in some cases desirable, for an implementation of CopyActiveValue to return *annotatedValue*.

The default behavior returns a new annotated value wrapped around a copy of *self*. IV values of *self* are not copied, the values are shared with the copy, except that IVs of *self* that contain annotated values are copied using the CopyActiveValue message.

### 5.2.4 Adding and Deleting Annotations

```
← self AddActiveValue containingObj varName
 propName type annotatedValue [ActiveValue method]
```

Adds the annotated value *annotatedValue* to the variable specified by *containingObj*, *varName*, *propName*, and *type*. If *annotatedValue* is not specified or is NIL, *annotatedValue* defaults to a newly created annotated value containing the active value *self*. If the variable is already an annotated value, the AddActiveValue method uses the WrappingPrecedence message (below) to

## Annotated Values and Active Values

determine if *annotatedValue* should be nested in the current annotated value or wrapped around it. The method returns *annotatedValue*.

← *self* WrappingPrecedence [ActiveValue method]

Specifies where an annotated value containing *self* should be added to an existing annotated value. T means that this active value must go on the outside of any other annotated values. NIL means it must go on the inside. A number specifies a precedence: active values with larger WrappingPrecedence values go outside ones with smaller WrappingPrecedence values. If two active values have the same (numeric) WrappingPrecedence, the order is not determined. The default implementation of WrappingPrecedence returns 100.

← *self* DeleteActiveValue containingObj varName propName type [ActiveValue method]

Finds the first annotated value on the variable specified by *containingObj*, *varName*, *propName*, and *type* that has *self* as its active value and deletes it from that variable. Returns that annotated value if one was found, NIL otherwise.

← *self* ReplaceActiveValue newVal containingObj  
varName propName type [ActiveValue method]

It is sometimes desirable to replace an annotated value in a variable with some new value. (← *self* ReplaceActiveValue newVal containingObj varName propName type) replaces the annotated value containing *self* in the variable described by *containingObj*, *varName*, *propName*, and *type* with the new value *newVal*.

### 5.2.4 Manipulating Active Values

Some programs need to explicitly test and trigger active values. The following functions can be used to access IVs and CVs without triggering active values.

GetValueOnly object varName propName [Function]

GetValueOnly is the same as GetValue, except that GetValueOnly does not trigger any active values. GetValueOnly returns the immediate value of the variable. If this is not an annotated value, GetValueOnly returns the same value as GetValue. If there is no local value, the inherited value is returned. See also the function GetIVHere.

GetClassValueOnly object varName propName [Function]

GetClassValueOnly is the same as GetClassValue, except that GetClassValueOnly does not trigger any active values. GetClassValueOnly returns the immediate value of the variable. If this is not an annotated value, GetClassValueOnly returns the same value as GetClassValue. *object* can be either an instance or a class.

ObjRealValue object varName value propName type [Macro]

If *value* is not an annotated value returns *value*, otherwise returns the value of (←AV GetWrappedValue object varName propName type). This macro is used by GetValue and



GetClassValue to trigger active values, and can be used by programs that explicitly test for active values.

PutValueOnly *object varName newValue propName* [Function]

PutValueOnly is the same as PutValue, except that PutValueOnly does not trigger any active values. PutValueOnly replaces the immediate value of the variable with *newValue*, even if the old value is an annotated value.

PutClassValueOnly *object varName newValue propName* [Function]

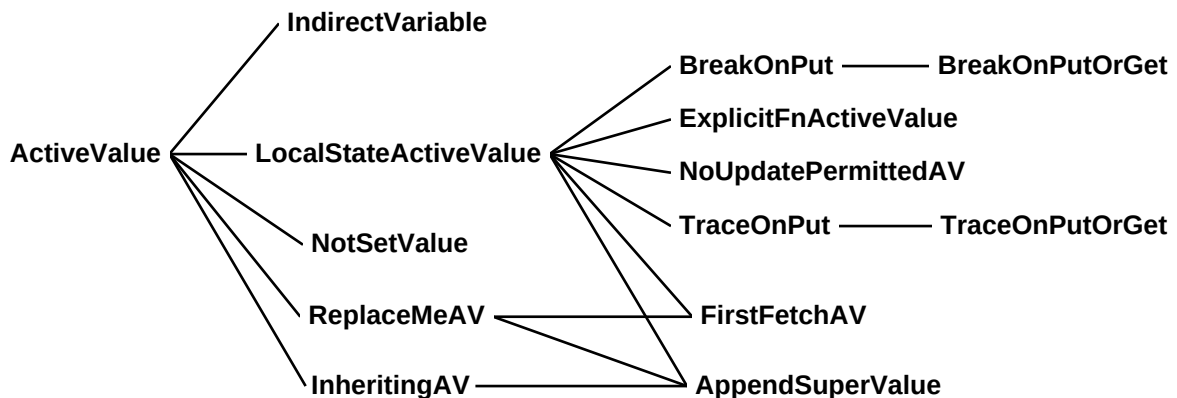
PutClassValueOnly is the same as PutClassValue, except that PutClassValueOnly does not trigger any active values. PutClassValueOnly replaces the immediate value of the variable with *newValue*, even if the old value is an annotated value. *object* can be either an instance or a class.

← *self* HasAV? *av* [ActiveValue method]

Returns true if the active value (or annotated value) *av* is nested inside in the active value *self*.

### 5.3. Specializations of ActiveValue

---



**The ActiveValue Class Hierarchy**

---

#### 5.3.1 NotSetValue and Variable Inheritance

NotSetValue [Variable]

LOOPS uses annotated values to trigger IV inheritance. When an instance is created, its IVs are initialized to contain (the value of) NotSetValue. NotSetValue is an annotated value whose active value is the prototype instance of the class NotSetValue. The class NotSetValue specializes the

## Annotated Values and Active Values

default `ActiveValue` protocol to trigger IV inheritance. In this way `GetValue` does not need to do any special check to see if a value needs to be inherited — all it needs to do is see if the value is an annotated value. Note that `GetValueOnly` does need to do a special check for `NotSetValue`, but see the function `GetIVHere`.

`NotSetValue form`

[Macro]

Returns true if *form* evaluates to `NotSetValue`, otherwise false.  $(\text{NotSetValue } form) \rightarrow (\text{EQ } form \text{ 'NotSetValue})$ . This is the approved way of testing a value to see if it is `NotSetValue`.

$\leftarrow self \text{ AVPrintSource}$

[NotSetValue method]

Returns the pair  $(\text{"#."} \text{ . NotSetValue})$ . This causes (the value of) `NotSetValue` to print out as `#.NotSetValue`. This will be read in as the value of the variable `NotSetValue`.

$\leftarrow self \text{ GetWrappedValue containingObj varName propName type}$

[NotSetValue method]

If *type* is `NIL` or `IV`, this evaluates  $(\leftarrow containingObj \text{ IVValueMissing varName propName 'GetValue})$  and returns the result; if *type* is `CV`, evaluates  $(\leftarrow class \text{ CVValueMissing varName propName 'GetValue})$  (where *class* is the class of *containingObj* if *containingObj* is an instance, else *containingObj* if it is a class) and returns the result; otherwise an error is generated. See the methods `IVValueMissing` and `CVValueMissing` on the class `Object`.

$\leftarrow self \text{ PutWrappedValue containingObj varName}$

*newValue propName type*

[NotSetValue method]

If *type* is `NIL` or `IV`, this evaluates  $(\leftarrow containingObj \text{ IVValueMissing varName propName 'PutValue newValue})$  and returns the result; if *type* is `CV`, evaluates  $(\leftarrow class \text{ CVValueMissing varName propName 'PutValue newValue})$  (where *class* is the class of *containingObj* if *containingObj* is an instance, else *containingObj* if it is a class) and returns the result; otherwise an error is generated. See the methods `IVValueMissing` and `CVValueMissing` on the class `Object`.

$\leftarrow self \text{ CopyActiveValue annotatedValue}$

[NotSetValue method]

Returns `#.NotSetValue`. There is only one `NotSetValue`.

$\leftarrow self \text{ WrappingPrecedence}$

[NotSetValue method]

Returns `NIL`. `#.NotSetValue` must always be on the inside of any sequence of nested active values.

`GetIVHere object varName propName`

[Function]

If *propName* is `NIL` and there is a local value for the IV *varName* in the instance *object*, that value is returned. If *propName* is not `NIL` and there is a local value for the IV property *propName* of the IV *varName* in the instance *object*, that value is returned. Otherwise, if *propName* is `NIL` `GetIVHere` returns `#.NotSetValue`, and if *propName* is not `NIL` `GetIVHere` returns (the value of) `NoValueFound`.

---

## LOOPS8

GetCVHere *object varName propName* [Function]

*object* must be a class. Returns the value of the class variable that is found in the class *object*. If none is found, then returns `#.NotSetValue`.

GetClassIV *class varName propName* [Function]

Returns the default value or property value of the instance variable *varName* in the class *class*.

PutClassIV *class varName newValue propName* [Function]

Stores *newValue* as the default value or property value of the instance variable *varName* in the class *class*. If *varName* is not already local to the class, this will cause an error. Returns *newValue*.

### 5.3.2. Indirect Variables

In some applications it is important to be able to access values indirectly from other instances. For example, Steele [Steele80] has recommended this as an approach for implementing equality constraints.

IndirectVariable [Class]

Mumble.

### 5.3.3. ReplaceMeAV

The active value mixin `ReplaceMeAV` can be used when an active value should be replaced when a variable is first set.

ReplaceMeAV [Abstract class]

Mumble.

### 5.3.4. LocalStateActiveValue

Many kinds of active values explicitly store the "real" value of the variable in an IV of the active value.

LocalStateActiveValue [Abstract class]

Mumble.

### 5.3.5. InheritingAV

Some kinds of active values want to compute a value based on what would have been inherited if the active value had not been present. For example, it might be desired to append items onto an inherited value (see the class `AppendSuperValue`).

InheritingAV [Abstract class]

Mumble.

---

## Annotated Values and Active Values

### 5.3.6. FirstFetchAV

Mumble.

FirstFetchAV [Class]

Mumble.

### 5.3.7. Breaking and Tracing Variable Access

Mumble.

BreakOnPut [Class]

Mumble.

BreakOnPutOrGet [Class]

Mumble.

TraceOnPut [Class]

Mumble.

TraceOnPutOrGet [Class]

Mumble.

UnBreakIt *self varName propName type* [Class]

Mumble.

### 5.3.8. NoUpdatePermittedAV

The active value class NoUpdatePermittedAV can be used to prevent a value from being updated.

NoUpdatePermittedAV [Class]

Mumble.

### 5.3.9. AppendSuperValue

The active value class AppendSuperValue can be used to append data to inherited values.

AppendSuperValue [Class]

Mumble.

### 5.3.10. ExplicitFnActiveValue

ExplicitFnActiveValue

[Class]

ExplicitFnActiveValue explicitly store functions that will be triggered when the variable is fetched or replaced. They have three IVs: *localState*, *getFn*, and *putFn*. The *localState* is the "real" value of the variable (possibly a nested active value), the *getFn* and *putFn* are names of functions that are applied with standard arguments by the *GetWrappedValue* and *PutWrappedValue* methods. The *getFn* and *putFn* are called with arguments *containingObj*, *varName*, *oldOrNewValue*, *propName*, *activeValue*, and *type*. ExplicitFnActiveValue active values print out as #. (\$A *localState* *getFn* *putFn*), where the *localState*, *getFn*, and *putFn* are the values of the corresponding IVs of the active value.

## 5.4. Compatibility with older versions

The following existed in older versions of LOOPS, which had a different implementation of active values. They are provided for compatibility purposes only. New programs should not use them. They are not fully supported, and will not exist in future releases. The current implementations of these use the new active values. They are fully compatible with the older versions except where noted.

### 5.4.1. Old Style Active Values

LOOPS used to combine the notion of annotated value and active value. Variable annotations were instances of the INTERLISP datatype *activeValue*.

*activeValue*

[Record]

In this version of LOOPS, the record *activeValue* is an access record that converts the three fields of the old active values to appropriate functions for accessing annotated values. Forms like (*type?* *activeValue* *form*) and (*fetch* *localState* *of* *activeValue*) will do the right thing. Reading in old style active values automatically converts them to annotated values wrapping an instance of the class *ExplicitFnActiveValue*.

*GetLocalState* *av self varName propName type*

[Function]

Works just like in the old LOOPS.

*PutLocalState* *av newValue self varName propName type*

[Function]

Works just like in the old LOOPS.

*GetLocalStateOnly* *av*

[Function]

Works just like in the old LOOPS.

---

### Annotated Values and Active Values

`PutLocalStateOnly av newValue` [Function]

Works just like in the old LOOPS.

`ReplaceActiveValue av newVal self varName propName type` [Function]

Works just like in the old LOOPS.

`MakeActiveValue self varOrSelector newGetFn newPutFn  
newLocalSt propName type` [Function]

Works just like in the old LOOPS, except that the interpretation of *newLocalSt* is different. `MakeActiveValue` ignores the value of *newLocalSt*, and always creates a new active value. This is the behavior that the old `MakeActiveValue` produced when *newLocalSt* was `Embed`.

### 5.4.2. GetFns and PutFns

`DefAVP fnName putFlg` [Function]

Works just like in the old LOOPS.

`NoUpdatePermitted self varname oldOrNewValue propName activeValue type` [Function]

Works just like in the old LOOPS.

`FirstFetch self varname oldOrNewValue propName activeValue type` [Function]

Works just like in the old LOOPS.

`GetIndirect self varname oldOrNewValue propName activeValue type` [Function]

Works just like in the old LOOPS.

`PutIndirect self varname oldOrNewValue propName activeValue type` [Function]

Works just like in the old LOOPS.

`ReplaceMe self varname oldOrNewValue propName activeValue type` [Function]

Works just like in the old LOOPS.

`AtCreation self varname oldOrNewValue propName activeValue type` [Function]

No longer works. Instead, you can use either the `FirstFetch` function, or the `:initForm` property of IVs.

## 5.5. Summary of Variable Access Functions

The following tables summarizes the available functions for variable access.

|                       | Inherit/Trigger                | Inherit/Don't Trigger                  | Don't Inherit/Don't Trigger  |
|-----------------------|--------------------------------|----------------------------------------|------------------------------|
| <i>from instances</i> |                                |                                        |                              |
| IV                    | GetValue<br>PutValue           | GetValueOnly<br>PutValueOnly           | GetIVHere                    |
| CV                    | GetClassValue<br>PutClassValue | GetClassValueOnly<br>PutClassValueOnly | <n.a.><br><n.a.>             |
| <i>from classes</i>   |                                |                                        |                              |
| IV                    | <n.a.><br><n.a.>               | GetClassIV<br>PutCIVHere               | GetClassIVHere<br>PutClassIV |
| CV                    | GetClassValue<br>PutClassValue | GetClassValueOnly<br>PutClassValueOnly | GetCVHere<br>PutCVHere       |

## Tailoring MasterScope

### Extending analysis of functions

MasterScope maintains a number of tables describing how (analyzed) functions relate to other objects. The template keywords **TEST**, **PROP**, **FUNCTION**, etc. correspond to some of these tables. For a number of applications, the user would like to be able to define new template words and the database tables that go along with them.

**(ADDTEMPLATEWORD WORD)** [Function]

defines a new table to hold a new MasterScope relation. The name of the table will be **WORD**, and **WORD** can be used in function templates. This is a new function.

Functions are also provided to allow the user to access these new tables inside of a MasterScope command.

**(MSADDRELATION RELATION TABLES)** [Function]

defines a new relation for MasterScopes parser and command interpreter. For example, **(MSADDRELATION ' (FETCH FETCHES FETCHING FETCHED) )** could have been used to define the **FETCH** relation.

**RELATION** is a list of **ROOT PRESENT PARTICIPLE** and **PAST** conjugations of the new relation. **TABLES** is a list of MasterScope database tables that will be **UNIONed** to compute the new relation. (If **TABLES** is an atom it will be coerced to a list containing that atom. If the tables do not already exist, they will automatically be created by **ADDTEMPLATEWORD**). **TABLES** defaults to the **ROOT** of the relation.

**(MSADDMODIFIER RELATION MODIFIERS TABLES)** [Function]

defines a new modifier for the given relation. For example, the phrase **SET FREE** could have been defined by **(MSADDMODIFIER 'SET 'FREE ' (SETFREE) )**.

**RELATION** is a known MasterScope relation (either built-in or defined by **MSADDRELATION** above). **MODIFIERS** is a list of equivalent modifiers. **TABLES** is a list of MasterScope database tables that should be **UNIONed** to compute the new, modified relation. (If any of the tables do not exist, they will be created by **ADDTEMPLATEWORD**). This is a new function.

**(MSADDTYPE TYPE TABLES HOWUSED SYNONYMS)** [Function]



tells MasterScope what it means to use an object of a given type. The phrase **USE THE FIELD** could be defined by **(MSADDDTYPE 'FIELD' (FETCH REPLACE) ' (USE FETCH REPLACE) )**.

**TYPE** is the type of the object being described. The word **TYPE** can then be used in MasterScope commands. **TABLES** indicates how the relation **USE THE <TYPE> . . .** is defined. **HOWUSED** is a list of verb describing how the type can be used. The above example not only lets you use the phrase **USE THE FIELD**, but also the phrases **FETCH THE FIELD** and **REPLACEUSE THE FIELD**. The default value of **HOWUSED** is **(USE)**. Finally, **SYNONYMS** is a list of synonyms for **TYPE**.

In addition to these new functions there are a number of other ways to tailor MasterScope.

Some MasterScope templates, for example **(IF expression template template )** can compute the template to be used. These expressions can access the current form via the free variable **EXPR**. The free variable **PARENT** can be used to access the expression that contains the current expression. This has existed all along, but has not been documented.

#### **ANALYZEUSERFNS**

[Variable]

is a list each of whose elements is a function that will be **APPLY\***ed to the name, definition, and the results of the MasterScope analysis of a function whenever it is analyzed. The result of each application becomes the new result of the MasterScope analysis.

The results of the MasterScope analysis is an ALIST associating relations (**BIND**, **CALL**, etc) with the corresponding data for the function. This can be used to compute relations that are determined by some global context. This has existed all along, but has not been documented.

#### **(SETSYNONYM PHRASE MEANING -)**

[Function]

defines a new synonym for MasterScope's parser. Both **MEANING** and **PHRASE** are lists of words; anywhre **PHRASE** is seen in a command, **MEANING** will be substituted. For example, **(SETSYNONYM 'GLOBALS' (VARS IN GLOBALVARS OR @(GETPROP X 'GLOBALVAR)))** would allow the user to refer with the single word **GLOBALS** to the set of variables which are either in **GLOBALVARS** or have a **GLOBALVAR** property.

**SETSYNONYM** includes a small pattern match ability. A **&** in **PHRASE** will match any word; the word will be substituted for **N** in **MEANING** where **N** is the number of **&**'s which have been matched. For example, **(SETSYNONYM ' (FOO & &) ' (IN 1 OR ON 2))** will take **FOO FIE FUM** into **IN FIE OR ON FUM**.

This is an old function, but the documentation (IRM p 13.20) was incomplete—specifically the pattern match stuff was not mentioned.

## **Extending MasterScope commands**

**DESCRIBELST**

[Variable]

is a list each of whose elements is a list containing a descriptive string and a form. The form is evaluated (it can refer to the name of the function being described by the free variable **FN**); if it returns a non-**NIL** value, the description string is printed followed by the value. If the value is a list, its elements are printed with commas between them. For example, the entry ("**types:** " **(GETRELATION FN ' (USE TYPE) T)** would include a listing of the types used by each function. From the IRM, p 13.7.

**MSCHECKFNS**

[Variable]

is a list of functions that will be used to extend the **CHECK** MasterScope command. The **CHECK** command will **APPLY\*** each function on **MSCHECKFNS** to the list of files being checked. This is new.

**Analyzing new types of objects**

*This doesn't work yet.*

**(MSADDANALYZE PLURAL SINGULAR ANALYZEFN RELATIONS)**

[Function]

defines a new type for MasterScope analysis. For example, **(MSADDANALYZE 'CLASSES 'CLASS 'AnalyzeClass ??)** will let you execute MasterScope commands like **. ANALYZE ANY CLASS ON 'MYFILE** and **. WHAT CLASS DEFINES THE IV foo**. The function **ANALYZEFN** should be a function of two arguments, the item name and the flag **REANALYZE?**, and should return an **ALIST** associating relation names to corresponding data for the object. This is a new function.

**Need to do**

**(DUMPDATABASE)** needs to store out all the new words, relation, etc.

Analyzing new types of things needs to store the info someplace, and **ERASE** needs to be able to find it.

Need a **FILEPKGTYPE** for the MasterScope words.

Proposal for Loops manual contents.

This ordering is based on the Interlisp Reference manual. Numbers in `[[...]]`'s indicate the corresponding parts in the (new) IRM. Other comments are enclosed in `{}`'s. I envision a manual in a single binder, with large tabs marking the IRM volume separation, and smaller tabs for each chapter.

*[[Volume 1 -- Language]]*

TOC *[[TOC]]*

Introduction *[[1]]*

Classes and Instances

MessageSending

The Golden Braid

Classes *[[2]]*

*{Defines the msg protocol for Classes}*

Creating instances

New msg

NewWithValues msg

\_New msg sending form

Destroying instances

DestroyInstance msg

Destroying a class

Destroy msg

Destroy! msg

Changing a class

Add msg

Delete msg

Put msg

Rename msg

ReplaceSupers msg

SetName msg

Accessing parts of a class

ListAttribute msg

ListAttribute! msg

Enumerating parts of classes

AllInstances msg

AllInstances! msg

SubClasses msg

Dealing with Methods

SpecializeMethod msg

CopyMethod msg

DefMethod msg

MoveMethod msg

MethodDoc msg

## Misc

NewClass msg

Subclass msg

Prototype msg

See also \_Proto

## Instances [[3]]

*{Defines the protocol for objects}*

### Getting the class of an instance

Function Class

Class msg

ClassName msg

Classes for lisp entities -- LispClassTable

### Accessing variables

GetValue, PutValue, GetIVHere

GetClassValue, PutClassValue, GetClassIVHere

@, @\*, \_@

problems with embedded "."s in IV names

GetIVValue msg if not an instance

NoValueFound

CHANGETRAN

IVMissing msg

CVMissing msg

### Creating instances

NewInstance method

### Naming instances

SetName msg

UnSetName msg

Rename msg

ErrorOnNameConflict

### Changing an instance

AddIV msg

DeleteIV msg

### Destroying instances

Destroy message

### Copying Instances

CopyShallow message

CopyDeep message

## MetaClasses [[4]]

*{Defines the msg protocol for MetaClasses}*

DestroyClass msg

- MetaClass
- AbstractClass
- AnnotatedValues [[5]]
  - NotSetValue
- Methods [[6]]
  - Method objects
    - Categories
  - Method functions
    - LAMBDA word Method
    - &OPTIONAL arguments
    - Advising inherited methods
      - \_Super
      - \_Super?
      - \_SuperFringe
      - DoMethod
      - ApplyMethod
    - Functions
      - ClassNameOfMethodOwner, SelectorOfMethodBeingCompiled, ArgsOfMethodBeingCompiled
  - Defining new methods
    - DefaultComment
    - SubclassResponsibility
- Conditionals and Iterative Statements [[9]]
  - Date type predicates
    - Object?
    - Class?
    - Instance?
    - AnnotatedValue?
  - Other predicates
    - HasIV, HasIV!, HasCV
    - Understands
    - InstOf, InstOf!
  - Iterative statements
    - in-supers-of
- Message sending forms [[10]]
  - List all message sending forms (even if described elsewhere)
    - \_
    - \_!
    - \_IV
    - \_Try
    - \_Proto
    - \_Super
    - \_Super?
    - \_SuperFringe
    - \_New
    - \_AV
  - Looking up methods

FetchMethod  
FetchMethodOrHelp

## Miscellaneous *[[12]]*

System version information  
Pattern match function MatchDescr  
\*FEATURES\*  
DELAASSOC

*[[Volume 2 -- Environment]]*

## Browsing and Exec Level Commands *[[13]]*

UNDOable versions of Loops fns (like PutValue) if typed at top level?

LASTWORD set by DefineMethods

LASTCLASS set by DefineClass

### Loops Icon

Left button menu  
Middle button menu  
Right button menu  
Background menu command?

### Browsers, in general

Interaction with GRAPHZOOM

MaxLatticeHeight

MaxLatticeWidth

Shift selecting from the background of a LatticeBrowser will COPYINSERT the graph.

### Class Browsers

PPDefault

UpdateClassBrowsers?

### File Browsers

### Other Browsers

### Top level commands

DefineClass  
LASTWORD  
USERWORDS  
DefineMethod

## Errors and Breaks *[[14]]*

Function HELPCHECK

LoopsDebugFlg

LoopsDebugFlg

ErrorOnNameConflict

IVMissing

IVValueMissing

MessageNotUnderstood

## Breaking and Tracing *[[15]]*

Breaking and Tracing IV access

Breaking and Tracing methods

## Gauges

### Editing [[16]]

EDITDEF

When do changes take effect?

Copying an AV inside of DEdit really adds a pointer to the same AV!

Edit msg

MakeEditSource msg

InstallEditSource msg

### File Package [[17]]

#### Loading Loops files

Effect of LDFLG = PROP, ALLPROP, ect

UNDOing

Selective loading of classes, etc from a file.

#### Filepackage coms

CLASSES

METHODS

INSTANCES

THESE-INSTANCES

#### Noticing changes

ObjectModified message

#### Moving and adding

MoveToFile msg

MoveToFile! msg

#### Saving instances

MakeFileSource message

DontSave IV property

SaveInstance message

SaveInstance? message

### Compiling [[18]]

Macros for GetValue, PutValue

Referring to Loops objects in compiled code

### Masterscope [[19]]

### DWIM [[20]]

No method, tries to correct the spelling

No method, but a fn=> use the fn. How about other way?

### Performance Issues [[22]]

#### Garbage Collection

Space

Speed

#### Caching

ClearAllCaches

Evals all forms on the global var ClearAllCaches

Method lookup caches

local cache  
 global cache hit rate of 97.5% in our tests

#### IV Lookup caches

local cache  
 global cache hit rate of 98% in our tests

#### Processes [[23]]

\_Process, \_Process!

#### [[Volume 3 -- Input/Output]]

#### Reading and Printing [[25]]

The # read macro character

How Classes print

#.(⌘ className) and the function ⌘

Class.FileOut

#.(⌘C className) and the function ⌘C

How Instances print

Object.PrintOn

Object.PP

PPObj

ObjectAlwaysPPFlag

ObjectDontPPFlag

Interaction with SYSPRETTYFLG

How AnnotatedValues print

DefaultActiveValueClassName

UIDs

When assigned

Functions HasUID?, UIDP, UID, GetObjFromUID, MapObjectUID

#### User Input/Output Packages [[26]]

Inspector

Extensions to ?=

method doc in PROMPTWINDOW

select selector from menu

doesn't work in the DEdit menu

#### Loops windows [[28]]

The default menu available on Loops windows is compatible with Lisp. It even includes a "Hardcopy" item that works correctly with LatticeBrowsers.

Methods Window.SetRegion and Window.SetOuterRegion

Interaction with ATTACHEDWINDOWS

#### Index [[Index]]

#### [[Loops Library Packages Manual]]

#### Virtual Copies



## Gauges

*[[Release Notes]]*

### Installation

Variables `LOOPSDIRECTORY`, `OptionalLispuserFiles`, and `LoadLoopsForms`.

Maybe even `LOOPUSERSDIRECTORIES`?

The function `LOADLOOPS`

The file `LOOPSSITE`

Just load the file `LOOPS`

Initial screen setup

### Customer Support

### Differences from old Loops

KnowledgeBases no longer a part of Loops

Functions `ReadLeafObj`, `AllGlobalNames`, `RememberName`, and `GlobalName` no longer exist

Variables `WritingSummaryFlg`, `WritingLayerFlg`, `LeafInstanceFlg`, `FirstEnvFlg`, `OpenKBFiles`, `DefaultKBName`, `CurrentEnvironment`, `CurrentNameTable`, and `CurrentUIDTable` no longer exist

Macro `Modified` no longer exists

Rules no longer a part of Loops

Rules are not distributed. If they are needed, they can be loaded by first recompiling them and then loading the file `LOOPSRULES-ROOT`. This replaces a few functions and methods in the standard system and then load the rules files.

Code cleaned up

Variables `VarNameIndexes`, `PrintStatusWindow`, and `TTY` no longer exist

Functions `DC`, `DE`, `UE`, `EM`, `EI`, `EC`, and `FILE` no longer exist

Macro `@@` no longer exists. Change `(@@ foo)` to `(@ ::foo)`, or even `(@ foo)`

Usermacros `PU`, `UE`, and `EU` no longer exist

The argument that controls updating in the window methods

`SetOuterRegion`, `SetRegion`, `Shape`, and `Shape1` have all been changed to `noUpdateFlg`. They used to be a mixture of `updateFlg` and `dontUpdateFlg`.

The interpretation of the `left`, `bottom`, `width`, and `height` IVs in the class `Window` is now the same as Lisp windows: `left` and `bottom` refer to the lower left corner of the outside of the window, `width` and `height` refer to the outside dimensions of the window (including title).

The file `LoopsMixin` has been deleted. The classes `DatedObject`, `IndirectObj`, `Node`, `Perspective`, `NamedObject`, `GlobalNamedObject`, `TextItem`, `VarLength`, `StrucMeta`, `ListMetaClass`, `TempClass`, and `Template` no longer exist.

Old messages `List` and `List!` are now called `ListAttribute` and `ListAttribute!`. This change was required for future migration to `CommonLoops`. The old methods for `List` and `List!` are still available in the `LOOPSBACKWARDS` file.

The functions `DebugLoops`, `LOOPSDIR`, `i/d`, and `TESTLOOPS` have been removed from the system.

The functions `CheckDestroyedObjects` and `RemoveClassDef` have been removed from the system.

## I/O

Support for reading in old style read macro (like `$(localState getFn putFn)` or `#mMumble`) is available in the file `LOOPSBACKWARDS`.

UIDs are no longer strings. They are CONSES of session-id's and uid numbers.

## Active Values

Most of the old functions for active values have been moved to `LOOPSBACKWARDS`. The functions `GetLocalState`, `PutLocalState`, `GetLocalStateOnly`, `PutLocalStateOnly`, `ReplaceActiveValue`, `MakeActiveValue`, and `DefAVP` are still around, and seem to work. The exception is `MakeActiveValue` — it now always `EMBEDS`.

Functions `GetActiveValueGetFn`, `GetActiveValuePutFn`, `GetActiveValueLocalState`, `PtcieauGtn`, `alueGetFn`, etc no longer exist. Replacements are available in the file `LOOPSBACKWARDS`.

`NotSetValue` is no longer ?—it is now an active value. This may require changes to any code that directly referred to ?.

## Virtual Copies

The messages `MakeCopy99`, `MakeCopyActiveValue99`, `MakeCopyList99`, and `MakeCopyObject99` in `VirtualCopies` have been renamed `\Internal/MakeCopy`, `\Internal/MakeCopyActiveValue`, etc

Method properties no longer exist

Function `PushNewValue` no longer defined. Instead, use `CHANGETRAN`.

Can't load new Loops files into old Loops.

## Conversion of old Loops code to new Loops

Old code will need to be recompiled.

File `LOOPSBACKWARDS`

Function `ConvertLoopsFiles`

## Future Directions

Proposal for Loops manual contents.

This ordering is based on the Interlisp Reference manual. Numbers in [[]]'s indicate the corresponding sections in the (new) IRM.

[[Volume 1 -- Language]]

Introduction [[1]]

Classes [[2]]

    The hierarchy

    Tofu

    Object

    Class

    LispClassTable

Instances [[3]]

MetaClasses [[4]]

    The Golden Braid

    MetaClass

    AbstractClass

AnnotatedValues [[5]]

    NotSetValue

Methods [[6]]

    Categories

    DefaultComment

    SubclassResponsibility

Conditionals and Iterative Statements [[9]]

    Date type predicates

        Object?

        Class?

        Instance?

        (*type? annotatedValue form*)

    Other predicates

        HasIV, HasIV!, HasCV

        Understands

        InstOf, InstOf!

    Iterative statements

        in-supers-of

Message sending forms [[10]]

    List all message sending forms (even if described elsewhere)

    Looking up methods

        FetchMethod

        FetchMethodOrHelp

Miscellaneous [[12]]

System version information

Pattern match functions

\*FEATURES\*

[[Volume 2 -- Environment]]

Browsing and Exec Level Commands [[13]]

UNDOable version of PutValue?

LASTWORD set by DefineMethods

LASTCLASS set by DefineClass

Loops Icon

Browsers, in general

MaxLatticeHeight

MaxLatticeWidth

Class Browsers

File Browsers

Other Browsers

Top level commands

DefineClass

DefineMethod

Errors and Breaks [[14]]

LoopsDebugFlg

ErrorOnNameConflict

IVMissing

IVValueMissing

MessageNotUnderstood

Breaking and Tracing [[15]]

Breaking and Tracing IV access

Breaking and Tracing methods

Gauges

Editing [[16]]

File Package [[17]]

Compiling [[18]]

Macros for GetValue, PutValue

Referring to Loops objects in compiled code

Masterscope [[19]]

DWIM [[20]]

No method, but a fn=> use the fn. How about other way?

Performance Issues [[22]]

Garbage Collection

Space

Speed

Caching

ClearAllCaches  
 Method lookup caches  
 IV Lookup caches

Processes [[23]]

\_Process, \_Process!

[[Volume 3 -- Input/Output]]

Reading and Printing [[25]]

The # read macro character

How Classes print

How Instances print

ObjectAlwaysPPFlag

ObjectDontPPFlag

How AnnotatedValues print

DefaultActiveValueClassName

UIDs

When assigned

User Input/Output Packages [[26]]

Inspector

Extensions to ?=

Loops windows [[28]]

[[Loops Library Packages Manual]]

Virtual Copies [[no chapter]]

Composite Objects [[no chapter]]

[[Release Notes]]

Installation

Customer Support

Conversion From old Loops

Future Directions

**XEROX LOOPS**

**RELEASE NOTES**

**REFERENCE MANUAL**

**LIBRARY PACKAGES MANUAL**

**XEROX**

XEROX LOOPS RELEASE NOTES

XEROX LOOPS REFERENCE MANUAL

XEROX LOOPS LIBRARY PACKAGES MANUAL

610E15980

Lyric/Medley Release

July 1988

The information in this document is subject to change without notice and should not be construed as a commitment by Xerox Corporation. While every effort has been made to ensure the accuracy of this document, Xerox Corporation assumes no responsibility for any errors that may appear.

Copyright © 1988 by Xerox Corporation.

Xerox LOOPS, Xerox Lisp, and Xerox Common Lisp are trademarks.

All rights reserved.

"Copyright protection claimed includes all forms and matters of copyrightable material and information now allowed by statutory or judicial law or hereinafter granted, including, without limitation, material generated from the software programs which are displayed on the screen, such as icons, screen display looks, etc."

This manual is set in Modern and Terminal typefaces with text written and formatted on Xerox Artificial Intelligence workstations. Xerox laser printers were used to produce text masters.

COVER TEXT:

NOTE: USE APPROPRIATE POINT SIZES TO ACHIEVE EFFECT  
SHOWN BELOW. USE XEROX STANDARDS FOR <<8-1/2x11>> BINDER COVERS

FONT

ARTIFICIAL INTELLIGENCE SYSTEMS  
XEROX LOOPS

HELVETICA  
BOLD ITALIC

XEROX LOOPS RELEASE NOTES  
XEROX LOOPS REFERENCE MANUAL  
XEROX LOOPS LIBRARY PACKAGES MANUAL  
  
LYRIC/MEDLEY RELEASE

REGULAR or  
TRIUMVIRATE  
BLACK  
ITALIC

Part Number 610E15980

SPINE (3 INCH) :

ARTIFICIAL INTELLIGENCE SYSTEMS  
  
XEROX LOOPS  
  
XEROX LOOPS RELEASE NOTES  
XEROX LOOPS REFERENCE MANUAL  
XEROX LOOPS LIBRARY PACKAGES MANUAL  
  
LYRIC/MEDLEY RELEASE

HELVETICA  
BOLD ITALIC

REGULAR or  
TRIUMVIRATE  
BLACK  
ITALIC



# ERRATA FOR THE LYRIC / MEDLEY RELEASE OF XEROX LOOPS

---

Please note the following corrections to the Lyric/Medley release of the manual and software. These descriptions and workarounds supplement the Lyric/Medley *Xerox LOOPS Release Notes*.

---

## Notes and Cautions

---

### Documentation comments in examples

---

Documentation props are now strings, as mentioned in the Lyric/Medley version of the *Xerox LOOPS Release Notes*. However, many of the manual's examples incorrectly show comments being used for documentation on methods, and instance and class variables.

### Conversion of instance files

---

We have encountered a case where instance files from Koto LOOPS failed to load after conversion under Lyric/Medley LOOPS. This occurs when instances refer to each other (their IVs contain one another).

This type of instance file must be written out in the INTERLISP readtable, since only that readtable handles the "hash dot" reader macro used to write out a reference to an instance. After conversion of such files, place a MAKEFILE-ENVIRONMENT property on the file that will cause it to be written and read in the INTERLISP readtable.

### GetIt for missing IV props

---

There is an inconsistency in **GetIt**'s behavior when retrieval of missing instance variable properties is attempted. If the object being retrieved from is an *instance*, this returns the value of **NoValueFound**, and triggers any active values. However, if the object being retrieved from is a *class*, **GetIt** returns the value of **NotSetValue**, and does not trigger active values. A workaround would be to check for both **NoValueFound** and **NotSetValue** as return values from **GetIt**.

## LOOPS Rules

---

The User's Module "Rules" has changed to track differences between Koto and Lyric/Medley LOOPS. In Koto LOOPS, one could create RuleSets that were not methods; this is no longer possible. RuleSets can no longer reside "bare" in a FNS definition.

Several things follow from this change:

- **RunRS** no longer works.
- **DefRSM** is now the only way to create new RuleSets. The documented **New** method for the class **RuleSet** is used internally by **DefRSM**; software should no longer specialize or depend on it.
- The RuleSet method **CopyRules** no longer works. RuleSets can be copied using **CopyMethod**.

## Converting Koto RuleSets

---

Converting older RuleSets to run in the Lyric/Medley release of Xerox LOOPS is a two-step process. The first step takes place in the Koto release of LOOPS; the second step occurs in the Lyric/Medley release of LOOPS.

While still running the Koto release of LOOPS, you must first pass preKoto RuleSets through the converter in the LOOPSBACKWARDS User's Module. This will upgrade the RuleSets to run in Koto LOOPS. RuleSets which reside in FNS must also be moved into methods at this time. A final ready-to-convert version of the RuleSet files should then be made.

After starting up the Lyric/Medley release of Xerox LOOPS any previously prepared Koto RuleSets can be passed through the converter in the CONVERSION-AIDS User's Module. Some types of Koto converted Buttriss RuleSets will print out a message during conversion; these must then be rule-compiled, i.e. translated from RuleSet to executable code by sending the RuleSet the **RE** message and exiting the rule editor with **OK&LispCompile**.

After these steps all RuleSet formats will be correctly updated and the fully converted files can be made.

[This page intentionally left blank]

# A. ACTIVE VALUES IN BUTTRESS LOOPS

---

In the Buttress version of LOOPS, the concept of active values was implemented differently. The current **ExplicitFnActiveValue** acts very much like the old active values, and is used to provide compatibility with existing Xerox LOOPS code. Most of the functions described in this chapter are found only in the LOOPSBACKWARDS user package, work as they did in the Buttress version of LOOPS, and should be used only to bring existing code into the current system.

Descriptions of the functionality in this appendix are written in terms of the new **ActiveValues** wherever possible.

The active value/annotated value system discussed in Chapter 8, Active Values, is a new implementation. Programs developed using the Buttress activeValue system automatically convert into the new system when loaded, using the **ExplicitFnActiveValue** capability described in Chapter 8, Active Values, and in this appendix.

Note: The following functions and records are maintained for compatibility purposes only; they are not fully supported and may not exist in future Xerox LOOPS releases. Programs that use these records and functions should be changed. The LOOPSBACKWARDS user package must be loaded for these functions to work.

---

## A.1 Buttress System of ActiveValues

---

The Buttress LOOPS implementation combined the notions of annotated value and active value. To annotate a variable, the value was replaced with an instance of an Interlisp-D data type called activeValue, but there were no LOOPS classes with similar names and functions as there are now.

| activeValue  |                                                                                                                                                                                                                                       | [Record]                                                                                                                 |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| Purpose:     | Buttress implementation of the active values concept. Specifically, the Lisp data type equivalent to the present annotatedValue.                                                                                                      |                                                                                                                          |
| Behavior:    | An activeValue placed as the value of a variable invoked evaluation of code on access attempts rather than just returning a stored value.                                                                                             |                                                                                                                          |
| Field Names: | <b>localState</b>                                                                                                                                                                                                                     | A place for data storage.                                                                                                |
|              | <b>getFn</b>                                                                                                                                                                                                                          | The name of a function applied when the program retrieves the value of a variable that contained an activeValue.         |
|              | <b>putFn</b>                                                                                                                                                                                                                          | The name of a function that was applied when the program replaces the value of a variable that contained an activeValue. |
|              | If either the <b>getFn</b> or <b>putFn</b> fields is NIL, default actions returned or replaced the <b>localState</b> , respectively. Nesting was accomplished by the <b>localState</b> of an activeValue being itself an activeValue. |                                                                                                                          |



**ExplicitFnActiveValue**

[Class]

- Purpose: Mimics the behavior of the Buttress-style active values and allows simple changes to the user code triggered by the **ActiveValue** mechanism.
- Behavior: **Get** accesses to the wrapped variable cause the **getFn** to be called, and **Put** accesses cause **putFn** to be called. Enables the old style activeValues to look like the new style without changing any functionality.
- Instance Variables:
- localState** A place for data storage.
  - getFn** The name of a function applied when the active variable is read.
  - putFn** The name of a function applied when the active variable is changed.

**(MakeActiveValue self varOrSelector newGetFn newPutFn newLocalState propName type)**

[Function]

- Purpose: Makes the value of some variable an active value.
- Behavior: Creates a new activeValue record and installs it according to the arguments.
- Arguments:
- self* Object whose variable is changed to an active value.
  - varOrSelector* Variable name or method selector where the data type activeValue is placed.
  - newGetFn* and *newPutFn* If NIL, the old values of **getFn** and **putFn** are not overwritten. If T, the values of **getFn** and **putFn** are changed to NIL. Any other values are placed in the **getFn** and **putFn** fields of the activeValue.
  - newLocalState* The value of this argument is ignored. A new **ActiveValue** instance is always created. The contents of **localState** is changed to the previous value of the variable or property being made active.
  - propName* Name of the property, if the active value is to be placed on a property list. This is NIL if the active value is associated with a variable or method.
  - type* Indicates the type of the variable *varNameOrSelector*. Must be one of IV (or NIL) for instance variable, CV for class variable, CLASS for a class property, or METHOD for a method property.

**(DefAVP fnName putFlg)**

[Function]

- Purpose: Creates a template for defining an active value function.
- Behavior: Creates a template and leaves you in the Interlisp-D function editor.
- Arguments:
- fnName* Name of the function.
  - putFlg* T indicates function is a **putFn**; NIL indicates a **getFn**.
- Returns: The function name on exit from the editor.

**(GetLocalState activeValue self varName propName type)**

[Function]

Purpose: Retrieves data from **localState**.

Behavior: Retrieves the value stored in the **localState** of *activeValue*. Nested active values will be triggered.

Arguments: *activeValue* An **ActiveValue**.

*self* The object containing the **ActiveValue**.

*varName* The name of the variable were the **ActiveValue** is stored.

*propName* The name of an instance or class variable property. This is NIL if the **ActiveValue** is associated with the value of the variable itself.

*type* Specifies where the **ActiveValue** was stored. NIL means an instance variable, CV means class variable, CLASS means a class property, METHOD means a method property.

Returns: Contents of the **localState** field of *activeValue*.

---

**(PutLocalState *activeValue newValue self varName propName type*)** [Function]

---

Purpose: Data replacement.

Behavior: Stores *newValue* in the **localState** field of *activeValue*. Nested active values will be triggered.

Arguments: *activeValue* An **ActiveValue**.

*newValue* A new value to be stored in **localState**.

*self* The object containing the **ActiveValue**.

*varName* The name of the variable were the **ActiveValue** is stored.

*propName* The name of an instance or class variable property. This is NIL if the **ActiveValue** is associated with the value of the variable itself.

*type* Specifies where the **ActiveValue** was stored. NIL means an instance variable, CV means class variable, CLASS means a class property, METHOD means a method property.

Returns: The value of *newValue*.

---

**(GetLocalStateOnly *activeValue*)** [Function]

---

Purpose: Gets a value from **localState** without triggering any nested **ActiveValue**.

Behavior: Retrieves the value stored in the **localState** field of the **ActiveValue** without triggering any nested **ActiveValue**.

Arguments: *activeValue* The **ActiveValue** in which the **getFn** and **putFn** is found.

Returns: The contents of **localState**.

---

**(PutLocalStateOnly** *activeValue newValue*)

[Function]

- Purpose: Puts a value into a **localState** without triggering any nested **ActiveValues**.
- Behavior: Replaces the value stored in the **localState** of *activeValue* without triggering any nested **ActiveValue**.
- Arguments: *activeValue* An **ActiveValue**.  
*newValue* Value used for the replacement.
- Returns: The value of *newValue*.

---

**(ReplaceActiveValue** *activeValue newVal self varName propName type*)

[Function]

- Purpose: In an object's variable which has an **ActiveValue** installed, overwrites *activeVal* with *newVal*, providing a way of removing an **ActiveValue**.
- Behavior: Searches arbitrarily deep nesting to replace the occurrence of *activeVal* with *newVal*. If no match is found in the list that is the value of the variable described by the arguments, an error is invoked.
- Arguments: *activeValue* The **ActiveValue** to be replaced.  
*newVal* A new value to be stored in the object's variable.  
*self* The object containing the **ActiveValue**.  
*varName* The name of the variable where the **ActiveValue** is stored.  
*propName* The name of an instance or class variable property. This is NIL if the **ActiveValue** is associated with the value of the variable itself.  
*type* Specifies where the **ActiveValue** was stored. NIL means an instance variable, CV means class variable, CLASS means a class property, METHOD means a method property.  
*newValue* Value used for the replacement.
- Returns: Value of *newVal*.

A.2 GETFNS AND PUTFNS

---

A.2 GETFNS AND PUTFNS

---

---

## A.2 getFns and putFns

---

In the Buttress version of LOOPS, where the only kind of active value was equivalent to **ExplicitFnActiveValue**, specialization of active values was done not the way it is in Xerox LOOPS, but by the equivalent of putting special purpose functions into the **getFn** and **putFn** instance variables. The following functions emulate the behaviors they had in the Buttress version, using the current **ActiveValue** mechanisms.

In all cases, the functions are installed in the **getFn** or **putFn** instance variable of an **ActiveValue**, and are called when an attempt is made to get or put the variable where the **ActiveValue** is stored. The arguments and values returned are irrelevant to the use of these functions.



**(NoUpdatePermitted** *self varName oldOrNewValue propName activeValue type*) [Function]

---

Purpose: **putFn** for preventing the updating of a variable.

Behavior: LOOPS-defined **putFn** that causes a break if an attempt is made to replace the value of the variable containing the **ActiveValue**.

**(FirstFetch** *self varName oldOrNewValue PropName activeValue type*) [Function]

---

Purpose: **getFn** for dynamic variable initialization.

Behavior: LOOPS-defined **getFn** that expects the **localState** of *activeValue* to be an Interlisp-D expression to be evaluated. On the first fetch, the expression is evaluated and the variable or property is set to the value of the expression.

**(GetIndirect** *self varName oldOrNewValue PropName activeValue type*) [Function]

---

Purpose: LOOPS-defined **getFn** that functions as a pointer to another variable.

Behavior: **GetIndirect** and **PutIndirect** together set up an **ActiveValue** whose **localState** contains a pointer to where the actual value is stored. This is used when the value of a variable should always be the same as another.

**(PutIndirect** *self varName oldOrNewValue PropName activeValue type*) [Function]

---

Purpose: LOOPS-defined **putFn** that functions as a pointer to another variable.

Behavior: See **GetIndirect**.

**(ReplaceMe** *self varName oldOrNewValue PropName activeValue type*) [Function]

---

Purpose: LOOPS-defined **putFn** which removes both itself and any **getFn**.

Behavior: In some cases, you may want to compute a default value if given, but replace the active value by the value given if you set the value of a variable. For this, you can employ **ReplaceMe**. Any replacement attempt at the variable containing an **ActiveValue** with this as its **putFn** results in the value of the variable being replaced, and the **ActiveValue** disappearing.

**(AtCreation** *self varName oldOrNewValue PropName activeValue type*) [Function]

---

Purpose: LOOPS-defined **getFn** used to replace the active value with a dynamically computed value at instance creation.

Behavior: This function no longer works.

To achieve the closest functionality, use the **FirstFetchAV** specialization of the class **ActiveValue** (see Chapter 8, Active Values) or the **FirstFetch** function described above.

[This page intentionally left blank]

## **LOOPS DOCUMENTATION KIT CONFIGURATIONS**

*New and Existing LOOPS Customers (Koto to Lyric/Medley):*

610E15980     *Xerox LOOPS Reference Manual, Lyric/Medley (7/88)*  
                 *Xerox LOOPS Release Notes*  
                 *Xerox LOOPS Library Packages Manual*

*Errata Sheet*

610E16000     *Xerox LOOPS Users' Packages Manual, Lyric/Medley (7/88)*

# LYRIC/MEDLEY TAB TEXT

POINT SIZE: 10

FONT: OPTIMA

TABS:

XEROX LOOPS RELEASE NOTES

XEROX LOOPS REFERENCE MANUAL

XEROX LOOPS LIBRARY PACKAGES MANUAL

TABS FOR LYRIC/MEDLEY LOOPS (1-1/2" D-RING BINDER)

TAB COLOR: PMS GRAY 422-C

BANK 1

|  |  |
|--|--|
|  |  |
|--|--|

XEROX LOOPS RELEASE NOTES

BANK 2

|  |  |
|--|--|
|  |  |
|--|--|

XEROX LOOPS REFERENCE MANUAL

SUBTABS: See following page

BANK 3

|  |  |
|--|--|
|  |  |
|--|--|

XEROX LOOPS LIBRARY PACKAGES MANUAL

\_\_\_\_\_

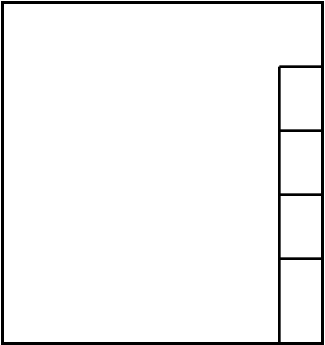
\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

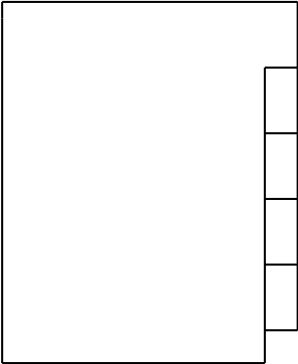
COLOR OF TABS: WHITE

BANK 1



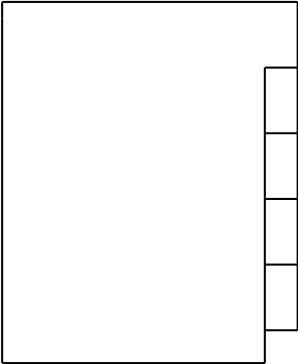
- Table of Contents
- 1. Introduction
  - 2. Instances
  - 3. Classes
  - 4. Metaclasses

BANK 2



- 5. Accessing Data
- 6. Methods
- 7. Message  
Sending Forms
- 8. Active Values
- 9. Data Type  
Predicates

BANK 3



- 10. Browsers
- 11. Errors  
and Breaks
- 12. Breaking  
and Tracing
- 13. Editing
- 14. File Manager

SUBTABS FOR LRYIC/MEDLEY LOOPS (1-1/2" D-RING BINDER)

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

COLOR OF TABS: WHITE

BANK 4

|  |  |
|--|--|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

- 15. Performance Issues
- 16. Processes
- 17. Reading and Printing
- 18. User Input/Output Modules
- 19. Windows

BANK 5

|  |  |
|--|--|
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

- 20. System Variables and Functions
- A. Active Values
- Glossary
- Index

POINT SIZE: 10

FONT: Optima

**SUBTABS:**

|                                                 |                          |                             |                                   |  |
|-------------------------------------------------|--------------------------|-----------------------------|-----------------------------------|--|
| Table of Contents<br>4. Metaclasses             | 1. Introduction          | 2. Instances                | 3. Classes                        |  |
| 5. Accessing Data<br>9. Data Type<br>Predicates | 6. Methods               | 7. Message<br>Sending Forms | 8. Active Values                  |  |
| 10. Browsers<br>14. File Manager                | 11. Errors<br>and Breaks | 12. Breaking<br>and Tracing | 13. Editing                       |  |
| 15. Performance<br>Issues                       | 16. Processes            | 17. Reading and<br>Printing | 18. User Input/<br>Output Modules |  |
| 20. System Variables<br>and Functions           | A. Active Values         | Glossary                    | Index                             |  |



---

**Replace this page with  
XEROX LOOPS RELEASE NOTES  
tab**

---

---

**Replace this page with  
XEROX LOOPS REFERENCE MANUAL  
tab**

---

---

Replace this page with  
XEROX LOOPS LIBRARY PACKAGES  
MANUAL  
tab

---

COVER TEXT:

NOTE: USE APPROPRIATE POINT SIZES TO ACHIEVE EFFECT  
SHOWN BELOW. USE XEROX STANDARDS FOR <<8-1/2x11>> BINDER COVERS

FONT

ARTIFICIAL INTELLIGENCE SYSTEMS  
XEROX LOOPS

HELVETICA  
BOLD ITALIC

USERS' PACKAGES MANUAL  
LYRIC/MEDLEY RELEASE

REGULAR or  
TRIUMVIRATE  
BLACK  
ITALIC

Part Number 610E16000

SPINE (1 INCH) :

ARTIFICIAL INTELLIGENCE SYSTEMS  
XEROX LOOPS  
USERS' PACKAGES MANUAL  
LYRIC/MEDLEY RELEASE

HELVETICA  
BOLD ITALIC

REGULAR or  
TRIUMVIRATE  
BLACK  
ITALIC

**LYRIC/MEDLEY LOOPS KIT**  
**on 5¼" floppies**  
**for Xerox 1186 workstation**  
**part number LYMLP1186**

**Documentation**

|        |                                                                           |                       |
|--------|---------------------------------------------------------------------------|-----------------------|
|        | Loose material:                                                           |                       |
| 150002 | Lyric/Medley LOOPS Cover Letter ( <i>single sheet</i> )                   |                       |
| 400008 | Documentation updates ( <i>4 pages, stapled</i> )                         |                       |
| 151010 | 1186 Lyric LOOPS Floppy Index and Directories ( <i>6 pages, stapled</i> ) |                       |
| 310000 | LOOPS MANUAL assembly:                                                    | (Xerox P/N 610E15980) |
|        | Large binder                                                              |                       |
|        | White ENVOS front cover                                                   |                       |
|        | Spine label: "LOOPS MANUAL" (Green)                                       |                       |
|        | LOOPS Reference Manual (310001)                                           |                       |
|        | LOOPS Library Packages Manual (310002)                                    |                       |
|        | LOOPS Release Notes (400009)                                              |                       |
| 310003 | LOOPS USERS' MODULES, manual assembly:                                    | (Xerox P/N 610E16000) |
|        | Small binder                                                              |                       |
|        | White ENVOS front cover                                                   |                       |
|        | Spine label: LOOPS USERS' MODULES (Green)                                 |                       |
|        | LOOPS Users' Packages Manual                                              |                       |

**Media**

|        |                       |
|--------|-----------------------|
| 210035 | Lyric LOOPS Library   |
| 210036 | Lyric LOOPS System #1 |
| 210037 | Lyric LOOPS System #2 |
| 210038 | Lyric LOOPS Users     |

**Envos Marketing Brochures**

|                            |                           |
|----------------------------|---------------------------|
| ROOMS™ Brochure            |                           |
| ROOMS™ Product Description | LOOPS Product Description |

**LYRIC/MEDLEY LOOPS KIT**  
**on 8" floppies**  
**for Xerox 1108 workstation**  
**part number LYML1108**

**Documentation**

|        |                                                                           |                       |
|--------|---------------------------------------------------------------------------|-----------------------|
|        | Loose material:                                                           |                       |
| 150002 | Lyric/Medley LOOPS Cover Letter ( <i>single sheet</i> )                   |                       |
| 400008 | Documentation updates ( <i>4 pages, stapled</i> )                         |                       |
| 151009 | 1108 Lyric LOOPS Floppy Index and Directories ( <i>5 pages, stapled</i> ) |                       |
| 310000 | LOOPS MANUAL assembly:                                                    | (Xerox P/N 610E15980) |
|        | Large binder                                                              |                       |
|        | White ENVOS front cover                                                   |                       |
|        | Spine label: "LOOPS MANUAL" (Green)                                       |                       |
|        | LOOPS Reference Manual (310001)                                           |                       |
|        | LOOPS Library Packages Manual (310002)                                    |                       |
|        | LOOPS Release Notes (400009)                                              |                       |
| 310003 | LOOPS USERS' MODULES, manual assembly:                                    | (Xerox P/N 610E16000) |
|        | Small binder                                                              |                       |
|        | White ENVOS front cover                                                   |                       |
|        | Spine label: LOOPS USERS' MODULES (Green)                                 |                       |
|        | LOOPS Users' Packages Manual                                              |                       |

**Media**

|        |                     |
|--------|---------------------|
| 212014 | Lyric LOOPS Library |
| 212015 | Lyric LOOPS System  |
| 212016 | Lyric LOOPS Users   |

**Envos Marketing Brochures**

|                            |                           |
|----------------------------|---------------------------|
| ROOMS™ Brochure            |                           |
| ROOMS™ Product Description | LOOPS Product Description |

**LYRIC/MEDLEY LOOPS KIT**  
**on 1/4" tape or 1/2" tape**  
**for Sun Workstations**  
**part number LYMLP3414 or LYMLP3412**

**Documentation**

Loose material:

|        |                                                                |
|--------|----------------------------------------------------------------|
| 150002 | Lyric/Medley LOOPS Cover Letter ( <i>single sheet</i> )        |
| 400008 | Documentation updates ( <i>4 pages, stapled</i> )              |
| 151012 | LOOPS Tape Directory ( <i>for 1/4-Inch tape, 1 page</i> )      |
| or     |                                                                |
| 151011 | LOOPS Tape Directory ( <i>for 1/2-Inch tape, 1 page</i> )      |
| 310000 | Sun installation of Lyric/Medley LOOPS (Document Update Sheet) |

Large binder w/"LOOPS MANUAL" spine label: (610E15980 Xerox)

- LOOPS Reference Manual (310001)
- LOOPS Library Packages Manual (310002)
- LOOPS Release Notes (400009)

Small binder w/"LOOPS USERS' MODULES" spine label:

|        |                                                |
|--------|------------------------------------------------|
| 310003 | LOOPS Users' Packages Manual (610E16000 Xerox) |
|--------|------------------------------------------------|

**Media**

|        |                                                 |
|--------|-------------------------------------------------|
| 215003 | Lyric/Medley LOOPS for the Sun 3, 1/4-Inch Tape |
| or     |                                                 |
| 216003 | Lyric/Medley LOOPS for the Sun 3, 1/2-Inch Tape |

**Envos Marketing Brochures**

|                            |                           |
|----------------------------|---------------------------|
| ROOMS™ Brochure            |                           |
| ROOMS™ Product Description | LOOPS Product Description |

**LYRIC/MEDLEY LOOPS KIT(OBSOLETE)**  
**on ½" tape**  
**for Sun 3 Workstation**  
**part number LYMLP3412**

**Documentation**

|        |                                                         |                       |
|--------|---------------------------------------------------------|-----------------------|
|        | Loose material:                                         |                       |
| 150002 | Lyric/Medley LOOPS Cover Letter ( <i>single sheet</i> ) |                       |
| 400008 | Documentation updates ( <i>4 pages, stapled</i> )       |                       |
| 151011 | LOOPS Tape Directory ( <i>for ½-Inch tape, 1 page</i> ) |                       |
| 310000 | LOOPS MANUAL assembly:                                  | (Xerox P/N 610E15980) |
|        | Large binder                                            |                       |
|        | White ENVOS front cover                                 |                       |
|        | Spine label: "LOOPS MANUAL" (Green)                     |                       |
|        | LOOPS Reference Manual (310001)                         |                       |
|        | LOOPS Library Packages Manual (310002)                  |                       |
|        | LOOPS Release Notes (400009)                            |                       |
| 310003 | LOOPS USERS' MODULES, manual assembly:                  | (Xerox P/N 610E16000) |
|        | Small binder                                            |                       |
|        | White ENVOS front cover                                 |                       |
|        | Spine label: LOOPS USERS' MODULES (Green)               |                       |
|        | LOOPS Users' Packages Manual                            |                       |

**Media**

|        |                                               |
|--------|-----------------------------------------------|
| 216003 | Lyric/Medley LOOPS for the Sun 3, ½-Inch Tape |
|--------|-----------------------------------------------|

**Envos Marketing Brochures**

|                            |                           |
|----------------------------|---------------------------|
| ROOMS™ Brochure            |                           |
| ROOMS™ Product Description | LOOPS Product Description |





# Sun 3 & 4 LOOPS — 1/4" tar Tape

## Manufacturing Instructions

(Envos Internal Use Only)

1. Go to Tree (or any available Sun 3 with a tape drive on it), and log in.
2. Insert a blank 1/4-inch cartridge in the tape drive. You must use a 600-foot tape.
3. Type the following commands (the text in bold):

```
tree%: cd /python/loops
tree%: makereleasetape
```

The system will type something like this:

You should have a tape in the cartridge drive already, and should be connected to the correct directory. Please verify that you are connected to the right place.

You are connected to:

/python/loops (or something ending in /python/pcl)

Type ^C to Abort

Type ^D to continue

4. Type control-D. The tape will now get written, followed by a message telling you that it is finished.
5. When the tape has finished moving (it rewinds automatically), remove it from the drive, and turn the write-protect tab to the "Safe" setting.
6. Label the cartridge and its container (green LOOPS labels). If those were the last 2 labels, make another 10 (Chuck's PC has the files).
7. Don't forget to log out.

# Sun 3 & 4 LOOPS — 1/2" 9-Track 1600 bpi

## Manufacturing Instructions

(Envos Internal Use Only)

1. Go to python and load a 1/2-inch tape into its tape drive. Hit the "Density" button until the light next to "1600" is lit. You'll need to use a small reel of tape for this.
2. At python's console, log in as yourself.
3. Type the following commands (the text in bold):

```
python%: cd /python/loops
python%: maketape1600
```

The system will type something like this:

You should have a tape mounted on python's mag tape drive, and should be connected to the correct directory. Please verify that you are connected to the right place.

You are connected to:

/python/loops (or something ending in /python/pcl)

Type ^C to Abort

Type ^D to continue

4. Type control-D. The tape will now get written, followed by a message telling you that it is finished.
5. When the tape has finished moving (it rewinds automatically), remove it from the drive, and remove the write-protect ring.
6. Label the tape (green Loops label). If that was the last label, make another 10 (Chuck's PC has the files).
7. Hit the "density" button on the tape drive until the light next to "Host" is lit.
8. Don't forget to log out.



*Venue*

# *LOOPS Reference Manual*

November, 1991  
Medley Release

---

Address comments to:  
Venue  
User Documentation  
1549 Industrial Road  
San Carlos, CA 94070  
415-508-9672

---

## LOOPS REFERENCE MANUAL

November, 1991

Copyright © 1988, 1991 by Venue.

All rights reserved.

LOOPS and Medley are trademarks of Venue.

UNIX® is a registered trademark of UNIX System Laboratories.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

---

The information in this document is subject to change without notice and should not be construed as a commitment by Venue. While every effort has been made to ensure the accuracy of this document, Venue assumes no responsibility for any errors that may appear.

Text was written and produced with Venue text formatting tools; Xerox printers were used to produce text masters. The typeface is Classic.

LOOPS integrates several programming paradigms to facilitate the design of artificial intelligence applications.

- Object-oriented programming, in which information is organized in terms of objects. Every object belongs to a class, and the classes are arranged in an inheritance lattice which allows complex objects to be described simply. Objects communicate with each other by sending messages. When an object receives a message, it performs some action, which can include sending messages to other objects.
- Procedure-oriented programming, in which smaller subroutines build larger procedures and in which data and instructions are kept separate.
- Access-oriented programming, in which accessing a value triggers an action. This paradigm is useful to monitor certain values.
- Rule-oriented programming, in which programs are organized around recursively composable sets of pattern-action rules. These rules provide a convenient way to describe flexible responses to a wide range of events. This part of LOOPS is included in the users' modules.

As a new user of LOOPS, you first must become familiar with its terminology and with the fundamental concepts described by that terminology. This chapter presents the terminology and related concepts.

---

## **1.1 Introduction to Objects**

---

This section shows the LOOPS hierarchy, called a lattice, in Figure 1-1, and describes the key terms in separate subsections. Terms appear in order of increasing complexity, with simpler terms described first and subsequent terms building on these simpler terms.

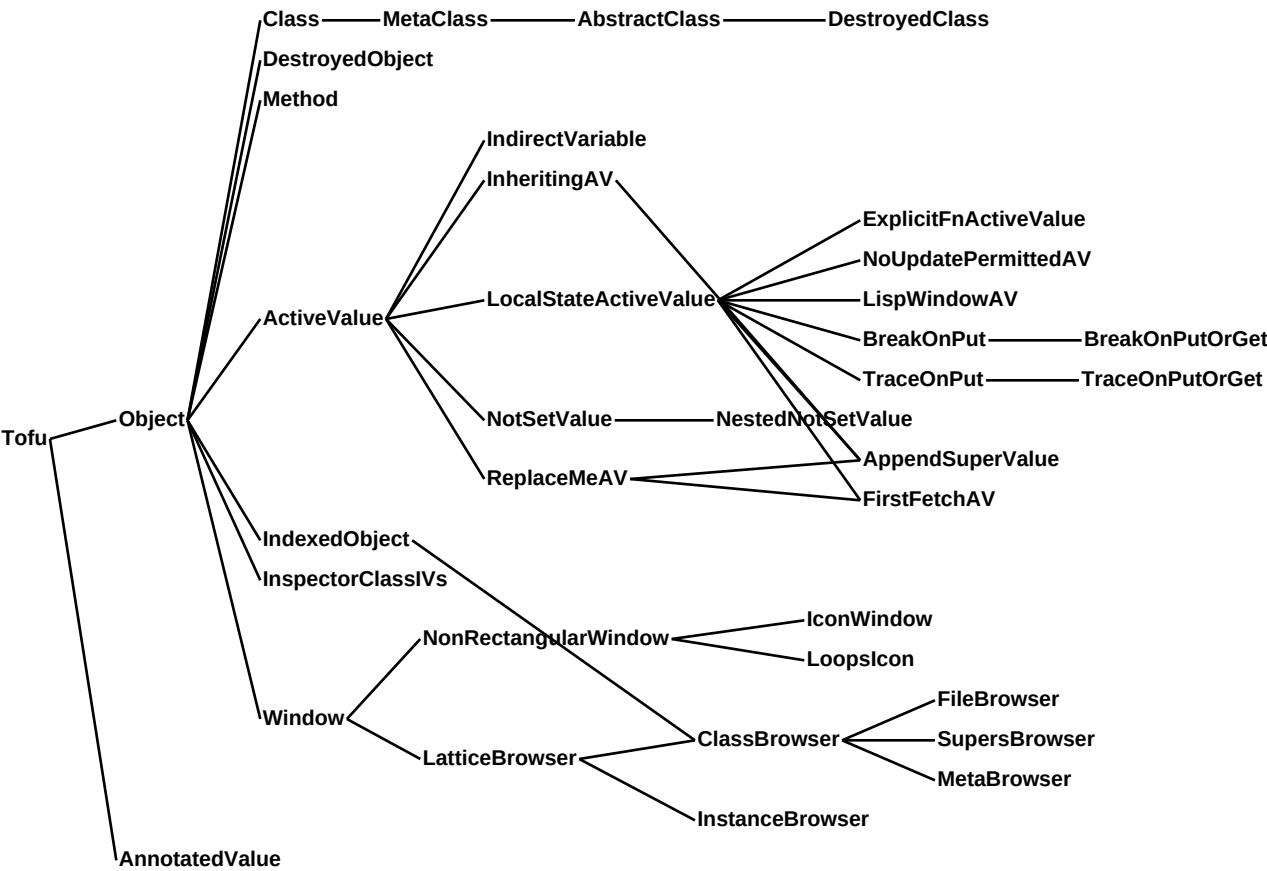


Figure 1-1. LOOPS Lattice

1.1.1 Object

As shown in Figure 1-2, an object is a structure consisting of data and a pointer to functionality that can manipulate the data. In procedure-oriented programming, data and functionality are considered as separate entities.

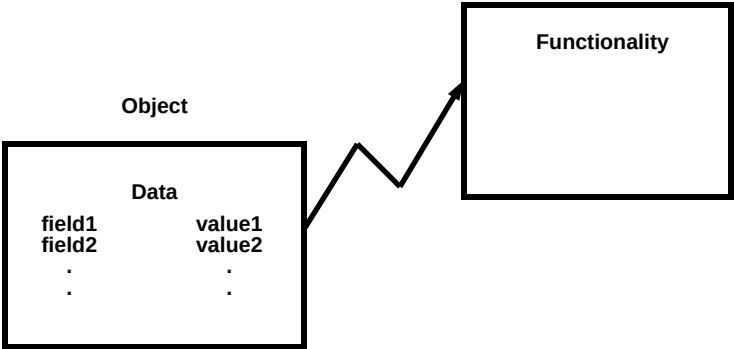


Figure 1-2. An Object

### 1.1.2 Message

Sending messages to objects provides an alternative to invoking procedures or calling functions. An object responds to a message by computing a value to be returned to the sender of the message, as shown in Figure 1-3. As a side effect, the data within an object may change during the computation. Messages contain a selector for the desired functionality. Messages may also contain arguments, as do procedures.

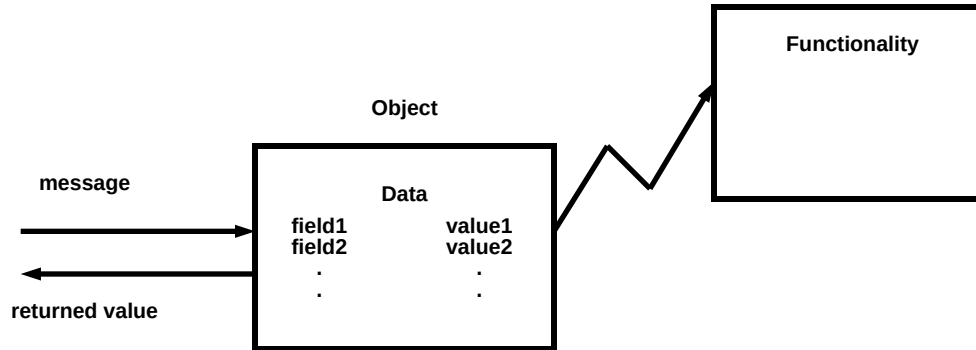


Figure 1-3. An Object Responding to a Message

### 1.1.3 Method

When an object receives a message, it determines what functionality it must apply to the arguments of the message. This functionality is called a method and is very similar to a procedure. A key concept that distinguishes methods from procedures is that in procedure-oriented programming, the calling routine determines which procedure to apply. In object-oriented programming, you determine the message to send and the object receiving the message determines the method to apply.

### 1.1.4 Selector

A message that is sent to an object contains a selector. The object uses the selector to determine which method is appropriate to apply to the message arguments. As shown in Figure 1-4, when an object receives a message with a specific selector, the object searches a lookup table containing selectors and methods to find the method associated with that particular selector.

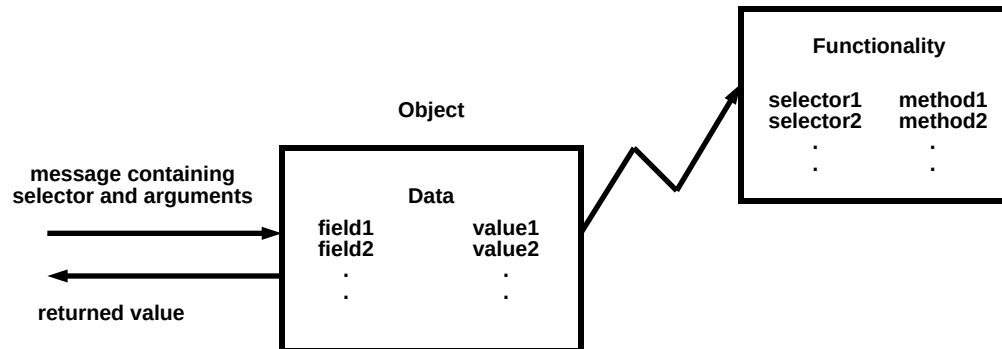


Figure 1-4. A Message Containing a Selector



### 1.1.5 Class

---

A class describes objects that are similar; that is, objects containing the same type of data fields and responding to the same messages, as shown in Figure 1-5. Think of the class that describes an object as being a template

for the functionality of its objects. When an object is sent a message, the class that describes that object handles the message, not the object itself. Different objects of the same class can respond to messages in the same way; that is, they apply the same method in response to receiving the same message.

To create new objects, send a message to a class requesting that a new object be created. Classes respond to messages because they are also objects.

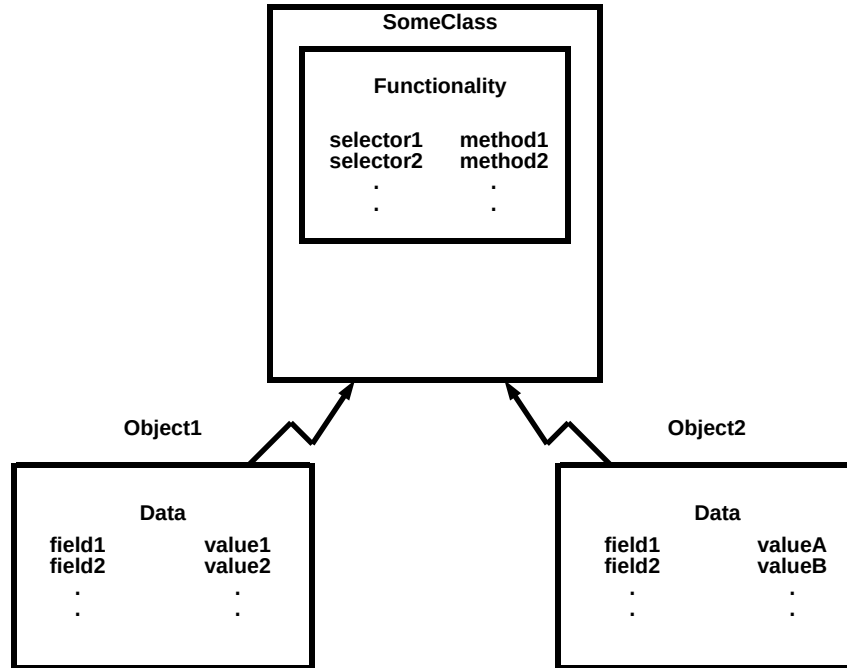


Figure 1-5. Class with Several Objects

### 1.1.6 Instance

An instance is an object described by a particular class. Every object within LOOPS is an instance of exactly one class.

## 1.2 Storage of Data in Objects

The data associated with an object is called an object's variables. Methods can change the values of these variables.

### 1.2 STORAGE OF DATA IN OBJECTS

#### 1.2 STORAGE OF DATA IN OBJECTS

### 1.2.1 Class Variables and Instance Variables

LOOPS supports two kinds of variables:

- Instance variables, often abbreviated IVs.

Instance variables contain the information specific to an instance.

- Class variables, often abbreviated CVs.

Class variables contain information shared by all instances of the class. A class variable is typically used for information about a class taken as a whole.

Both kinds of variables have names, values, and other properties. For example, the class for **Point** could specify two instance variables, **x** and **y**, and a class variable, **lastPoint**, used by methods associated with all points.

For any particular instance, you can access the values for the instance variables specific to that instance. You can also access the values for the class variables that are available to all instances of the same class.

Determining the value of a class variable requires a similar lookup procedure to that occurring when searching for a method to execute. Instance variable values are stored within the instances, and class variable values are stored within the class.

A class describes the structure of its instances by specifying the names and default values of instance variables, as shown in Figure 1-6. In this way, when a message is sent to a class to create a new instance, LOOPS can determine from the class description the number of instance variables for which it must allocate space the the initial values for those variables.

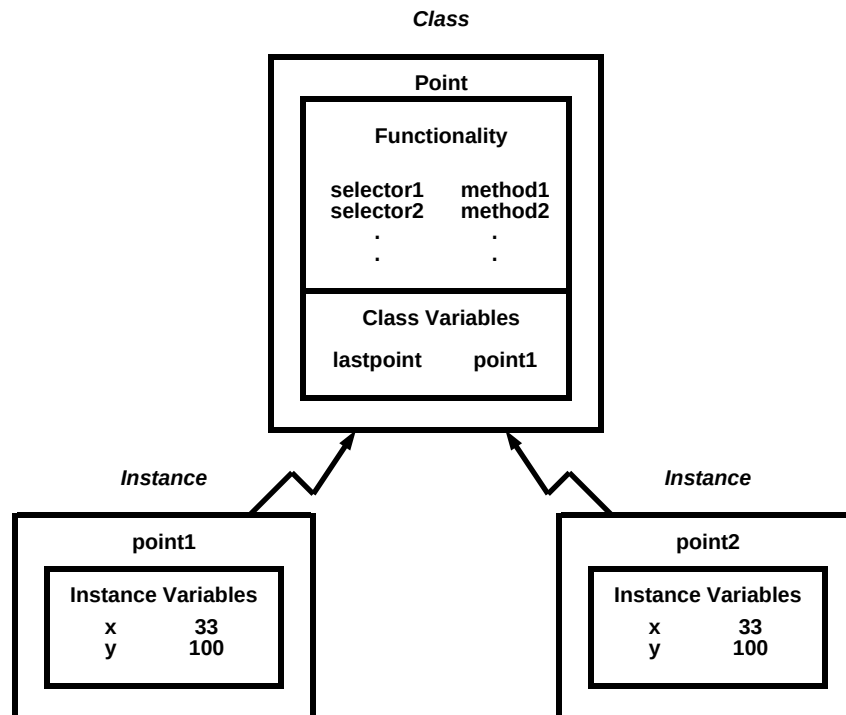


Figure 1-6. Class Variables and Instance Variables

---

## 1.2.2 Properties

LOOPS provides extensible property lists for classes, their variables, and their methods. Property lists provide places for storing documentation and additional kinds of information. For example, in a knowledge engineering

application, a property list for an instance variable could be used to store the following information:

- Support (reasons for believing a value)
- Certainty factor (numeric assessments of degree of belief) • Containments on values
- Dependencies (relationships to other variables)
- Histories (previous values)

## 1.3 METACLASSES

## 1.3 METACLASSES

## 1.3 Metaclasses

Classes themselves are instances of some class. Metaclasses are classes whose instances are classes. When a class is sent a message, its metaclass determines the response. For example, instances of a class are created by sending the class the message **New**. This message is handled by the class that describes the class receiving the message. For most classes, this method is provided by the standard metaclass for classes **Class**.

To create a new class, send a message to the class **Class**. The class that handles this message is **MetaClass**. Instances of **MetaClass** are classes that describe objects which are classes. Instances of **Class** are classes whose instances are not classes. Figure 1-7 shows an instance of **MetaClass**, which is a class named **Class**, and instances of **Class**, which are named **Window** and **Point**.

Another class available in the system is **AbstractClass**. This is useful when creating classes that implement general functionality, which must then be specialized into instantiable classes. Instances of this class are classes that are impossible to instantiate. An example of an **AbstractClass** is **ActiveValue**, which is described in Chapter 5, Active Values.

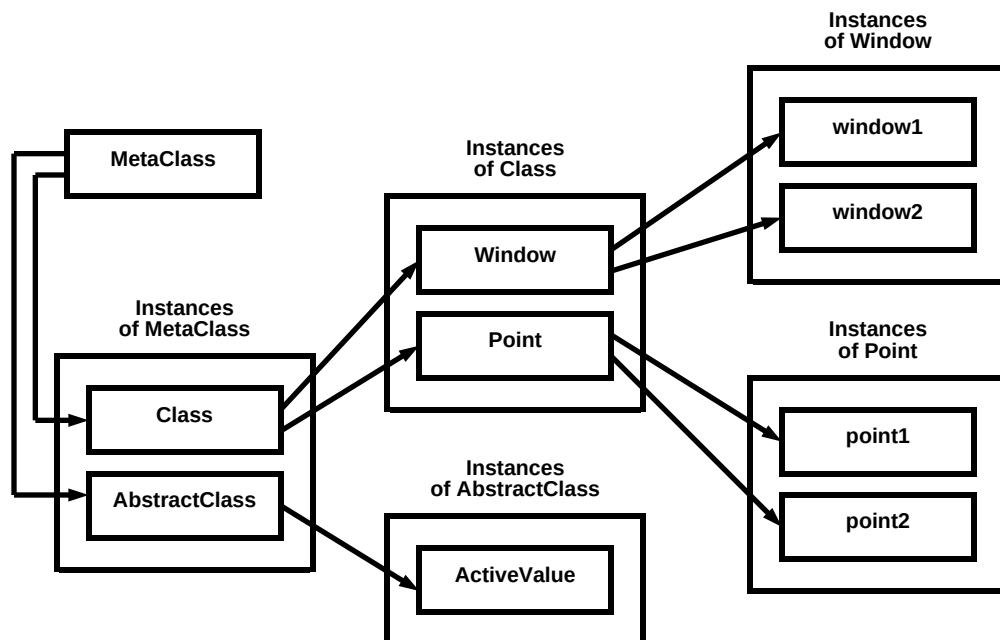


Figure 1-7. A Metaclass and its Instances

---

## 1.4 Introduction to Inheritance

---

Inheritance allows you to organize information in objects. With a few incremental changes, you can use inheritance to create objects that are almost like other objects. Inheritance allows you to avoid specifying redundant information and simplifies updating, since information that is common to several objects need be changed in only one place.

LOOPS objects exist in an inheritance network of classes. Figure 1-8 shows an example in which a class **3DPoint** is a subclass of another class **Point**. Instances of **3DPoint** contain instance variables that are defined in

1.4 INTRODUCTION TO INHERITANCE

---

### 1.4 INTRODUCTION TO INHERITANCE

---

**Point** as well as **3DPoint**. **Point** is referred to as a superclass of **3DPoint**. When an instance of **3DPoint** is created, the instance variables it contains and the messages to which it responds are not limited to those instance variables or methods as defined in the class **3DPoint**. For example, the object **pt2** contains three instance variables; two of them are inherited from the class **Point** and the other defined in the class **3DPoint**. This instance can also respond to three different messages containing one of the three different selectors: **selector1**, **selector2**, or **selector3**.

All descriptions in a class are inherited by a subclass unless overridden in the subclass. For methods and class variables, this is implemented by a runtime search for the information, looking first in the class, and then at the superclasses specified by its supers list. For instance variables, no search is made at run time. Default values are cached in the class, and are updated if any superclass is changed, thus maintaining the same semantics as the search. Each class can specify inheritance of structure and behavior from any number of superclasses.

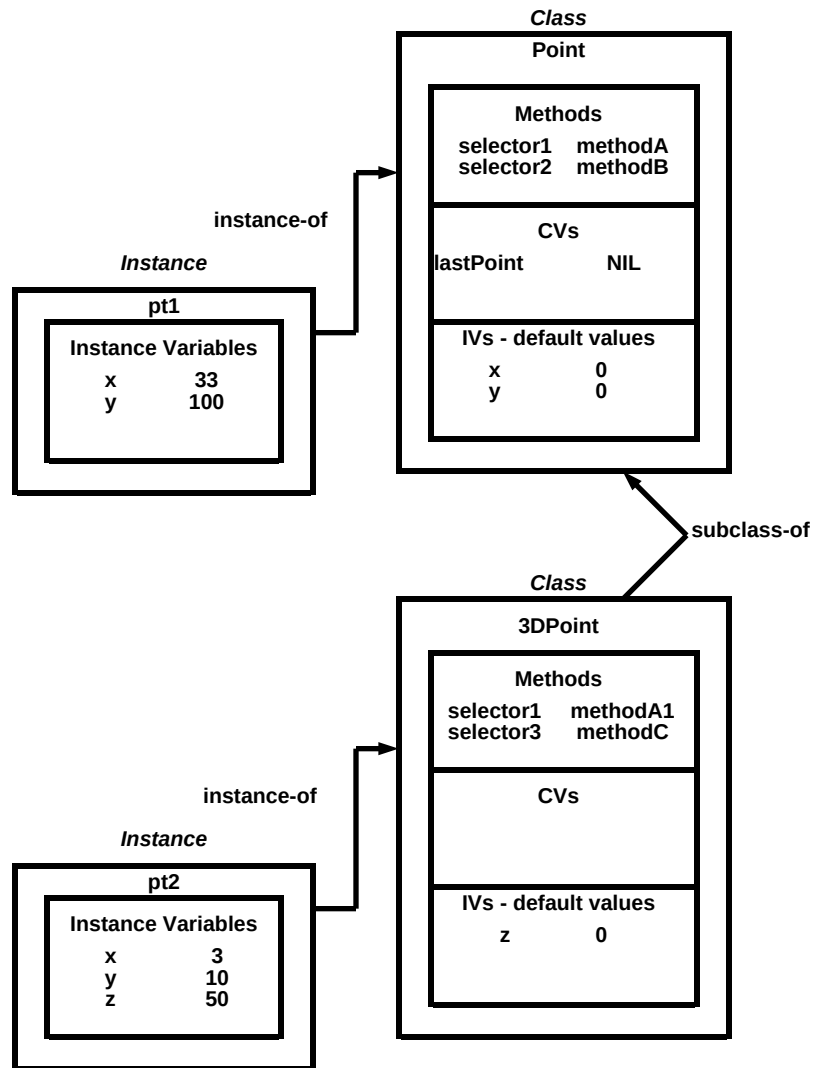


Figure 1-8. A Sample Inheritance Network

### 1.4.1 Single Superclasses

In the simplest case, each class specifies only one superclass. If the class **A** has the supers list (**B**), which is a one-element list containing **B**, then all of the instance variables specified local to **A** are added to those specified for **B**, recursively. That is, **A** gets all those instance variables described in **B** and all of **B**'s supers. For example, in Figure 1-9, **A** has instance variables **x**, **z**, and **B1**.

Any conflict of variable names is resolved by using the description closer to **A** in traversing up the hierarchy to its top at the class **Object**. Method lookup uses the same conflict resolution. The method to respond to a message is obtained by first searching in **A**, and then searching recursively in **A**'s supers list. For example, in Figure 1-9, the method **selector2** uses **methodA2** instead of **methodB2**.

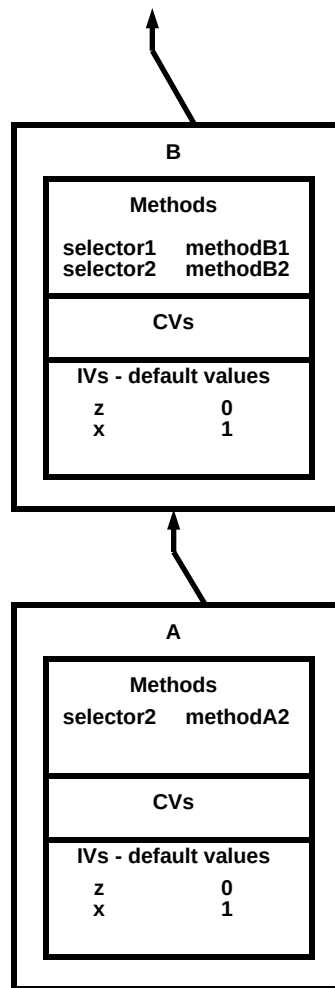


Figure 1-9. A Class with a Single Superclass

## 1.4.2 Multiple Superclasses

---

Classes in LOOPS can have more than one class specified on their supers list. Multiple superclasses permit a modular programming style where the following conditions hold:

- Methods and associated variables for implementing a particular feature are placed in a single class.
- Objects requiring combinations of independent features inherit them from multiple supers.

As in Figure 1-10, if **A** has the supers list (**B C**), first the description from **A** is used, then the description from **B** and its supers is inherited, and finally the description from **C** and its supers. In the simplest usage, the different features have unique variable names and selectors in each super. In case of a name conflict, LOOPS uses a depth first left-to-right precedence.

For example, if any super of **B** had a method for **selector3**, then it would be used instead of the method **methodC3** from **C**. In every case, inheritance from **Object** is only considered after all other classes on the recursively defined supers list. The general rule is left-to-right, depth first, up to where the separate branches of the hierarchy join together; that is, up to any class that is repeated. Alternatively, consider the list as generated by listing all the

superclasses in a depth first left-to-right order, eliminating all but the last occurrence of a class in the list.

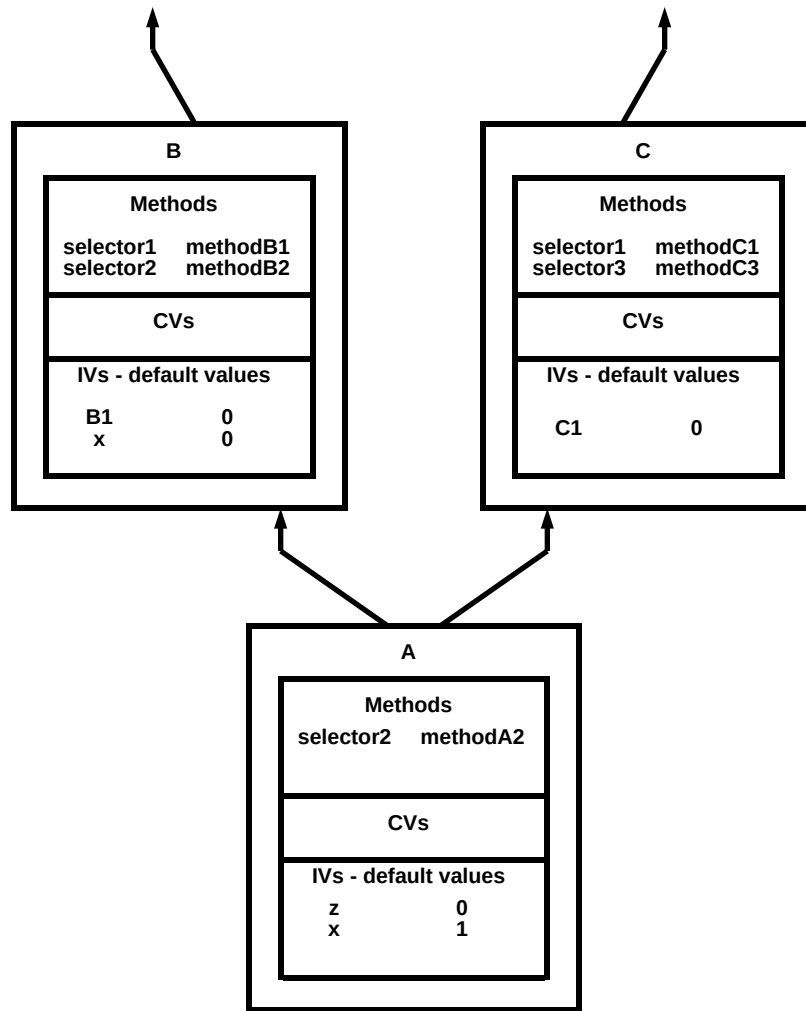


Figure 1-10. A Class with Multiple Superclasses

## 1.5 Introduction to Access-Oriented Programming: Using Active Values

In access-oriented programming, you can specify a particular procedure to invoke for read or write access of any variable of an object. LOOPS checks every object variable access to determine whether the value is marked as an active value. An active value is a LOOPS object. If a variable is marked as an active value, then aa message is sent to the active value object whenever the variable is read or set. This mechanism is dual to the notion of sending messages. Messages are a way of telling objects to perform operations, which can change their variables as a side effect. Active values are a way of accessing variables, which can send messages as a side effect.

The messages sent to the active value object will depend on the type of access. If you try to read a variable, the message **GettingWrapped Value** is sent to the active value object. If you try to set a variable, the message **Putting Wrapped Value** is sent. The object receiving the message may or may not trigger side effects as the result of receiving these messages. In this way, you have control over the side effects that may occur as a result of accessing data.



Active values enable one process to monitor another one. For example, LOOPS has debugging tools that use active values to trace and trap references to object variables. A graphics module updates views of particular objects on a display when their variables are changed. In both cases, the monitoring process is invisible to, and isolated from, the monitored process. No changes to the code of the monitored object are necessary to enable monitoring.

Active values can also be used to maintain constraints among data in a system. As one piece of data changes, the active value associated with that data can contain functionality that updates other data within the system. Examples of this are spreadsheets or electric circuit modeling.

A powerful feature of active values is that they can be nested to yield a natural composition of the access functions.

---

## 1.6 Introduction to the LOOPS User Interface

---

A key feature of LOOPS is its smooth integration with the Venue Medley environment. Many of the tools within Medley have been extended to provide the necessary functionality for manipulating objects. Among these tools are the following:

- SEdit
- The inspector
- Masterscope
- The File Manager
- The Library Module Grapher

This section describes how LOOPS interfaces with each of these Medley tools.

Another aspect of LOOPS is that objects have a name space that is separate from the Lisp name space. LOOPS names are Interlisp symbols. Applying a LOOPS function **\$** to a Medley symbol extracts a pointer to a LOOPS object. Objects can also be pointed to as a Lisp value.

---

### 1.6.1 SEdit

---

Class structures are Lisp data types. To change a class structure, LOOPS creates a list structure source for the class definition. This list can then be edited easily by SEdit. Upon exiting SEdit, the list structure is converted back to a data type. This process of converting to and from a list is hidden from view.

---

### 1.6.2 Inspector

---

Inspector macros have been defined within LOOPS that allow you to view the necessary class and instance data while hiding implementation details. Inspectors opened on classes or instances also provide functionality for changing the way one views an object. As an example, you can inspect a class and see or not see information inherited from its superclasses.

### **1.6.3 Masterscope**

---

The Library Module Masterscope has been extended to a LOOPS Library Module so that message sending and the use of instance and class variables are understood. The functionality of CHECK has been extended to allow consistency checking of LOOPS methods.

### **1.6.4 File Manager**

---

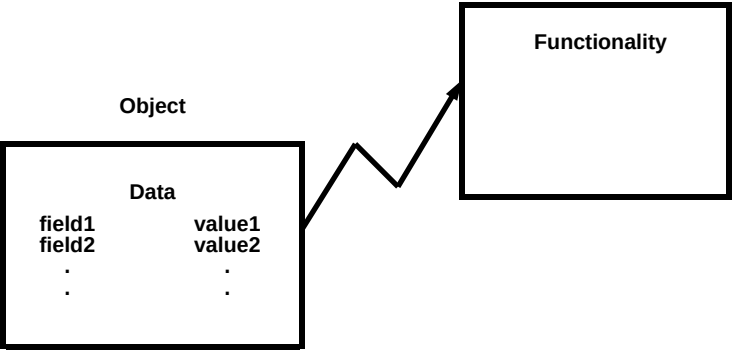
Additional File Manager commands have been added to allow you to save classes, instances, and methods on files.

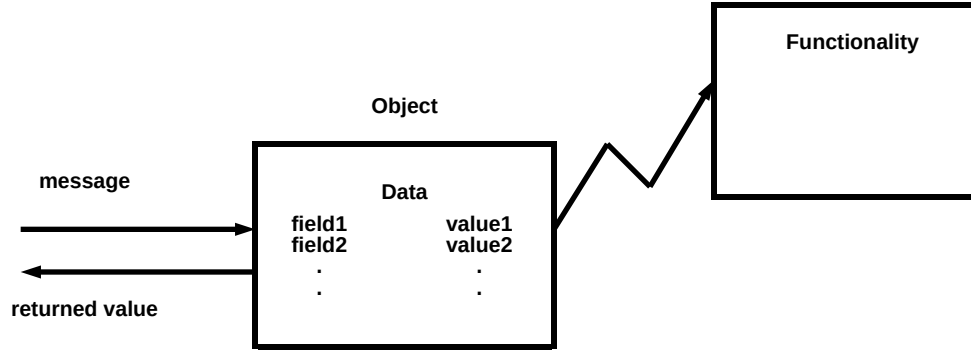
### **1.6.5 Grapher Module**

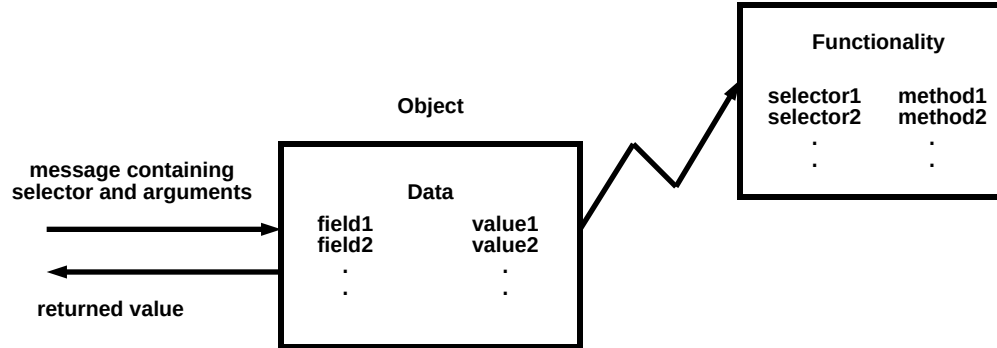
---

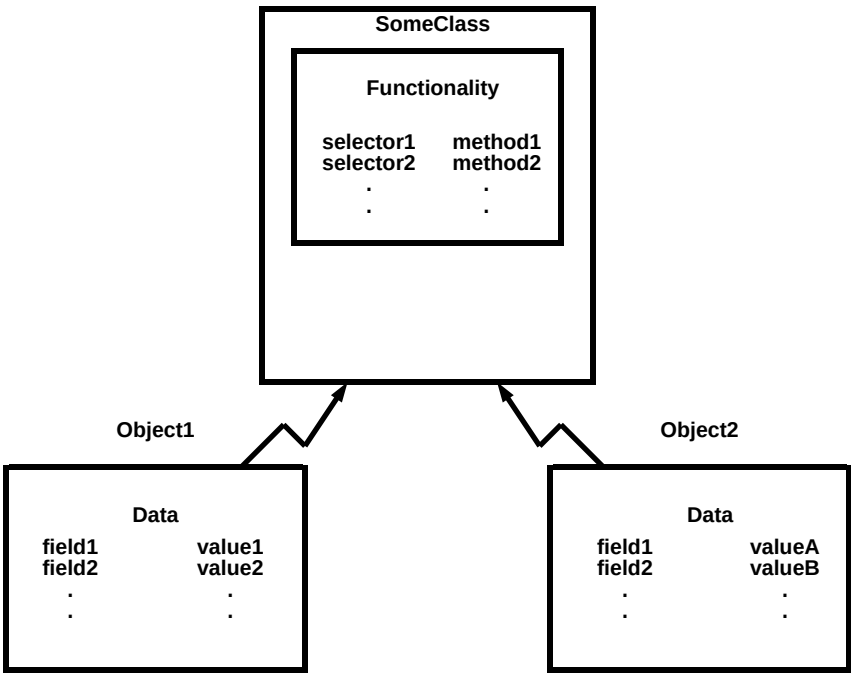
An important part of the LOOPS interface is its ability to show relationships between objects and to enable the programmer to easily manipulate those objects. Browsers of various kinds are in the system to allow you to understand the relationships between classes and how those classes are related to files. The browsers are built upon the Library module Grapher. You can easily extend the built-in browsers to create views onto any object relationship. An example of this is a decision tree where each node was an object representing a particular state of a system.

[This page intentionally left blank]

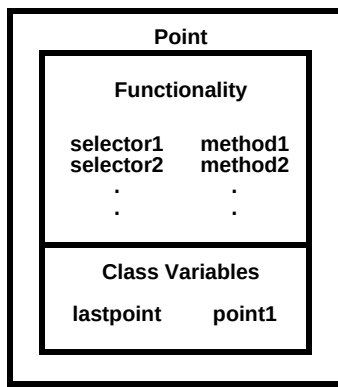








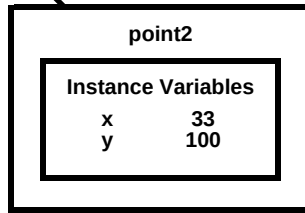
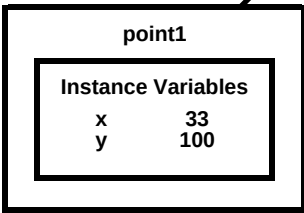
*Class*



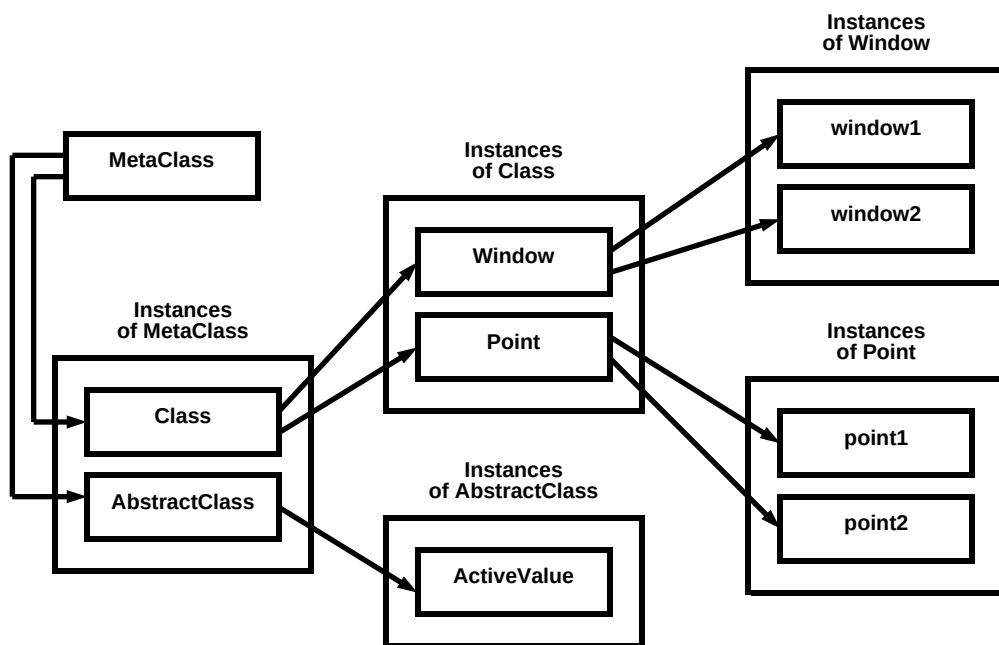
*Instance*

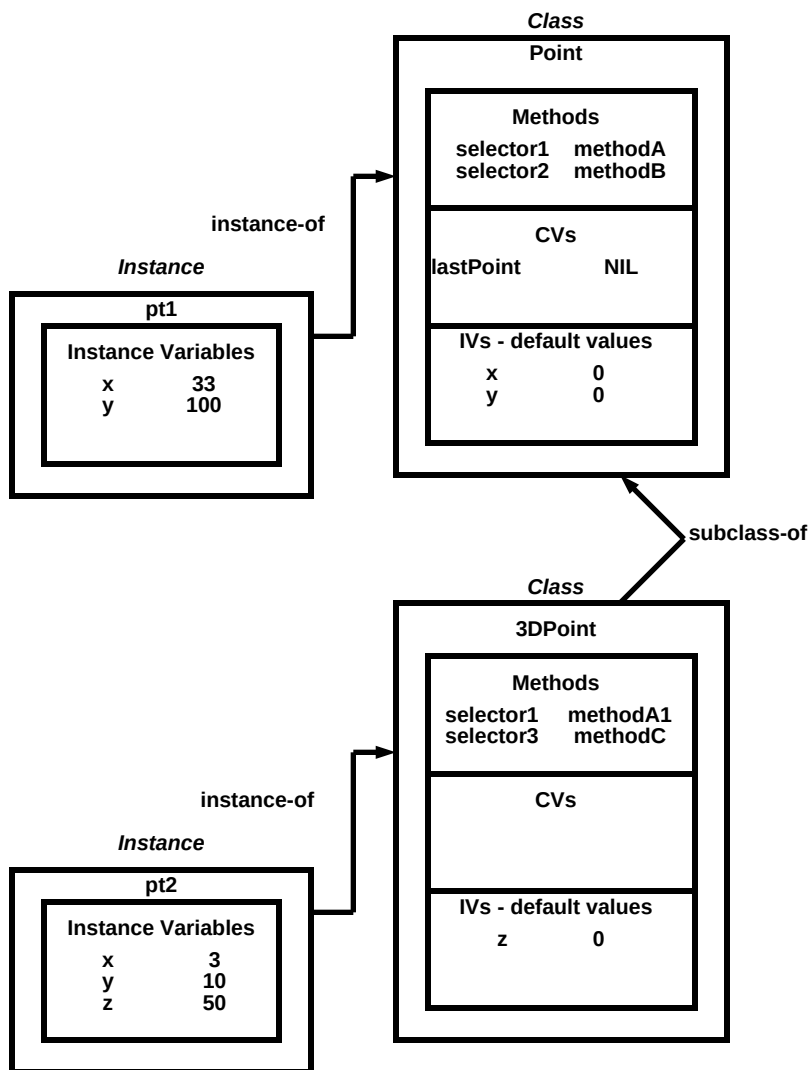


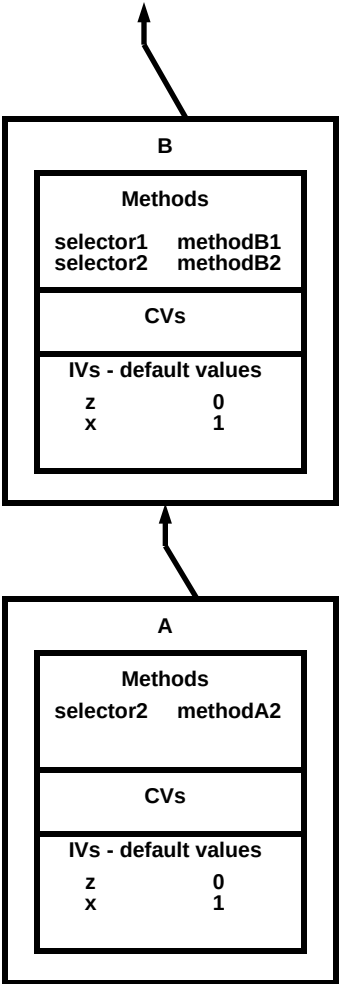
*Instance*

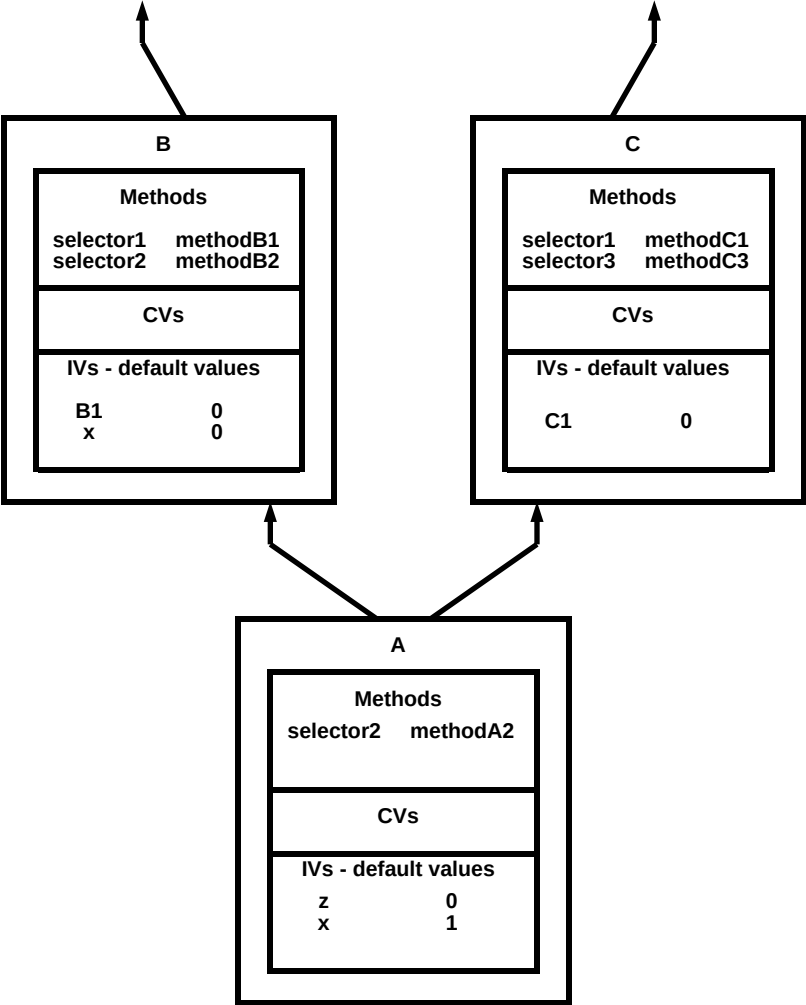












# TABLE OF CONTENTS

---

PREFACE

xv

## 1. INTRODUCTION

|                                                                            |      |
|----------------------------------------------------------------------------|------|
| 1.1 Introduction to Objects .....                                          | 1-1  |
| 1.1.1 Object .....                                                         | 1-2  |
| 1.1.2 Message .....                                                        | 1-3  |
| 1.1.3 Method .....                                                         | 1-3  |
| 1.1.4 Selector .....                                                       | 1-3  |
| 1.1.5 Class .....                                                          | 1-3  |
| 1.1.6 Instance .....                                                       | 1-4  |
| 1.2 Storage of Data in Objects .....                                       | 1-4  |
| 1.2.1 Class Variables and Instance Variables .....                         | 1-4  |
| 1.2.2 Properties .....                                                     | 1-5  |
| 1.3 Metaclasses .....                                                      | 1-6  |
| 1.4 Introduction to Inheritance .....                                      | 1-6  |
| 1.4.1 Single Superclasses .....                                            | 1-7  |
| 1.4.2 Multiple Superclasses .....                                          | 1-8  |
| 1.5 Introduction to Access-Oriented Programming: Using Active Values ..... | 1-9  |
| 1.6 Introduction to the LOOPS User Interface .....                         | 1-10 |
| 1.6.1 SEdit .....                                                          | 1-10 |
| 1.6.2 Inspector .....                                                      | 1-10 |
| 1.6.3 Masterscope .....                                                    | 1-10 |
| 1.6.4 File Manager .....                                                   | 1-11 |
| 1.6.5 Grapher Module .....                                                 | 1-11 |

## 2. INSTANCES

|                                                                    |      |
|--------------------------------------------------------------------|------|
| 2.1 Instance Naming Conventions .....                              | 2-1  |
| 2.2 Creating Instances .....                                       | 2-4  |
| 2.3 Data Storage in Instances at Creation Time .....               | 2-8  |
| 2.4 Changing the Number of Instance Variables in an Instance ..... | 2-10 |
| 2.5 Moving Variables.....                                          | 2-13 |
| 2.6 Destroying Instances.....                                      | 2-15 |
| 2.7 Methods Concerning the Class of an Object.....                 | 2-16 |
| 2.8 Copying Instances.....                                         | 2-19 |
| 2.9 Querying Structure of Instances.....                           | 2-21 |
| 2.10 Other Instance Items.....                                     | 2-24 |

## 3. CLASSES

|                                                  |      |
|--------------------------------------------------|------|
| 3.1 Creating Classes.....                        | 3-1  |
| 3.1.1 Function Calling and Message Sending ..... | 3-2  |
| 3.1.2 Dynamic Mixins.....                        | 3-4  |
| 3.2 Destroying Classes .....                     | 3-5  |
| 3.3 Inheritance .....                            | 3-7  |
| 3.4 Editing Classes .....                        | 3-10 |
| 3.5 Modifying Classes.....                       | 3-11 |
| 3.6 Methods for Manipulating Class Names .....   | 3-16 |
| 3.7 Querying the Structure of a Class.....       | 3-17 |
| 3.8 Copying Classes and Their Contents.....      | 3-23 |
| 3.9 Enumerating Instances of Classes.....        | 3-24 |
| 3.10 Dealing with Inheritance.....               | 3-27 |

## 4. METACLASSES

|                               |     |
|-------------------------------|-----|
| 4.1 Specific Metaclasses..... | 4-1 |
| 4.1.1 Metaclass Class.....    | 4-1 |

|                                      |     |
|--------------------------------------|-----|
| 4.1.2 Metaclass Metaclass .....      | 4-2 |
| 4.1.3 Metaclass AbstractClass .....  | 4-2 |
| 4.1.4 Metaclass DestroyedClass ..... | 4-2 |
| 4.2 Pseudoclasses .....              | 4-2 |
| 4.3 Defining New Metaclasses .....   | 4-5 |
| 4.4 Tofu .....                       | 4-6 |

## 5. ACCESSING DATA

|                                             |      |
|---------------------------------------------|------|
| 5.1 Generalized Get and Put Functions ..... | 5-1  |
| 5.2 Accessing Data in Instances .....       | 5-4  |
| 5.2.1 Compact Accessing Forms .....         | 5-10 |
| 5.2.2 Support for Changetran .....          | 5-13 |
| 5.3 Accessing Data in Classes .....         | 5-13 |
| 5.3.1 Metaclasses and Property Access ..... | 5-13 |
| 5.3.2 Class Variable Access .....           | 5-16 |
| 5.3.3 Instance Variable Access .....        | 5-19 |

## 6. METHODS

|                                                     |      |
|-----------------------------------------------------|------|
| 6.1 Categories .....                                | 6-1  |
| 6.2 Structure of Method Functions .....             | 6-3  |
| 6.3 Creating, Editing, and Destroying Methods ..... | 6-4  |
| 6.4 Escaping from Message Syntax .....              | 6-6  |
| 6.5 Movement between Classes .....                  | 6-8  |
| 6.5.1 Movement of Methods .....                     | 6-8  |
| 6.5.2 Stack Method Macros .....                     | 6-10 |

## 7. MESSAGE SENDING FORMS

## 8. ACTIVE VALUES

|                                                                                    |      |
|------------------------------------------------------------------------------------|------|
| 8.1 Using Active Values .....                                                      | 8-2  |
| 8.2 Specializations of the Class <code>ActiveValue</code> .....                    | 8-2  |
| 8.2.1 <code>IndirectVariable</code> .....                                          | 8-3  |
| 8.2.2 <code>LocalStateActiveValue</code> .....                                     | 8-6  |
| 8.2.2.1 <code>ExplicitFnActiveValue</code> .....                                   | 8-8  |
| 8.2.2.2 <code>NoUpdatePermittedAV</code> .....                                     | 8-9  |
| 8.2.2.3 <code>LispWindowAV</code> .....                                            | 8-10 |
| 8.2.2.4 Breaking and Tracing Active Values .....                                   | 8-10 |
| 8.2.2.5 <code>AppendSuperValue</code> .....                                        | 8-11 |
| 8.2.2.6 <code>FirstFetchAV</code> .....                                            | 8-12 |
| 8.2.3 <code>InheritingAV</code> .....                                              | 8-14 |
| 8.2.4 <code>ReplaceMeAV</code> .....                                               | 8-15 |
| 8.2.5 <code>NotSetValue</code> .....                                               | 8-15 |
| 8.2.5.1 <code>NestedNotSetValue</code> .....                                       | 8-16 |
| 8.2.6 User Specializations of Active Values .....                                  | 8-16 |
| 8.3 Active Value Methods .....                                                     | 8-16 |
| 8.3.1 Adding and Deleting Active Values .....                                      | 8-17 |
| 8.3.2 Fetching and Replacing Wrapped Values .....                                  | 8-19 |
| 8.3.3 Get and Put Functions Bypassing the <code>ActiveValue</code> Mechanism ..... | 8-22 |
| 8.3.4 Shared Active Values in Variable Inheritance .....                           | 8-22 |
| 8.3.5 Creating Your Own Active Values .....                                        | 8-23 |
| 8.4 Annotated Values .....                                                         | 8-24 |
| 8.4.1 Explicit Control over Annotated Values .....                                 | 8-25 |
| 8.4.2 Saving and Restoring Annotated Values .....                                  | 8-26 |
| 8.5 Active Values in Class Structures .....                                        | 8-27 |



## 9. DATA TYPE PREDICATES AND ITERATIVE OPERATORS

|                               |     |
|-------------------------------|-----|
| 9.1 Data Type Predicates..... | 9-1 |
| 9.2 Iterative Operators ..... | 9-3 |

## 10. BROWSERS

|                                                                    |       |
|--------------------------------------------------------------------|-------|
| 10.1 Types of Built-in Browsers .....                              | 10-1  |
| 10.1.1 Lattice Browsers .....                                      | 10-2  |
| 10.1.2 Class Browsers.....                                         | 10-2  |
| 10.1.3 File Browsers .....                                         | 10-2  |
| 10.1.4 Supers Browsers .....                                       | 10-2  |
| 10.1.5 Metaclass Browsers.....                                     | 10-2  |
| 10.1.6 Instance Browsers .....                                     | 10-3  |
| 10.2 Opening Browsers.....                                         | 10-3  |
| 10.2.1 Using Menu Options to Open Browsers .....                   | 10-3  |
| 10.2.1.1 Overview of Background Menu and LOOPS Icon.....           | 10-3  |
| 10.2.1.2 Command Summary .....                                     | 10-4  |
| 10.2.2 Using Commands to Open Browsers .....                       | 10-5  |
| 10.3 Using Class Browsers, Meta Browsers, and Supers Browsers..... | 10-7  |
| 10.3.1 Selecting Options in the Title Bar Menu .....               | 10-8  |
| 10.3.1.1 Recompute and its Suboptions .....                        | 10-8  |
| 10.3.1.2 AddRoot and its Suboptions .....                          | 10-10 |
| 10.3.1.3 Add Category Menu .....                                   | 10-10 |
| 10.3.2 Selecting Options in the Left Menu .....                    | 10-11 |
| 10.3.2.1 PrintSummary and its Suboptions.....                      | 10-12 |
| 10.3.2.2 Doc (ClassDoc) and its Suboptions .....                   | 10-13 |
| 10.3.2.3 WhereIs and its Suboptions .....                          | 10-14 |
| 10.3.2.4 DeleteFromBrowser and its Suboptions .....                | 10-16 |
| 10.3.2.5 SubBrowser .....                                          | 10-16 |
| 10.3.2.6 TypeInName .....                                          | 10-16 |
| 10.3.2.7 Extending Functionality with the Left Mouse Button .....  | 10-16 |
| 10.3.3 Selecting Options in the Middle Menu .....                  | 10-17 |

|          |                                                  |       |
|----------|--------------------------------------------------|-------|
| 10.3.3.1 | Box/UnBoxNode .....                              | 10-17 |
| 10.3.3.2 | Methods (EditMethod) and its Suboptions .....    | 10-18 |
| 10.3.3.3 | Add (AddMethod) and its Suboptions .....         | 10-20 |
| 10.3.3.4 | Delete (DeleteMethod) and its Suboptions .....   | 10-21 |
| 10.3.3.5 | Move (MoveMethodTo) and its Suboptions .....     | 10-22 |
| 10.3.3.6 | Copy (CopyMethodTo) and its Suboptions .....     | 10-23 |
| 10.3.3.7 | Rename (RenameMethod) and its Suboptions .....   | 10-23 |
| 10.3.3.8 | Edit (EditClass) and its Suboptions.....         | 10-24 |
| 10.4     | Using File Browsers .....                        | 10-24 |
| 10.4.1   | Selecting Options in the Title Bar Menu .....    | 10-25 |
| 10.4.1.1 | Recompute and its Suboptions .....               | 10-25 |
| 10.4.1.2 | AddRoot and its Suboptions .....                 | 10-25 |
| 10.4.1.3 | Add Category Menu .....                          | 10-25 |
| 10.4.1.4 | Change display mode and its Suboptions.....      | 10-25 |
| 10.4.1.5 | Uses IV? and its Suboptions.....                 | 10-26 |
| 10.4.1.6 | Edit FileComs and its Suboptions .....           | 10-28 |
| 10.4.1.7 | CLEANUP file and its Suboptions.....             | 10-30 |
| 10.4.2   | Selecting Options in the Left Menu .....         | 10-30 |
| 10.4.2.1 | PrintSummary and its Suboptions.....             | 10-30 |
| 10.4.2.2 | Doc (ClassDoc) and its Suboptions .....          | 10-30 |
| 10.4.2.3 | WhereIs (WhereIsMethod) and its Suboptions ..... | 10-30 |
| 10.4.2.4 | DeleteFromBrowser and its Suboptions .....       | 10-31 |
| 10.4.2.5 | SubBrowser .....                                 | 10-31 |
| 10.4.2.6 | TypeInName .....                                 | 10-31 |
| 10.4.2.7 | AddSubs and its Suboptions.....                  | 10-31 |
| 10.4.3   | Selecting Options in the Middle Menu .....       | 10-31 |
| 10.4.3.1 | BoxNode .....                                    | 10-32 |
| 10.4.3.2 | Methods (EditMethod) and its Suboptions .....    | 10-32 |
| 10.4.3.3 | Add (AddMethod) and its Suboptions .....         | 10-32 |
| 10.4.3.4 | Delete (DeleteMethod) and its Suboptions .....   | 10-32 |
| 10.4.3.5 | Move (MoveMethodTo) and its Suboptions .....     | 10-32 |
| 10.4.3.6 | Copy (CopyMethodTo) and its Suboptions .....     | 10-32 |
| 10.4.3.7 | Rename (RenameMethod) and its Suboptions.....    | 10-32 |
| 10.4.3.8 | Edit (EditClass) and its Suboptions.....         | 10-32 |

|                                                               |       |
|---------------------------------------------------------------|-------|
| 10.4.3.9 UsesIV and its Suboptions.....                       | 10-33 |
| 10.5 Programmer's Interface to Lattice Browsers .....         | 10-33 |
| 10.5.1 Instance Variables for the Class LatticeBrowser .....  | 10-33 |
| 10.5.2 Class Variables for the Class LatticeBrowser .....     | 10-34 |
| 10.5.3 Methods for the Class LatticeBrowser .....             | 10-35 |
| 10.6 Instance Browsers.....                                   | 10-50 |
| 10.6.1 Instance Variables for the Class InstanceBrowser ..... | 10-50 |
| 10.6.2 Methods for the Class InstanceBrowser .....            | 10-50 |
| 10.6.3 Selecting Options in the Title Bar Menu .....          | 10-51 |
| 10.6.4 Selecting Options in the Left Menu .....               | 10-51 |
| 10.6.5 Selecting Options in the Middle Menu .....             | 10-52 |
| 10.7 Automatic Updates of Class Browsers.....                 | 10-52 |

## 11. ERRORS AND BREAKS

|                                                |      |
|------------------------------------------------|------|
| 11.1 Error Handling Functions and Methods..... | 11-1 |
| 11.2 Error Messages.....                       | 11-5 |
| 11.1.1 Classes and Instances.....              | 11-6 |
| 11.1.2 Methods and Messages.....               | 11-7 |
| 11.1.3 Naming Objects .....                    | 11-8 |
| 11.1.4 Annotated and Active Values.....        | 11-9 |
| 11.1.5 Miscellaneous .....                     | 11-9 |

## 12. BREAKING AND TRACING

|                                         |      |
|-----------------------------------------|------|
| 12.1 Breaking and Tracing Methods ..... | 12-1 |
| 12.2 Breaking and Tracing Data .....    | 12-3 |

## 13. EDITING

|                            |      |
|----------------------------|------|
| 13.1 Editing Classes ..... | 13-1 |
|----------------------------|------|

## TABLE OF CONTENTS

---

|                              |      |
|------------------------------|------|
| 13.2 Editing Instances ..... | 13-5 |
|------------------------------|------|

## 14. FILE MANAGER

|                                          |       |
|------------------------------------------|-------|
| 14.1 Manipulating Files .....            | 14-1  |
| 14.2 Loading Files .....                 | 14-2  |
| 14.3 LOOPS File Manager Commands .....   | 14-3  |
| 14.4 Saving LOOPS Objects on Files ..... | 14-6  |
| 14.5 Storing Files .....                 | 14-10 |
| 14.6 Compiling Files.....                | 14-12 |

## 15. PERFORMANCE ISSUES

|                                    |      |
|------------------------------------|------|
| 15.1 Garbage Collection .....      | 15-1 |
| 15.2 Instance Variable Access..... | 15-1 |
| 15.3 Method Lookup .....           | 15-3 |
| 15.4 Cache Clearing .....          | 15-3 |

## 16. PROCESSES

## 17. READING AND PRINTING

|                                      |       |
|--------------------------------------|-------|
| 17.1 Reading Objects .....           | 17-1  |
| 17.2 Print Flags .....               | 17-2  |
| 17.3 Printing Classes .....          | 17-4  |
| 17.4 Printing Objects .....          | 17-8  |
| 17.5 Printing Active Values .....    | 17-11 |
| 17.6 Printing Methods .....          | 17-12 |
| 17.7 Unique Identifiers (UIDs) ..... | 17-14 |

## 18. USER INPUT/OUTPUT MODULES

|                                                           |       |
|-----------------------------------------------------------|-------|
| 18.1 Inspector .....                                      | 18-1  |
| 18.1.1 Overview of the User Interface .....               | 18-1  |
| 18.1.2 Using Instance Inspectors .....                    | 18-2  |
| 18.1.2.1 Titles of Instance Inspector Windows .....       | 18-2  |
| 18.1.2.2 Menu for the Title Bar .....                     | 18-3  |
| 18.1.2.3 Menu for the Left Column .....                   | 18-4  |
| 18.1.2.4 Menu for the Right Column .....                  | 18-6  |
| 18.1.3 Using Class Inspectors .....                       | 18-7  |
| 18.1.3.1 Titles of Class Inspector Windows .....          | 18-7  |
| 18.1.3.2 Menu for the Title Bar .....                     | 18-8  |
| 18.1.3.3 Menu for the Left Column .....                   | 18-8  |
| 18.1.3.4 Menu for the Right Column .....                  | 18-8  |
| 18.1.4 Using Class IVs Inspectors .....                   | 18-9  |
| 18.1.4.1 Titles of Class IVs Inspector Windows .....      | 18-9  |
| 18.1.4.2 Menu for the Title Bar .....                     | 18-9  |
| 18.1.4.3 Menu for the Left Column .....                   | 18-10 |
| 18.1.4.4 Menu for the Right Column .....                  | 18-10 |
| 18.1.5 Functional Interface for Instance Inspectors ..... | 18-11 |
| 18.1.6 Customizing the Inspector .....                    | 18-15 |
| 18.2 Extensions to ?= .....                               | 18-16 |
| 18.2.1 Message Sending .....                              | 18-16 |
| 18.2.2 Record Creation .....                              | 18-17 |

## 19. WINDOWS

|                                         |       |
|-----------------------------------------|-------|
| 19.1 The Class Window .....             | 19-1  |
| 19.2 Basic Window Methods .....         | 19-2  |
| 19.3 Prompt Windows .....               | 19-9  |
| 19.4 Mouse and Menu Functionality ..... | 19-14 |

## TABLE OF CONTENTS

---

|                                   |       |
|-----------------------------------|-------|
| 19.4.1 Menu Item Structure .....  | 19-17 |
| 19.4.2 Caching Menus .....        | 19-18 |
| 19.5 Subclasses of a Window ..... | 19-18 |
| 19.6 Lisp Windows .....           | 19-22 |

## 20. SYSTEM VARIABLES AND FUNCTIONS

|             |         |
|-------------|---------|
| INDEX ..... | INDEX-1 |
|-------------|---------|

|                |            |
|----------------|------------|
| GLOSSARY ..... | GLOSSARY-1 |
|----------------|------------|

Every object within the LOOPS system is an instance of some class. In this manual, however, the word instance generally refers to objects that are not themselves classes. Instances are a data type that contain local storage for instance variables, a pointer to the class that describes the instance, the Unique Identifier (UID), and other information.

This chapter describes naming and creating instances, accessing data stored within instances or pointed to by instances, and other related topics.

## 2.1 Instance Naming Conventions

A separate name space for LOOPS objects is maintained by the LOOPS system within a separate object name table. Since Lisp structures and LOOPS objects are stored in separate name tables, you can use the same symbol to refer to both a Lisp structure and a LOOPS object.

Note: The separate name space is not implemented by using the Common Lisp Package System.

Instances are not created with names; therefore, it may be necessary to keep pointers to them. Two ways are available to create pointers:

- Use Lisp variables, as in:

```
(SETQ window1 (← ($ Window) New))
```

This creates an instance of the class **Window** that can be referenced by the Lisp variable **window1**.

- Use a LOOPS name. This can be done in two ways:

- Assign a name at the same time the instance is created. This can be done by using

```
(← ($ Window) New 'window2)
```

as described above. This creates an instance of the class **Window** that can be referenced by the LOOPS expression `($ window2)`.

- Use the message **SetName** if you have a pointer to an object and want to assign a LOOPS name to that object.

The following table shows the items that manipulate LOOPS names.

| Name | Type              | Description                                                                                                              |
|------|-------------------|--------------------------------------------------------------------------------------------------------------------------|
| \$   | NLambda and Macro | Distinguishes between the Lisp value of a symbol and the LOOPS value of the same symbol; does not evaluate its argument. |

## 2.1 INSTANCE NAMING CONVENTIONS

---

|                            |          |                                                                                                                  |
|----------------------------|----------|------------------------------------------------------------------------------------------------------------------|
| <b>\$!</b>                 | Function | Distinguishes between the Lisp value of a symbol and the LOOPS value of the same symbol; evaluates its argument. |
| <b>SetName</b>             | Method   | Assigns a LOOPS name to an object.                                                                               |
| <b>UnSetName</b>           | Method   | Removes a name pointer to an object.                                                                             |
| <b>Rename</b>              | Method   | Changes the name of an object.                                                                                   |
| <b>GetObjectNames</b>      | Function | Returns the names of an object, including its UID.                                                               |
| <b>ErrorOnNameConflict</b> | Variable | Causes a break to occur when an attempting to name an object that already has a LOOPS name.                      |

---

**(\$ name)** [NLambda and Macro]

---

Purpose/Behavior: Returns a pointer to a LOOPS object specified by the LOOPS name *name*. If no object exists for *name*, NIL is returned.

Arguments: *name* A LOOPS name.

Returns: Pointer to a LOOPS object or NIL; see Behavior.

Example: Given that

```
24←(← ($ Window) New 'window2)
#, ($& Window (NEW0.1Y%:.;h.eN6 . 495))
```

then

```
25←($ window2)
#, ($& Window (NEW0.1Y%:.;h.eN6 . 495))
```

The returned value is a pointer to the new window instance. For a further explanation, see Chapter 18, Reading and Printing.

---

**(\$! name)** [Function]

---

Purpose/Behavior: Returns a pointer to an object specified by the value of the variable *name*, given that the value is a LOOPS name. If no object exists for *name*, NIL is returned.

Arguments: *name* Evaluates to a valid LOOPS name.

Returns: Pointer to a LOOPS object or NIL; see Behavior.

Example: Given that

```
26←(SETQ foo 'Window)
Window
```

and **Window** is a LOOPS object, then

```
27←($! foo)
#, ($C Window)
```

---

**(← self SetName name)** [Method of Object]

---

Purpose: Assigns a LOOPS name to an object.

Behavior: If *name* is NIL, then a break occurs. If *name* is not a symbol, a break occurs. If *name* is already in use as a LOOPS name, and if the variable



*ErrorOnNameConflict* is non-NIL, then a break occurs, giving you the chance to OK "rebinding" *name*.

Note: If an object has multiple names, (*← self* **SetName** *NewName*) results in both the old name and new name appearing when (FILES?) is executed. The instance is also printed twice on the file if both names are specified to be saved.

Arguments: *self* An object.  
*name* The LOOPS name to be given to the object; must be a symbol.  
 Returns: *self*  
 Categories: Object  
 Specializations: Class

Example: Given the commands

```
28←(SETQ window1 (← ($ Window) New))
#,$& Window (NEW0.1Y%.:;h.eN6 . 496))
```

```
29←(← window1 SetName 'window3)
#,$& Window (NEW0.1Y%.:;h.eN6 . 496))
```

the Lisp variable **window1** and the LOOPS expression

```
($ window3)
```

now point to the same object.

---

(*← self* **UnSetName** *name*) [Method of Object]

---

Purpose: Removes a LOOPS name pointer to an object.  
 Behavior: Removes the reference of *name* to *self* from the object name table maintained by the LOOPS system. If *name* is NIL, all names pointing to *self* in the object name table are removed from the files on **FILELST**. If *name* is non-NIL and the instance is associated with any files on **FILELST**, the instance is removed from those files. If *name* is not a valid LOOPS name for the object in question, an error occurs.  
 Arguments: *self* An object.  
*name* A LOOPS name.  
 Returns: Used for side effect only.  
 Categories: Object

---

(*← self* **Rename** *newName* *oldNames*) [Method of Object]

---

Purpose: Changes the name of an object.  
 Behavior: If *oldNames* is NIL, removes all old names when *newName* is installed as the name for *self*; otherwise replaces only names specified in *oldNames* by *newName*. If *oldNames* is not a valid LOOPS name for the object in question, an error occurs.  
 Arguments: *self* Evaluates to a LOOPS name.  
*newName* The LOOPS name to be given to the object; must be a symbol.

## 2.1 INSTANCE NAMING CONVENTIONS

---

|                  |                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                 |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
|                  | <i>oldNames</i>                                                                                                                                                                                                                                                                                                                       | List of symbols whose names are to be removed; if NIL, all old names are removed when <i>newName</i> is installed as the name for <i>self</i> . |
| Returns:         | <i>self</i>                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                 |
| Categories:      | Object                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                 |
| Specializations: | Class                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                 |
| Example:         | Examine the following expressions to see the effects of <b>Rename</b> .<br><br>30←(\$ window2)<br>#,\$(& Window (NEW0.1Y%:.H53.G2A . 496))<br><br>31←(← (\$ window2) Rename 'MyWindow')<br>#,\$(& Window (NEW0.1Y%:.H53.G2A . 496))<br><br>32←(\$ window2)<br>NIL<br><br>33←(\$ MyWindow)<br>#,\$(& Window (NEW0.1Y%:.H53.G2A . 496)) |                                                                                                                                                 |

| (GetObjectNames <i>object</i> ) |                                                                                                                                                                                | [Function]      |
|---------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|
| Purpose/Behavior:               | Returns the names of <i>object</i> , including its UID.                                                                                                                        |                 |
| Arguments:                      | <i>object</i>                                                                                                                                                                  | A LOOPS object. |
| Returns:                        | The names of <i>object</i> , including its UID.                                                                                                                                |                 |
| Example:                        | The command<br><br>(PROGN<br>(← (\$ Window) New 'w1)<br>(← (\$ w1) SetName 'wlagain)<br>(GetObjectNames (\$ w1)))<br><br>returns<br><br>(wlagain w1 (NEW0.1Y%:.H53.G2A . 497)) |                 |

| ErrorOnNameConflict | [Variable]                                                                                                                                                                                                                                                                                                                          |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose/Behavior:   | Behavior depends on the value. <ul style="list-style-type: none"><li>• If NIL, the existing object is replaced by a new object.</li><li>• If non-NIL, a break occurs when an attempt is made to give an object a name that is already in use as a LOOPS name.</li></ul> Initially, the value for <b>ErrorOnNameConflict</b> is NIL. |

---

## 2.2 CREATING INSTANCES

---

| 2.2 Creating Instances                                                                                                                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| When an instance is created by sending the <b>New</b> message to a class, the default behavior for <b>Class.New</b> is to send the message <b>NewInstance</b> to the |

newly created object. If you require that special or additional operations occur at instance creation time, specialize the method **NewInstance**. Specializations of the **NewInstance** method should return *self*. You also have the capability to pass arguments to the **NewInstance** method when the **New** message is sent to create the instance. For example, the following defines a class **NamedClass** which adds the instance variable **name** and specializes **New** to set that instance variable to the name of the instance when created.

```
(DefineClass 'NamedClass)
(← ($ NamedClass) AddIV 'name)
(DefineMethod ($ NamedClass) 'New ' (self name)
' (←@ (←self NewInstance name) name name))
```

You can also indicate whether instances are to be saved on files using the File Manager, which is described in Chapter 14, File Manager.

The following table shows the methods in this section.

| Name                 | Type   | Description                                                   |
|----------------------|--------|---------------------------------------------------------------|
| <b>New</b>           | Method | Creates a new object of a particular class.                   |
| <b>←New</b>          | Macro  | Creates an object and sends a message to it.                  |
| <b>NewInstance</b>   | Method | Allows initialization of newly created instances by class.    |
| <b>NewWithValues</b> | Method | Creates an object with instance variables of assigned values. |

(← class **New** name arg1 arg2 ...) [Method of Class]

Purpose: Creates a new object, which is an instance of the class *class*.

Behavior: Creates a new instance *name* and then sends the message (← "the new instance" **NewInstance** *name* arg1arg2 ...)

In the default case, the **New** method uses the default values for the instance variable values in the newly created instance. These default values are given in the instance variable descriptions of the given class. When that process is finished, the instance can be altered in various ways by sending it messages. Specializations of the **New** method should return the new instance, and can take more arguments after *name*.

The internal data structure of an instance contains a pointer to the class of which it is an instance.

Arguments: *class* Pointer to a class.

*name* Name assigned to the instance; if NIL, object does not have a LOOPS name.

*arg1arg2...* Arguments passed to the **NewInstance** method.

Returns: Newly created instance of the class.

Categories: Class

Specializations: AbstractClass, MetaClass

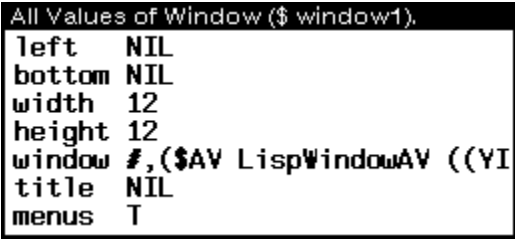
Example: The following command creates a new instance named **window1** of class **Window**.

```
20← (← ($ Window) New 'window1)
#, ($& Window (NEW0.1Y%.:.;h.eN6 . 515))
```

The command

```
21← (INSPECT (← ($ Window) New))
```

results in the following inspector window:



```
All Values of Window ($ window1).
left NIL
bottom NIL
width 12
height 12
window #, ($AV LispWindowAV ((YI
title NIL
menus T
```

Some of the values assigned to the various instance variables are default values. These values are defined in the class **Window**.

---

**(←New class selector args)**

[Macro]

Purpose: Creates an instance and sends a message to it within one form.

←**New** is pronounced "send new."

Behavior: Is equivalent to the form

```
(← (← class New) selector args)
```

Arguments: *class* Evaluates to a class.

*selector* Name of the message to be sent to the new instance.

*args* Arguments to be sent to the function invoked by the message.

Returns: The new instance.

Example: The command

```
23← (←New ($ Window) Open)
```

creates a new instance of the class **Window** and then sends the message **Open** to the newly created object.

---

**(← self NewInstance name arg1 arg2 arg3 arg4 arg5)**

[Method of Object]

Purpose: Allows initialization of newly created instances by the class of the instance, as opposed to the metaclass. Subclasses of **Object** that specialize this method should have a ←**Super** form within the method to allow the execution of the default behavior.

Behavior: Not normally called directly, but is sent by method **New**. The default behavior is as follows.

If *name* is non-NIL, the message **SetName** is sent to *self*.

Within *self*, instance variables that are bound to the value of **NotSetValue** and have an **:initForm** property in the class description are filled. This allows you to override the **:initForm** behavior by setting values for instance variables before executing the ←**Super** form. See the discussion of **:initForm** in Section 2.3, "Data Storage in Instances at Creation Time."

Sends the message **SaveInstance** to *self* with the argument *name*.

Note: Specializations of the **NewInstance** method should return *self*.

|                  |                                   |                                                                    |
|------------------|-----------------------------------|--------------------------------------------------------------------|
| Arguments:       | <i>self</i>                       | Evaluates to a class.                                              |
|                  | <i>name</i>                       | LOOPS name given to a new instance.                                |
|                  | <i>arg1...arg5</i>                | Optional arguments referenced by user-written specialization code. |
| Returns:         | LOOPS name of new object created. |                                                                    |
| Categories:      | Object                            |                                                                    |
| Specializations: | IndexedObject                     |                                                                    |

(← *class* **NewWithValues** *valDescriptionList*)

[Method of Class]

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                                                                   |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:         | Creates a new object and initializes the instance variables specified in <i>valDescriptionList</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                   |
| Behavior:        | Creates the object with no other initialization, directly installs the values and property lists specified in <i>valDescriptionList</i> , and returns the created object. Variables that have no description in <i>valDescriptionList</i> are given no value in the instance and thus inherit the default value from the class.<br><br><b>NewWithValues</b> does not invoke the <b>NewInstance</b> method or the <b>:initForm</b> properties (see Section 2.3, "Data Storage in Instances at Creation Time"). This means that the instance is not recognized by the File Manager; to be recognized, the instance must be named. |                                                                                                                                                                                                                                                                   |
| Arguments:       | <i>class</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | Pointer to a class.                                                                                                                                                                                                                                               |
|                  | <i>valDescriptionList</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | Evaluates to a list of value descriptions, each of which is a list of variableNames and properties, for example,<br><br>$((\text{VarName1 value1 prop1a propVal1a prop1b propVal1b ...})$ $(\text{VarName2 value2 prop2a propVal2a prop2b propVal2b ...}) \dots)$ |
| Returns:         | The created object.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                   |
| Categories:      | Class                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                   |
| Specializations: | MetaClass                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                   |
| Example:         | The command                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                   |

```
22←(INSPECT (← ($ Window) NewWithValues '((width 300)(height 200))))
```

results in the following inspector window:

```
All Values of Window ($ (MWX0.;F5.o28.Z;
left NIL
bottom NIL
width 300
height 200
window #,($AV LispWindowAV ((YI
title NIL
menus T
```

Contrast the values for the instance variables width and height with the inspector window for **New**, above.

## 2.3 DATA STORAGE IN INSTANCES AT CREATION TIME

### 2.3 DATA STORAGE IN INSTANCES AT CREATION TIME

## 2.3 Data Storage in Instances at Creation Time

When an instance is first created, the value of the variable **NotSetValue** is assigned to its instance variables. **NotSetValue** is initialized to be an active value of the class **NotSetValue** and should not be changed by the user. Trying to access an instance variable triggers this active value which in turn triggers the method **IVValueMissing**.

Data is stored in instances on all Puts and on **GetValues** when the default value is an active value but not **NotSetValue**. Be aware that in reading the value of an instance variable that is not stored in the instance, changes in the default value of the instance variable in the class description are seen in accesses of the instance.

One exception to this method of data storage at creation time is if an instance variable has the property **:initForm** in the class description. In this case, data is stored in the instance at the time of creation.

Testing for whether data is stored locally in the instance can be done in two ways:

- Through the user interface, you can inspect an instance in the local mode. (See Chapter 18, User Input/Output Modules, for more information.) Values not locally stored appear as #,NotSetValue.
- Programmatically, through the function **GetIVHere** with the macro **NotSetValue**.

The following table describes the items in this section.

| Name                  | Type        | Description                                                                                                          |
|-----------------------|-------------|----------------------------------------------------------------------------------------------------------------------|
| <b>IVValueMissing</b> | Method      | Handles cases when an attempt is made to access the value of an instance variable that is not stored in an instance. |
| <b>NotSetValue</b>    | Macro       | Determines if its argument is equivalent to the value of <b>NotSetValue</b> .                                        |
| <b>:initForm</b>      | IV Property | Signals a property value that can be evaluated.                                                                      |

(← *self* **IVValueMissing** *varName propName typeFlg newValue*) [Method of Object]

**Purpose:** Invoked by the system to handle the cases when you try to access the value of an instance variable that is not stored in an instance. This is the mechanism the system uses to access default values.

**Behavior:** Varies according to the functionality that invoked it.

- **GetValueOnly** accesses return the default value of the instance variable stored in the class.
- **GetValue** accesses return the default value of the instance variable stored in the class if it is not an active value. If the default value is an active value, a copy of the active value is made, stored in the instance, and sent the **GetWrappedValue** message.
- **PutValueOnly** accesses store the new value in the instance.
- **PutValue** accesses store the new value in the instance unless the default value of the instance variable stored in the class is an active value. If this is the case, a copy of the active value is made, stored in the instance, and sent the **PutWrappedValue** message.

**Arguments:** *varName* Instance variable name.

*propName* Property name for instance variable *varName*.

*typeFlg* Used internally to indicate the type of access.

*newValue* If called by **PutValueOnly** or **PutValue**, this is the value to be placed into the instance variable or property name.

Returns: Value depends on the functionality that invoked this method; see Behavior.

Categories: Object

**(NotSetValue arg)**

[Macro]

Purpose: Determines if *arg* is **EQ** to the value of **NotSetValue**.

Arguments: *arg* Any value.

Returns: NIL or T.

Example: Given that

```
51←(← ($ Window) New 'w)
#,($& Window (NEW0.1Y%.:;h.eN6 . 515))

then

52←(NotSetValue (GetIVHere ($ w) 'title))
T
```

**:initForm**

[IV property]

Purpose: This allows instance variables to be initialized at the time of the creation of an instance. The **:initForm** property and its value are in the class definition. Its value is a form that is evaluated when an instance is created. The result of the evaluation is stored as the value of the instance variable containing this property in the newly created instance.

This behavior does not hold if the value of the instance variable is not **NotSetValue**. Refer to the method **Object.NewInstance** in Section 2.2, "Creating Instances," for more information.

Example: Given the commands

```
53←(DefineClass 'testclass)
#,($C testclass)

54←(AddCIV ($ testclass) 'date NIL '(:initForm| (DATE)))
date

then

55←(INSPECT (← ($ testclass) New))
```

returns the following inspector window:

```
All Values of testclass ($ (MWX0.:F
date "30-Mar-88 13:53:37")
```

## 2.4 CHANGING THE NUMBER OF INSTANCE VARIABLES IN AN INSTANCE

### 2.4 CHANGING THE NUMBER OF INSTANCE VARIABLES IN AN INSTANCE

## 2.4 Changing the Number of Instance Variables in an Instance

An instance can contain more instance variables than are defined in the class that describes it. It is not possible to remove an instance variable from an instance if the instance variable is defined in the class.

When you try to access the value of an instance variable that is not defined as an instance variable in the instance, the **IVMissing** method is invoked.

The following table shows the functions and methods in this section.

| Name                  | Type     | Description                                                                                                                |
|-----------------------|----------|----------------------------------------------------------------------------------------------------------------------------|
| <b>AddIV</b>          | Function | Adds an instance variable to an instance.                                                                                  |
| <b>AddIV</b>          | Method   | Adds an instance variable to <i>self</i> .                                                                                 |
| <b>DeleteIV</b>       | Function | Removes an instance variable or property from an instance.                                                                 |
| <b>DeleteIV</b>       | Method   | Removes an instance variable or property from <i>self</i> .                                                                |
| <b>ConformToClass</b> | Method   | Makes <i>self</i> contain only those instance variables that are defined or inherited by the class of <i>self</i> .        |
| <b>IVMissing</b>      | Method   | Is sent by the system when an attempt is made to access an instance variable that does not exist. It is used for recovery. |

**(AddIV self name value propName)** [Function]

Purpose: Adds an instance variable to an instance.

Behavior: Varies according to the arguments.

- If *propName* is non-NIL and if *name* already exists, it is added as a property to the instance variable *name* with the value *value*.
- If *name* already exists, and if *propName* is NIL, the value of the instance variable *name* is changed to *value*.
- If *name* does not exist and if *propName* is non-NIL, the instance variable *name* is added to the instance and given the value of the variable **NotSetValue**. It is given the property *propName* with the value *value*.
- If *name* and *propName* already exist, the value of the property *prop* is changed to *value*.

Arguments:

|                 |                                                       |
|-----------------|-------------------------------------------------------|
| <i>self</i>     | A pointer to the instance.                            |
| <i>name</i>     | The name of the instance variable to be added.        |
| <i>value</i>    | The value the new instance variable will be assigned. |
| <i>propName</i> | Property name of instance variable name; may be NIL.  |

Returns: Used for side effect only.

Example: Given that

```
55←(← ($ Window) New 'w)
```

the command

```
56←(AddIV ($ w) 'left 1234)
```

changes the value of the instance variable **left** to 1234. The command



```
57← (AddIV ($ w) 'foo 1234)
```

adds the instance variable **foo** to (\$ w) and gives it the value 1234.

(← *self* **AddIV** *name* *value* *propName*)

[Method of Object]

Purpose: Adds an instance variable to *self*.

Behavior: Method form of the function **AddIV**.

Arguments: See the function **AddIV**.

Returns: NIL

Categories: Object

Specializations: Class

Example: Given that

```
58← (← ($ Window) New 'w)
```

the command

```
59← (← ($ w) AddIV 'left 1234)
```

changes the value of the instance variable **left** to 1234. The command

```
60← (← ($ w) AddIV 'foo 1234)
```

adds the instance variable **foo** to (\$ w) and gives it the value 1234.

(**DeleteIV** *self* *varName* *propName*)

[Function]

Purpose: Removes an instance variable or property from an instance.

Behavior: Varies according to the arguments.

- If *self* does not have *varName*, an error occurs.
- If *varName* is defined in the class or a super class of *self*, an error occurs.
- If the instance *self* has *varName*, and *propName* is NIL, the instance variable is deleted.
- If *propName* is non-NIL, it is deleted only if it is a locally stored property, that is, not defined in a class. If *propName* is not a property of *varName* or is defined in a class, no error occurs.

Arguments: *self* A pointer to the instance from which the instance variable is to be deleted.

*varName* The name of the instance variable to be deleted.

*propName* If non-NIL, specifies that a property, not an instance variable, is to be deleted.

Returns: If no errors occur, this returns *self*.

Example: The following command deletes the instance variable **foo** from (\$ w):

```
62← (DeleteIV ($ w) 'foo)
```

---

(← *self* **DeleteIV** *varName propName*)

[Method of Object]

Purpose: Deletes an instance variable or property from *self*.

Behavior: Method version of the function **DeleteIV**.

Arguments: See the function **DeleteIV**.

Returns: If no errors occur, this returns *self*.

Categories: Object

---

(← *self* **ConformToClass**)

[Method of Object]

Purpose/Behavior: Makes *self* contain only those instance variables that are defined in or inherited by the class of *self*.

Returns: NIL

Categories: Object

Example: This example adds an instance variable to an instance and shows how **ConformToClass** removes it.

```
63←(← ($ Window) New 'w1)
(#, ($& Window (|MXWO.:F5.G18.Z:?.18))

64←(← ($ w1) AddIV 'NewIV 1234)
1234

65←(INSPECT ($ w1))
```

This produces the following inspector window:

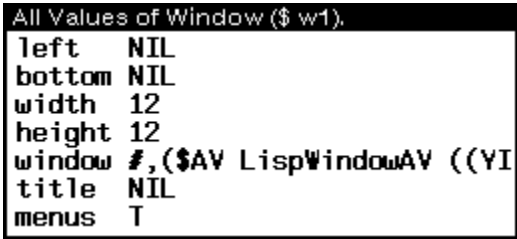


```
All Values of Window ($ w1).
left NIL
bottom NIL
width 12
height 12
window #, ($AV LispWindowAV ((YI
title NIL
menus T
NewIV 1234
```

```
66←(← ($ w1) ConformToClass)
NIL

67←(INSPECT ($ w1))
```

This produces the following inspector window:



```
All Values of Window ($ w1).
left NIL
bottom NIL
width 12
height 12
window #, ($AV LispWindowAV ((YI
title NIL
menus T
```

---

(← *self* **IVMissing** *varName propName typeFlg newValue*)

[Method of Object]

Purpose: This message is sent by the system when an attempt is made to access an instance variable that does not exist. It is used for recovery.

Behavior: Varies according to the arguments.

- If the instance variable *varName* is now defined in the class, copy it to *self*. This can happen if the class was changed after the instance was created.
- If there is a class variable with the name *varName*, use it. The method of use is determined by the **:allocation** class variable property:

- `dynamicCached`

Copy the class variable to *self* on puts or gets.

- `dynamic`

Copy the class variable to *self* on puts. If the access is by **GetValue** or **GetValueOnly**, then get the value from the class. The value retrieved from the class is dependent on the value of *propName* and the class variable property **:initform**. If *propName* is NIL and there is a class variable property **:initform**, then retrieve the value returned from evaluating **:initform**. Otherwise, retrieve the value of the class variable *varName* if *propName* is NIL or the value of the property *propName* if it is non-NIL.

- `class` (the default if there is no **:allocation** property)

Do not copy the class variable *varName* to *self*. On puts, store the value in the class. With gets, do the same as the case when the **:allocation** property is dynamic. Essentially, this allows you to access class variables with the same syntax as instance variables.

An attempt is made to correct the spelling of *varName* and try the above steps again before breaking.

Arguments: *self* A pointer to the instance.

*varName* Instance variable name for *self*.

*propName* Property name of instance variable *varName*.

*typeFlg* One of **PutValue**, **PutValueOnly**, **GetValue**, **GetValueOnly**.

*newValue* Value to be stored in *varName*.

Returns: If doing a put, this returns *NewValue*; else this returns the value of the instance variable name.

Categories: Object

Example: If **w1** is a **Window**, then the following command breaks under **Object.IVMissing** because windows do not have an instance variable named **mumble**.

```
(← ($ w1) Get 'mumble)
```

## 2.5 MOVING VARIABLES

## 2.5 MOVING VARIABLES

## 2.5 Moving Variables

These functions allow you to move variables between classes.

| Name                  | Type     | Description                         |
|-----------------------|----------|-------------------------------------|
| <b>RenameVariable</b> | Function | Changes a variable name in a class. |

|                          |          |                                                       |
|--------------------------|----------|-------------------------------------------------------|
| <b>MoveVariable</b>      | Function | Moves an instance variable from one class to another. |
| <b>MoveClassVariable</b> | Function | Moves an class variable from one class to another.    |

---

**(RenameVariable** *className oldVarName newVarName classVarFlg*) [Function]

|            |                                                                                                                                                                                                                                 |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Changes <i>oldVarName</i> to <i>newVarName</i> in class <i>className</i> .                                                                                                                                                      |
| Behavior:  | Can cause inconsistency without warning; does not test for references to the variable in methods of <i>className</i> .                                                                                                          |
| Arguments: | <i>className</i> Class in which function is defined.<br><i>oldVarName</i> Old name of variable.<br><i>newVarName</i> New name of variable.<br><i>classVarFlg</i> If not NIL, then <i>oldVarName</i> refers to a class variable. |
| Returns:   | If successful, returns <i>newVarName</i> ; else NIL.                                                                                                                                                                            |
| Example:   | The following command renames the class variable <b>OldVar</b> to <b>NewVar</b> .<br><pre>27← (RenameVariable (\$ MyClass) 'OldVar 'NewVar T)</pre>                                                                             |

---

**(MoveVariable** *oldClassName newClassName varName*) [Function]

|            |                                                                                                                                                |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Moves an instance variable from <i>oldClassName</i> to <i>newClassName</i> .                                                                   |
| Behavior:  | Moves both the <i>varName</i> instance variable and its description to <i>newClassName</i> . Deletes <i>varName</i> from <i>oldClassName</i> . |
| Arguments: | <i>oldClassName</i> Source class.<br><i>newClassName</i> Destination class.<br><i>varName</i> Variable to be moved.                            |
| Returns:   | Used for side effect only.                                                                                                                     |

---

**(MoveClassVariable** *oldClassName newClassName varName*) [Function]

|            |                                                                                                                                           |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Moves a class variable from <i>oldClassName</i> to <i>newClassName</i> .                                                                  |
| Behavior:  | Moves the class variable <i>varName</i> and its properties to <i>newClassName</i> . Deletes <i>varName</i> from the <i>oldClassName</i> . |
| Arguments: | <i>oldClassName</i> Source class.<br><i>newClassName</i> Destination class.<br><i>varName</i> Class variable to be moved.                 |
| Returns:   | Used for side effect only.                                                                                                                |

## 2.6 DESTROYING INSTANCES

---

## 2.6 Destroying Instances

---

A protocol allows you to customize the behavior of the system at instance destruction time. The naming convention is somewhat asymmetrical to that of creation time. To programmatically influence instance creation, specialize the method **NewInstance**. To programmatically influence instance destruction, specialize the method **Destroy**. Include a (**←Super**) in specializations of **Destroy** to guarantee normal system behavior.

The following table describes the methods in this section.

| Name                   | Type   | Description                                                                                        |
|------------------------|--------|----------------------------------------------------------------------------------------------------|
| <b>Destroy</b>         | Method | Removes an object from the environment.                                                            |
| <b>Destroy!</b>        | Method | Removes an object from the environment. If the object is a class, it also destroys all subclasses. |
| <b>DestroyInstance</b> | Method | Modifies the data structure of an instance as described above.                                     |

---

(**← self Destroy**) [Method of Object]

---

Purpose: Removes an object from the environment.

Behavior: Sends the **DestroyInstance** message with *self* as an argument to the class of *self*. **UnmarkedAsChanged** is called to remove the instance from the notice of the File Manager.

Arguments: *self*          A pointer to the instance.

Returns: Used for side effect only.

Categories: Object

Specializations: Class, DestroyedClass, IndexedObject, Window

Example: The following command destroys an instance named **window1**.

```
70← (← ($ window1) Destroy)
```

---

(**← self Destroy!**) [Method of Object]

---

Purpose/Behavior: Removes an object from the environment. If the object is a class, it also destroys all subclasses.

Arguments: *self*          A pointer to the instance.

Returns: Used for side effect only.

Categories: Object

Specializations: Class, DestroyedClass, DestroyedObject

---

(**← class DestroyInstance instance**) [Method of Class]

---

Purpose/Behavior: Destroys *instance* by overwriting its contents. When an instance is destroyed, several things occur:

- The instance is removed from any files on **FILELST**. See the *Interlisp-D Reference Manual*.
- The instance is deleted from system hash tables used for maintaining object identities.
- The class of the instance is changed to **DestroyedObject**.
- Other fields of the internal instance data structure are set to NIL.

If an instance is only pointed to by a LOOPS name, its data structure is freed for garbage collection.

Arguments: *class*        Class of *instance*.  
*instance*      The instance being destroyed.

Returns: Used for side effect only.

Categories: Class

Specializations: MetaClass, DestroyedClass

---

## 2.7 METHODS CONCERNING THE CLASS OF AN OBJECT

---

### 2.7 METHODS CONCERNING THE CLASS OF AN OBJECT

---

---

## 2.7 Methods Concerning the Class of an Object

---

Given an instance, you often need to manipulate the class of an instance. This section describes how to perform this manipulation.

| Name               | Type     | Description                                                                   |
|--------------------|----------|-------------------------------------------------------------------------------|
| <b>ChangeClass</b> | Method   | Changes the class of an instance.                                             |
| <b>Class</b>       | Macro    | Determines the class of an object.                                            |
| <b>Class</b>       | Method   | Determines the class of an object.                                            |
| <b>ClassName</b>   | Function | Returns the class name of an object.                                          |
| <b>ClassName</b>   | Method   | Returns the class name of an object.                                          |
| <b>InstOf</b>      | Method   | Determines if <i>self</i> is an instance of a class.                          |
| <b>InstOf!</b>     | Method   | Determines if <i>self</i> is an instance of a class or any of its subclasses. |

You can also compute a class corresponding to a Lisp data type for Lisp objects by using **GetLispClass**, described in Chapter 4, Metaclasses.

---

(← *self* **ChangeClass** *newClass*) [Method of Object]

---

Purpose: Changes the class of an instance.

Behavior: Creates a blank instance of the *newClass*. Any instance variables that are locally stored within *self* are added to the new instance.

---

If *newClass* is not the name of a class or a pointer to the class, an error occurs.

Arguments: *self* A pointer to an instance.  
*newClass* Either the name of a class or a pointer to the class.

Returns: *self*

Categories: Object

Specializations: IndexedObject

Example: Create an instance of class **Window** and assign a local value to the instance variable **width** - all other instance variables of (\$ w) lack local values. Then, when the class of (\$ w) is changed to **IndirectVariable**, (\$ w) will have all of the instance variables of its new class, plus the one instance variable of its old class which had a local value, **width**.

```
71←(← ($ Window) New 'w)
#,$& Window (NEW0.1Y%.:;h.eN6 . 501))

72←(←@ ($ w) width 123)
123

73←(← ($ w) ChangeClass 'IndirectVariable)
#,$& IndirectVariable (NEW0.1Y%.:;h.eN6 . 502))

74←(← ($ w) Inspect)
```

This produces the following inspector window:

| All Values of Indi |     |
|--------------------|-----|
| object             | NIL |
| varName            | NIL |
| propName           | NIL |
| type               | NIL |
| width              | 123 |

---

(Class *self*)

[Macro]

Purpose: Determines the class of an object.

Behavior: If *self* is a LOOPS object, return its class.  
If *self* is not a LOOPS object, evaluate (**GetLispClass** *self*)

Arguments: *self* A pointer to a LOOPS or Lisp object.

Returns: Value depends on the arguments; see Behavior.

Example: Given that

```
75←(← ($ Window) New 'window1)
#,$& Window (NEW0.1Y%.:;h.eN6 . 503))

then

76←(Class ($ window1))
#,$C Window)
```

Note: If *self* is an annotated value, the method **Class** and the macro **Class** return different values. See Chapter 8, Active Values, for more information on annotated values.

---

(← *self* **Class**)

[Method of Object]

Purpose/Behavior: Method version of the macro **Class**.

Arguments: *self* A pointer to a LOOPS object or a Lisp data structure.

Returns: Value depends on the arguments; see Behavior of the macro **Class**.

Categories: Object

Example: Given that

```
77←(← ($ Window) New 'window1)
#,$& Window (NEW0.1Y%.:;h.eN6 . 504))

then

78←(← ($ window1) Class)
#,$C Window
```

---

(**ClassName** *self*)

[Function]

Purpose: Returns the class name of the class of the object.

Behavior: Varies according to the argument.

- If *self* is a class, this returns the name of that class.
- If *self* is an instance, this returns the name of the class that describes that instance.
- If *self* is neither of these, an attempt is made to get the class of *self* by applying the function **GetLispClass** to *self*. If this returns NIL, the function **LoopsHelp** is called with the arguments *self* and "has no class name."

Arguments: *self* Can have multiple values; see Behavior.

Returns: Value depends on the argument; see Behavior.

Example: The command

```
80←(ClassName ($ Window))

returns

Window
```

---

(← *self* **ClassName**)

[Method of Object]

Purpose/Behavior: Method version of the function **ClassName**.

Arguments: See the function **ClassName**.

Returns: Value depends on the arguments; see Behavior of the function **ClassName**.

Categories: Object

---

(← *self* **InstOf** *class*)

[Method of Object]

Purpose/Behavior: Determines if *self* is an instance of *class*.

Arguments: *self* A pointer to an instance.

*class* A symbol name of a class or a pointer to a class.



Returns: T or NIL

Categories: Object

Example: Given that

```

83←(← ($ Window) New 'w1)
#, ($& Window (NEW0.1Y%:.;h.eN6 . 505))

then

84←(← ($ w1) InstOf 'Window)
T

85←(← ($ w1) InstOf ($ Window))
T

```

(← *self* **InstOf!** *class*) [Method of Object]

Purpose: Determines if *self* is an instance of *class* or any of *class*'s subclasses.

Behavior: Tests if class of *self* is a subclass of *class*.

Arguments: *self* A pointer to an instance.  
*class* A symbol name of a class or a pointer to a class.

Returns: Object

Categories: Object

2.8 COPYING INSTANCES

## 2.8 COPYING INSTANCES

## 2.8 Copying Instances

This section describes the methods for copying instances.

| Name               | Type   | Description                                                                                                                                                    |
|--------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CopyDeep</b>    | Method | Copies all nested objects, annotated values, and lists.                                                                                                        |
| <b>CopyShallow</b> | Method | Creates a new instance of the same class as <i>oldInstance</i> . Fills the instance variables of the new instance with the data contained in the old instance. |

(← *oldInstance* **CopyDeep** *newObjAList*) [Method of Object]

Purpose: Copies all nested objects, annotated values, and lists. All other values are shared, not copied. This method is similar to the Interlisp function **COPYALL**.

Behavior: Creates a new instance of the same class as *oldInstance*. Fills the instance variables of the new instance with copies of lists, active values, and instances pointed to by *oldInstance*.

Arguments: *oldInstance* A pointer to an instance.  
*newObjAList*

An association list of old copied objects with their associated copies; used to allow copying of circular structures. Users typically let this argument default to NIL.

Returns: The value of the new instance.

Categories: Object

Example: Create the class **CopyTest** with the following structure:

```
SEdit CopyTest Package: INTERLISP
((MetaClass Class Edited%: ; Edited 22-Mar-88
 ; 11:07 by jrb:
)
 (Supers Object) (ClassVariables)
 (InstanceVariables (var NIL)
 (list NIL)
 (instance NIL))
 (MethodFns))
```

Create the instance **CopyTest1** and initialize it as shown in the following inspector:

```
All Values of CopyTest ($ CopyTest1).
var 123
list (ABC)
instance #,($ CopyTest2)
```

Now create a copy and inspect it.

```
(INSPECT (SETQ DeepCopy (← ($ CopyTest1) CopyDeep)))
```

```
All Values of CopyTest ($ (MUX0.:F5.G18.?7C . 517)).
var 123
list (ABC)
instance #,($ CopyTest (MUX0.?:F5.G18.?7C . 518))
```

The value of the instance variable **instance** is different. Also,

```
(EQ (@ ($ CopyTest1) list) (@ DeepCopy list))
```

returns NIL.

(← *oldInstance* **CopyShallow**)

[Method of Object]

Purpose/Behavior: Creates a new instance of the same class as *oldInstance*. Fills the instance variables of the new instance with the data contained in the old instance.

Arguments: *oldInstance*  
A pointer to an instance.

Returns: A copy filled with the values shared by the instances.

Categories: Object

Example: Compare this example to the **CopyDeep** example above. Use the same **CopyTest1** instance as above.

```
(INSPECT (SETQ ShallowCopy (← ($ CopyTest1) CopyShallow)))
```

```

All Values of CopyTest ($ CopyTest1).
var 123
list (ABC)
instance #, ($ CopyTest2)

```

The value of the instance variable **instance** is the same. Also,

```
(EQ (@ ($ CopyTest1) list) (@ ShallowCopy list))
```

returns T.

## 2.9 QUERYING STRUCTURE OF INSTANCES

## 2.9 QUERYING STRUCTURE OF INSTANCES

## 2.9 Querying Structure of Instances

At run time, user-written code may need to determine the structure of some object which has been passed into it. This section describes the methods to do this.

| Name                  | Type   | Description                                                                                            |
|-----------------------|--------|--------------------------------------------------------------------------------------------------------|
| <b>HasCV</b>          | Method | Determines if a class variable can be accessed via <i>self</i> .                                       |
| <b>HasIV</b>          | Method | Determines if an instance variable can be accessed via <i>self</i> .                                   |
| <b>Inspect</b>        | Method | Inspects <i>self</i> as a class or instance.                                                           |
| <b>ListAttribute</b>  | Method | Determines instance variable or instance variable property names contained in an instance.             |
| <b>ListAttribute!</b> | Method | Recursively determines instance variable or instance variable property names contained in an instance. |
| <b>WhereIs</b>        | Method | Searches the supers hierarchy to find a class where a specified name is defined.                       |

(← *self* **HasCV** *cvName* *propName*) [Method of Object]

Purpose: Returns T if the class variable *cvName* (or its property *propName* if it is non-NIL) can be accessed via *self*; otherwise NIL.

Behavior: Sends the message **HasCV** to the class of *self* passing the arguments *cvName* and *propName*.

Arguments: *self*            A pointer to an instance or a class.  
*cvName*            Class variable name  
*propName*        Property name for class variable *cvName*.

Returns: T or NIL; see Behavior.

Categories: Object

Specializations: Class

Example: The following command checks if an instance **window1** has the class variable **RightButtonItem**:

```
87← (← ($ window1) HasCV 'RightButtonItem)
T
```

---

(← *self* **HasIV** *ivName* *propName*)

[Method of Object]

Purpose/Behavior: Returns T if the instance variable *ivName* (or its property *propName* if it is non-NIL) can be accessed via *self*; otherwise NIL.

Arguments: *self* A pointer to an instance or a class.  
*ivName* Instance variable name.  
*propName* Property name for instance variable *ivName*.

Returns: T or NIL; see Behavior.

Categories: Object

Specializations: Class

---

(← *self* **Inspect** *INSPECTLOC*)

[Method of Object]

Purpose/Behavior: Inspects *self* as a class or an instance. Uses *INSPECTLOC* as the region for the inspector window if it is given.

Arguments: *self* A pointer to an instance.  
*INSPECTLOC* The region for the inspector window. If NIL, the system prompts you to place the window.

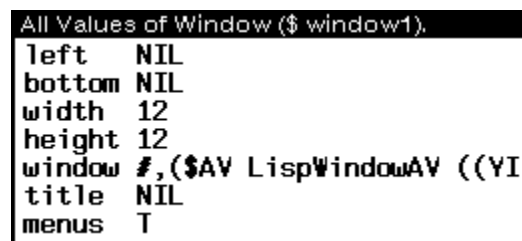
Returns: The Lisp window used by the inspector.

Categories: Object

Example: The following command inspects an instance (\$ window1)

```
88← (← ($ window1) Inspect)
```

This results in the following inspector window:



The image shows a terminal window titled "All Values of Window (\$ window1)". It displays a list of instance variables and their values: left is NIL, bottom is NIL, width is 12, height is 12, window is #, (\$AV LispWindowAV ((YI, title is NIL, and menus is T.

```
All Values of Window ($ window1).
left NIL
bottom NIL
width 12
height 12
window #, ($AV LispWindowAV ((YI
title NIL
menus T
```

---

(← *self* **ListAttribute** *type* *name*)

[Method of Object]

Purpose: Determines instance variable or instance variable property names contained in an instance.

Behavior: Converts *type* into uppercase on entry. The remaining behavior varies according to the arguments.

- If *type* is one of IV, IVPROPS, or NIL, and *name* is the name of an instance variable of *self*, this returns a list of property names of *name* that have property values locally stored in the instance.

- If *type* is IVS, this returns a list of the instance variable names of *self*, whether or not the values for the instance variables are locally stored.
- If *type* is none of the above, this evaluates (`← (Class self) ListAttribute type name`).

Note: Using a *type* of **SUPERS** or **SUPERCLASSES** returns a list of the names of the super classes.

Arguments: *self* A pointer to an instance.  
*type* See Behavior.  
*name* If *type* is one of IV, IVPROPS, or NIL, then *name* is an instance variable of *self*; else it is NIL.

Returns: Value depends on the arguments; see Behavior.

Categories: Object

Specializations: Class

Example: Given that

```
90←(← ($ Window) New 'w1)
#,($& Window (NEW0.1Y%.:;h.eN6 . 515))
```

then

```
91←(← ($ w1) ListAttribute 'IVS)
(left bottom width height title menus)
```

```
92←(← ($ w1) ListAttribute 'IV 'menus)
NIL
```

After opening (`$ w1`), positioning the cursor anywhere on the window, and pressing the left and right mouse buttons to create some menus, then

```
93←(← ($ w1) ListAttribute 'IV 'menus)
(LeftButtonItem RightButtonItem)
```

(`← self ListAttribute! type name verboseFlg`)

[Method of Object]

Purpose: Provides a recursive form of **ListAttribute**. Omits inheritance from the classes **Object** and **Tofu** unless *verboseFlg* is T.

Behavior: Converts *type* into uppercase on entry. The remaining behavior varies according to the arguments.

- If *type* is IVS, this is the same as **ListAttribute**.
- If *type* is one of IV, IVPROPS, or NIL, and *name* is the name of an instance variable of *self*, this returns a list of property names of *name*.
- If *type* is none of the above, this evaluates (`← (Class self) ListAttribute! type name`).

Note: Using a *type* of **SUPERS** or **SUPERCLASSES** returns a list of the names of the super classes.

Arguments: *self* A pointer to an instance.  
*type* See Behavior.  
*name* If *type* is one of IV, PROPS, or NIL, then *name* is an instance variable of *self*; else it is NIL.

*verboseFlag* T or NIL; if T, inheritance from object and **Tofu** are included. If NIL, they are omitted.

Returns: Value depends on the arguments; see Behavior.

Categories: Object

Specializations: Class

Example: Given that

```
95←(← ($ Window) New 'w1)
#,$& Window (NEW0.1Y%:.;h.eN6 . 515))
```

then

```
96←(← ($ w1) ListAttribute! 'IV 'menus)
(RightButtonItem doc TitleItems ...)
```

Purpose: Searches supers hierarchy to find class where *name* is defined.

Behavior: Performs the method **Class.ListAttribute** for *self* and for each super class of *self*, checking to see if *name* (or *propName* as appropriate) is a member of the list returned. The value returned is the class where *name* (or *propName*) is first found.

The *type* argument is changed to uppercase and then coerced to a valued type argument for **ListAttribute**.

- If *type* is one of METHOD, METHODS, NIL, or T, it becomes METHODS. **WhereIs** then looks for a method with the name *name*.
- If *type* is one of IVPROP or IVPROPS, it becomes IVPROPS. **WhereIs** then looks for an instance variable property with the name *name*.
- If *type* is one of IV or IVS, it becomes IVS. **WhereIs** then looks for an instance variable with the name *name*.
- If *type* is one of CV or CVS, it becomes CVS. **WhereIs** then looks for a class variable with the name *name*.

Arguments: *self* A pointer to an instance.  
*type* See Behavior.  
*name* The name of an object attribute being searched for.  
*propName* Property name for instance variable *name*.

Returns: The class where *name* or *propName* is first found.

Categories: Object

Example: The command

```
97←(← (← ($ LatticeBrowser) New) WhereIs 'left 'IV)
returns
#, ($C Window)
```

2.10 OTHER INSTANCE ITEMS

---

2.10 OTHER INSTANCE ITEMS

---

---

## 2.10 Other Instance Items

---

This section describes other items involved with instances.

---

### NoValueFound

---

[Variable]

Purpose/Behavior: Returned as a result of various accesses; initially set to NIL. When developing code, rebind this to the symbol **NoValueFound** to assist in debugging.

---

### (NoValueFound *arg*)

---

[Macro]

Purpose/Behavior: Returns value of (EQ **NoValueFound** *arg*).

Arguments: *arg* Any value.

Returns: T or NIL.

**(ValueFound *arg*)**

[Macro]

Purpose/Behavior: Returns value of (NEQ **NoValueFound** *arg*).

Arguments: *arg* Any value.

Returns: T or NIL.



---

[This page intentionally left blank]

# LIST OF FIGURES

---

|                                                                |       |
|----------------------------------------------------------------|-------|
| 1-1. LOOPS Lattice .....                                       | 1-2   |
| 1-2. An Object .....                                           | 1-2   |
| 1-3. An Object Responding to a Message .....                   | 1-3   |
| 1-4. A Message Containing a Selector .....                     | 1-3   |
| 1-5. Class with Several Objects .....                          | 1-4   |
| 1-6. Class Variables and Instance Variables .....              | 1-5   |
| 1-7. A Metaclass and its Instances .....                       | 1-6   |
| 1-8. A Sample Inheritance Network .....                        | 1-7   |
| 1-9. A Class with a Single Superclass .....                    | 1-8   |
| 1-10. A Class with Multiple Superclasses .....                 | 1-9   |
| 3-1. Simple Inheritance Lattice .....                          | 3-8   |
| 3-2. Multiple Inheritance Lattice .....                        | 3-9   |
| 3-3. Sample Display Editor Window.....                         | 3-11  |
| 4-1. Class Browser Showing Metaclasses.....                    | 4-1   |
| 4-2. Specializations of Tofu.....                              | 4-6   |
| 8-1. The Class <b>ActiveValue</b> and its Specializations..... | 8-3   |
| 10-1. Sample Lattice Browser.....                              | 10-2  |
| 10-2. Sample Supers Browser .....                              | 10-2  |
| 10-3. Sample Metaclass Browser .....                           | 10-3  |
| 10-4. LOOPS Icon.....                                          | 10-4  |
| 10-5. Shading Available for a Node .....                       | 10-47 |
| 18-1. Sample Inspector .....                                   | 18-1  |

[This page intentionally left blank]

Classes provide a description of instances within the object domain. The following information is contained within a class:

- The metaclass for this class. See Chapter 4, Metaclasses, for a discussion of metaclasses.
- Class Properties. Examples of class properties are an edit stamp and documentation.
- The supers list for this class. Classes exist in a hierarchy and the supers list places the class within that hierarchy. Instances of the class contain data and respond to messages that are described within the class and superclasses of the class.
- Class variables, their values, and their properties and values.
- Instance variables, their default values, and their properties and values.

This chapter covers creating and destroying classes, editing, accessing data stored in classes, inheritance, and related topics. Other chapters that contain information relevant to this chapter are Chapter 4, Metaclasses, since a metaclass is a class of classes, and Chapter 10, Browsers, since the primary user interface for manipulating classes is the browser.

---

## 3.1 Creating Classes

---

Several ways are available to create a class:

- Use the browser interface.
- Use function calling or message sending.
- Use dynamic mixins to dynamically create classes.

The rules for naming classes are the same as those for naming instances. Simply stated, a class name must be a litatom. One exception to this rule is the naming of dynamic mixin classes, which is discussed later in this chapter.

A class is generally referred to with this form: (\$ className). See Chapter 2, Instances, for more details regarding LOOPS names.

As discussed in Chapter 2, Instances, the protocol that is followed when instances are created is for the LOOPS system to send the **NewInstance** message to the newly created instance. The **NewInstance** message can be specialized to incorporate behavior specific to the creation time of an instance. Similarly, the system follows a protocol when creating a class using the **New** message. After the class is created, it is sent the **NewClass** message.

3.1.1 Function Calling and Message Sending

The following table shows the items in this section.

| Name               | Type     | Description                                                       |
|--------------------|----------|-------------------------------------------------------------------|
| <b>DefineClass</b> | Function | Creates a new class.                                              |
| <b>New</b>         | Method   | Creates a new class.                                              |
| <b>CreateClass</b> | Method   | Creates a new class.                                              |
| <b>NewClass</b>    | Method   | Provides a placeholder for modifying the class creation protocol. |

**(DefineClass** *name supers self*) [Function]

Purpose: Creates a new class.

Behavior: If *name* is not a litatom, a break occurs.

- If *supers* is non-NIL, it should be a list of classes or names of classes to be the supers for the newly created class. If the list contains multiple classes, this results in a class that has multiple super classes (see Section 3.3, "Inheritance"). The order of classes in the list specifies the order in which lookup will proceed. If one of the these classes is not a valid class, a break occurs.
- If *supers* is NIL and if *self* is (\$ MetaClass), then the supers list is (Class).
- If both *supers* is NIL and *self* is NIL, the supers list is (Object).

If *self* is non-NIL, it is installed as the metaclass for the newly created class. See Chapter 4, Metaclasses.

A class is then built with an **Edited:** property containing the date and time and the value of variable **INITIALS**. (See the *Interlisp-D Reference Manual*.)

The newly created class has no class variables, instance variables, or methods.

The variable **LASTWORD** is set to *name*, which is added to **USERWORDS** for spelling escape completion. (See the *Interlisp-D Reference Manual* for information on **LASTWORD** and **USERWORDS**.)

Arguments: *name* A LOOPS name to be given to the class.

*supers* A list of classes.

*self* A metaclass.

Returns: The class object.

Examples: The following command defines a subclass of the class **Object**.

```
(DefineClass 'ExampleClass)
```

The following command defines a subclass of the class **Window**.

```
(DefineClass 'MyClass '(Window))
```

The following command defines a class with multiple supers: **ExampleClass** and **Window**.

```
(DefineClass 'AnotherClass '(ExampleClass Window))
```

The following command defines a subclass of the class **Window** that has **AbstractClass** as its metaclass.

```
(DefineClass 'DontMakeMe ' (Window) ($ AbstractClass))
```

---

(← *class* **New** *name* *supers* *init1* *init2* *init3*) [Method of Metaclass]

---

Purpose: Creates a new class.

Behavior: Sends the message **CreateClass** to *class*, passing the arguments *name* and *supers*. This returns a new class which is then sent the message **NewClass** passing the arguments *init1*, *init2*, and *init3*.

Arguments: *class* A pointer to a class.  
*name* A LOOPS name to be given to the class.  
*supers* A list of classes.  
*init1, init2, inti3*  
See Behavior.

Returns: The new class.

Categories: Object

Specializes: Class

Specializations: AbstractClass

Example: The following command creates the class, **AClass**, which is a subclass of the class **Window**. The metaclass of **AClass** is **Class**.

```
(← ($ Class) New 'AClass ' (Window))
```

After **AClass** is created, the system sends the following message:

```
(← ($ AClass) NewClass)
```

---

(← *self* **CreateClass** *name* *supers*) [Method of Metaclass]

---

Purpose: Creates a new class.

Behavior: Method version of **DefineClass**.

Arguments: *self* A metaclass.  
*name* The name of the newly created class.  
*supers* A list of classes.

Returns: The class object.

Categories: MetaClass

---

(← *class* **NewClass** *init1* *init2* *init3*) [Method of Class]

---

Purpose: Provides a hook into class initialization. If you want special actions to occur when creating a class, specialize this method.

Arguments: *class* A pointer to a class.  
*init1, init2, init3*  
Dependent on user-defined functionality.

Returns: `class`

Categories: `Class`

Example: Create a subclass of **Class** called **MyClass**:

```
(DefineClass 'MyClass ' (Class))
```

Give it a method **NewClass**:

```
(DefineMethod ($ MyClass) 'NewClass ' (init1 init2 init3)
' (PROGN (PutClass self init1 'prop1) self))
```

This looks like the following display editor window:

```
SEdit MyClass.NewClass Package: INTERLISP
(Method ((MyClass NewClass) self init1 init2 init3)
;; This demonstrates the NewClass protocol
(PutClass self init1 'prop1) self)
```

Now send the class **MyClass** the following command:

```
(← ($ MyClass) New 'testclass NIL "this is a test")
```

This results in the creation of the class shown in the following display editor window:

```
SEdit #,($C testclass) Package: INTERLISP
((MetaClass MyClass Edited%: ; Edited 2-Dec-87
; 15:24 by
; Martin.pasa
prop1 "this is a test")
(Supers Object) (ClassVariables)
(InstanceVariables) (MethodFns))
```

To display the class, enter

```
(← ($ testclass) Edit)
```

---

### 3.1.2 Dynamic Mixins

In some programming situations, you may develop sets of mixins that are designed to be used together. (Mixins are classes that are used only in conjunction with another class to create a subclass, or provide some functionality useful in more than one class.) For example, the class **NamedClass** adds one instance variable **name** and specializes the **New** message to ensure that the instance variable **name** contains the name of the object.

```
(DefineClass 'NamedClass)
(← ($ NamedClass) AddIV 'name)
(DefineMethod ($ NamedClass) 'New ' (self name)
' (←@ (←self NewInstance name) name name))
```

Other classes that want the names of their objects in an instance variable **name** can use **NamedClass** as a mixin.

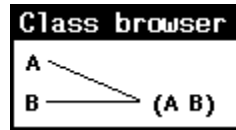
As another example, assume that you have one set consisting of **A1**, **A2**, **A3**, and **A4** and another set containing **B1**, **B2**, and **B3**. Formerly, to allow creation of an instance taking properties from arbitrary combinations of an element from each set, you had to create in advance all 12 combinations of classes with a super from **A** and a super from **B**. This was even more cumbersome if the **As** and **Bs** can also combine with any of a set of 5 **Cs**.

What is desired is the ability to create combinations of these classes on the fly, without having to invent a name for each combination and without having each present in the system when only a few may be needed in any given application. To meet this need, LOOPS now provides the dynamic mixin class. The name of such a class is a list, in order, of the classes which are to be the supers of the class. Such a class is automatically created the first time it is referred to. Thus, the following sequence

```
(DefineClass 'A)
(DefineClass 'B)
(← ($ (A B)) New)
```

creates the class whose supers are **A** and **B** (if it did not already exist), and builds an instance of that class.

Dynamic mixins appear in browsers as shown in this sample window.



All of the browser operations still function on dynamic mixin classes.

These classes print as

```
#, ($C (A B))
```

## 3.2 DESTROYING CLASSES

### 3.2 DESTROYING CLASSES

## 3.2 Destroying Classes

The following messages have been provided to destroy a class that has been created. Destroyed classes, if not being pointed to in some fashion, are eventually collected by the garbage collector.

The following table shows the methods in this section.

| Name                | Type   | Description                                |
|---------------------|--------|--------------------------------------------|
| <b>Destroy</b>      | Method | Removes a class from the LOOPS system.     |
| <b>Destroy!</b>     | Method | Destroys a class and its subclasses.       |
| <b>DestroyClass</b> | Method | Destroys a class by deleting its contents. |

(← class **Destroy**)

[Method of Class]

Purpose: Removes a class from the LOOPS system.

Behavior: If *self* has any subclasses, a break occurs and you are prompted to determine if you want to use **Destroy!**.

Sends the message **DestroyClass** to the metaclass of *self*.

Specializations of this method may be necessary to undo any actions that might have been performed by user specializations of the **NewClass** method. If you specialize **Destroy**, be sure to include a ←**Super** to guarantee that the functionality of the **Destroy** method is performed.



### 3.1 CREATING CLASSES

---

Arguments: *class*      Must be a class.

Returns: NIL

Categories: Object

Specializes: Object

Specializations: DestroyedClass

Example: The following command destroys the class **Datum**:

(← (\$ Datum) Destroy)

---

(← *class* **Destroy!**)

[Method of Class]

Purpose: Destroys a class and its subclasses.

Behavior: Recursively sends the **Destroy** message to *self* and its subclasses.

Arguments: *class*      Must be a class.

Returns: NIL

Categories: Object

Specializes: Object

Specializations: DestroyedClass

---

(← *class* **DestroyClass** *classToDestroy*)

[Method of Class]

Purpose: Destroys *classToDestroy* by deleting its contents. This method is invoked by the LOOPS system and should generally not be called directly by user code. However, it can be specialized to change the way classes are destroyed.

Behavior: Performs the following actions:

- Removes *classToDestroy* from any files on **FILELST**.
- Sends the **Destroy!** message to all methods locally associated with *classToDestroy*.
- Removes *classToDestroy* from any subclass data contained in the supers of *classToDestroy*.
- Changes the class name of *classToDestroy* to **\*aDestroyedClass\***.
- Changes the supers list of *classToDestroy* to **DestroyedObject** and **Object**.
- Changes the metaclass of *classToDestroy* to **DestroyedClass**.
- Sets other fields of the internal class data structure to NIL.

Arguments: *class*      Metaclass of *classToDestroy*.

*classToDestroy*  
Class to destroy.

Returns: NIL

Categories: Class

Specializations: DestroyedClass

---

## 3.3 Inheritance

---

Classes exist in an ordered lattice or hierarchy. Information contained within a class - the supers list - defines where that class is located within the lattice. The supers list specifies the classes immediately above a given class. When an instance of a class is created, it contains not only the instance variables of the defining class, but also the instance variables of all of the classes above the defining class in the class hierarchy. When you try to determine the value of a class variable associated with an instance, all classes above the defining class may be searched. When you send a message to an instance, all classes above the defining class may be searched for the appropriate method.

There are two types of inheritance:

- Simple, in which a class has only one superclass.
- Multiple, in which a class has two or more classes on its supers list.

When an instance is created, it may contain an instance variable that is defined in more than one class. The default value for that instance variable depends on its inheritance. In the case of simple inheritance, the instance variable gets the value from the class that is lowest in the hierarchy. In multiple inheritance, the instance variable gets the value from the class that is lowest in an inheritance list. To create this list,

1. Put the first class that describes the instance.
2. Begin with the first class on its supers list and move up from it, making a list of classes which assume simple inheritance.
3. Build one of these lists for all successive super classes.
4. Append these lists together.
5. Remove all occurrences of any classes that appear in the list a multiplicity of times except for the last entry.

Another way to think about this, which creates the same inheritance list, is the following:

1. Begin with the first super class and walk up the hierarchy until you reach a class where the inheritance paths merge.
2. Walk up each path leading from each successive super class to where paths merge.
3. Take the class where the paths merge and walk up from there.

As an example of simple inheritance, examine Figure 3-1 which shows some of the class variables and instance variables defined within each class.

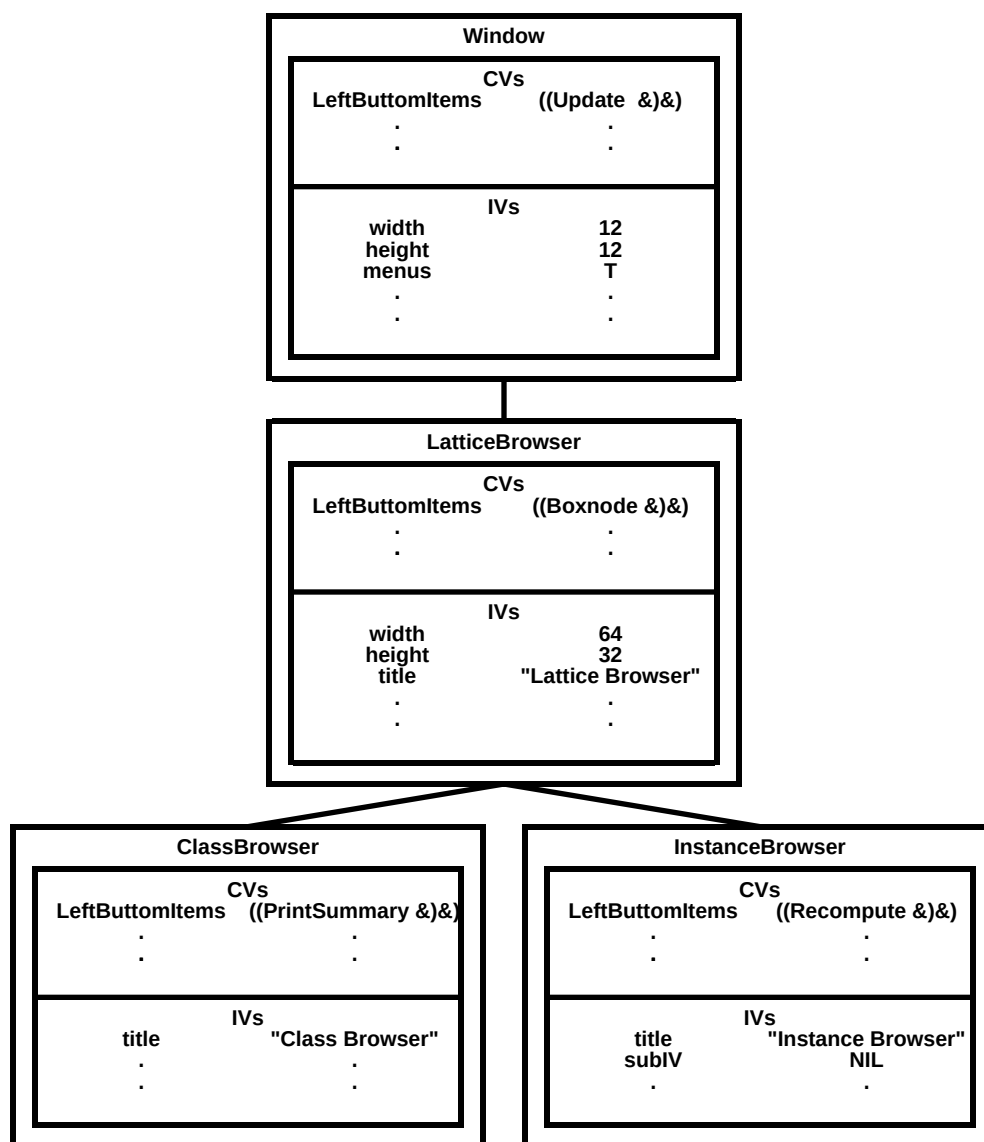


Figure 3-1. Simple Inheritance Lattice

An instance of the class **ClassBrowser** has this as an inheritance list:

ClassBrowser  
 LatticeBrowser  
 Window  
 Object  
 Tofu

The instance variable values of this instance are as follows:

| IV            | Value           | From Class            |
|---------------|-----------------|-----------------------|
| <b>title</b>  | "Class browser" | <b>ClassBrowser</b>   |
| <b>width</b>  | 64              | <b>LatticeBrowser</b> |
| <b>height</b> | 32              | <b>LatticeBrowser</b> |
| <b>menus</b>  | T               | <b>Window</b>         |

Accessing the value of the class variable **LeftButtonItems** causes this value to come from the class **ClassBrowser**.

Figure 3-2 shows an example of multiple inheritance.

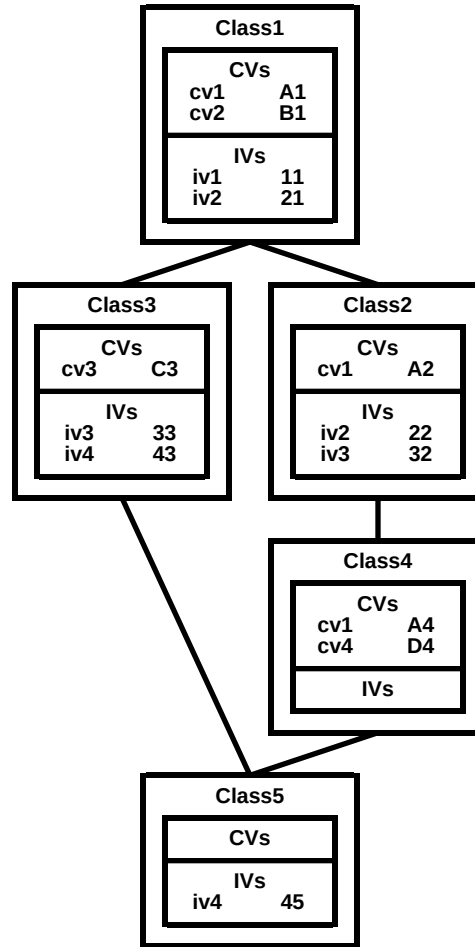


Figure 3-2. Multiple Inheritance Lattice

If the order of the supers for **Class5** is **Class3** and then **Class4** (that is, its supers list is (Class3 Class4 )), then the inheritance list for an instance of **Class5** is as follows:

**Class5**  
**Class3**  
**Class4**  
**Class2**  
**Class1**

The instance variable and class variable values this instance are as follows:

| IV  | Value | From Class    | CV  | Value | From Class    |
|-----|-------|---------------|-----|-------|---------------|
| iv1 | 11    | <b>Class1</b> | cv1 | A4    | <b>Class4</b> |
| iv2 | 22    | <b>Class2</b> | cv2 | B     | <b>Class1</b> |
| iv3 | 33    | <b>Class3</b> | cv3 | C     | <b>Class3</b> |
| iv4 | 45    | <b>Class5</b> | cv4 | D4    | <b>Class4</b> |

3.4 EDITING CLASSES

## 3.4 EDITING CLASSES

---

## 3.4 Editing Classes

---

Changing the contents of a class typically involves using the display editor, although programmatical ways to make these changes are available. To edit a class structure, the LOOPS system first changes the structure to a list and then passes that list to the display editor. Upon exit from the display editor, the system translates the modified list back into the class structure.

The editor is most often called from the browser interface. (See Chapter 10, Browsers.) The following method provides a programmatical way to invoke the editor.

---

(← *class* **Edit** *commands*)

[Method of Class]

Purpose: Edits a class definition.

Behavior: Calls **EDITE** (see the *Interlisp-D Reference Manual*) with the translated class structure passed as the **EXPR** argument and *commands* passed as the **COMS** argument.

This method binds the variable **LASTCLASS** to the class name of *self*.

Arguments: *class*        Pointer to a class.  
*commands*    Commands passed to **EDITE**.

Returns: Name of the class.

Categories: Object

Specializes: Object

Example: The following command causes a display editor window to appear.

```
(← ($ LoopsIcon) Edit)
```

Calling the editor causes a structure to appear in a display editor window. At this time, you can change the structure of the class by using any of the following techniques:

- Changing the value of the class's metaclass. This is done by changing the class name after the word **MetaClass**.
- Changing the superclasses for the class. The form for this is :

```
(Supers class1 class2 ...)
```

At least one class must be in the supers list. The order of this list determines the order of inheritance; the first class after the word **Supers** on this list is the first class to search for inherited data and methods.

- Adding or removing class properties. Class properties occur within the same list as **MetaClass**, after the metaclass class name. The form for this is

```
(MetaClass metaclassName classProp1 propVal1 classProp2
propVal2 ...)
```

- Adding or removing class variables or associated properties. The form for class variables is:

```
(ClassVariables
(cvName1 cvVal1 prop1a propVal1a prop1b propVal1b ...)
(cvName2 cvVal2 prop2a propVal2a prop2b propVal2b ...)
...)
```

It is not necessary to have any properties for a class variable. If the length of each class variable list is not an even number, a break occurs under the editor. The message in the break window describes an odd length list the first time you try to exit from the editor.

- Adding or removing instance variables or associated properties. These have the same form as class variables with the distinction that the value listed for each instance variable is not its value, but only its default value for the purposes of instantiation.

For example, examine the display editor window in Figure 3-3.

```
SEdit #,($C IndirectVariable) Package: INTERLISP
((MetaClass Class doc
 (* Active Value for redirecting references to another
 variable)
 Edited%: (* smL " 9-May-86 09:52"))
 (Supers ActiveValue) (ClassVariables)
 (InstanceVariables
 (object NIL doc (* The object with the "real" variable))
 (varName NIL doc (* The name of the "real" variable))
 (propName NIL doc (* The prop name of the "real" variable))
 (type NIL doc (* The type of the "real" variable)))
 (MethodFns IndirectVariable.GetWrappedValueOnly
 IndirectVariable.PutWrappedValueOnly
 IndirectVariable.WrappingPrecedence))
```

Figure 3-3. Sample Display Editor Window

This figure shows the following information:

- The title bar of the display editor window indicates the class being edited.
- The metaclass of the class **IndirectVariable** in this example is the class **Class**. **IndirectVariable** has two class properties. The first is a **doc** property. The second is an **Edited:** property.
- This class has one super class: **ActiveValue**.
- This class has no class variables. It has four instance variables: **object**, **varName**, **propName**, and **type**. Each has a **doc** property.
- The **MethodFns** are listed in this structure as a convenience. It is not possible to add or delete elements of this list from the editor and have any changes actually occur. Selecting one of the method function names and then selecting Edit (Meta-O in SEdit) allows you to edit that method either as its method code (METHOD-FNS), its method object (METHODS), or its Interlisp code (FNS).

### 3.5 MODIFYING CLASSES

### 3.5 MODIFYING CLASSES

## 3.5 Modifying Classes

In addition to the editing technique for changing a class, you can use programmatic means to modify the structure of a class. This section describes the functions and methods for modifying classes.

| Name          | Type   | Description                       |
|---------------|--------|-----------------------------------|
| <b>Add</b>    | Method | Adds a component to a class.      |
| <b>Delete</b> | Method | Deletes a component from a class. |

|                        |          |                                                                           |
|------------------------|----------|---------------------------------------------------------------------------|
| <b>DeleteClassProp</b> | Function | Removes a class property from a class.                                    |
| <b>AddCV</b>           | Function | Adds a class variable to a class.                                         |
| <b>AddCV</b>           | Method   | Adds a class variable to a class.                                         |
| <b>DeleteCV</b>        | Function | Deletes a class variable or one of its properties from a class.           |
| <b>AddCIV</b>          | Function | Adds an instance variable to a class; can also add properties to a class. |
| <b>AddIV</b>           | Method   | Adds an instance variable to a class.                                     |
| <b>DeleteCIV</b>       | Function | Removes an instance variable or property from a class.                    |
| <b>ReplaceSupers</b>   | Method   | Changes the super classes of a class.                                     |

---

(← *class* **Add** *type name value prop*) [Method of Class]

---

Purpose/Behavior: Adds a component to a class.

Arguments:

|              |                                                                    |
|--------------|--------------------------------------------------------------------|
| <i>class</i> | Pointer to a class.                                                |
| <i>type</i>  | One of IV, IVPROP, CV, CVPROP, METAClass, or METHOD.               |
| <i>name</i>  | The name of the item to be added.                                  |
| <i>value</i> | The value, or default value if <i>type</i> is one of IV or IVPROP. |
| <i>prop</i>  | The name of the property, if a property is to be added.            |

Returns: NIL

Categories: Class

Example: The following command adds a new instance variable **color** to class **Datum**:

```
(← ($ Datum) Add 'IV 'color)
```

---

(← *class* **Delete** *type name prop*) [Method of Class]

---

Purpose: Deletes a component from a class.

Behavior: Varies according to the arguments.

- If *type* is one of IV, IVPROP, or NIL, this calls (**DeleteCIV** *class name prop*).
- If *type* is one of CV or CVPROP, this calls (**DeleteCV** *class name prop*).
- If *type* is META, METAClass, or CLASS, and if *prop* is NIL, then the metaclass of *self* is changed to the class **Class**.
- If *type* is META, METAClass, or CLASS, and if *prop* is non-NIL, then this calls (**DeleteClassProp** *class prop*).
- If *type* is METHOD or SELECTOR, this calls (**DeleteMethod** *class name prop*).

Arguments:

|              |                           |
|--------------|---------------------------|
| <i>class</i> | A pointer to a class.     |
| <i>type</i>  | See Behavior.             |
| <i>name</i>  | IV, CV, or selector name. |

*prop*            A property name.

Returns:    NIL

Categories:    Class

Example:    The following command deletes the instance variable **color** from the class **Datum**:

```
(← ($ Datum) Delete 'IV 'color)
```

---

**(DeleteClassProp classRec propName)** [Function]

Purpose:    Removes a class property from a class.

Behavior:    Marks *classRec* as changed.

Arguments:    *classRec*    Pointer to a class.

*propName*    Property to be deleted.

Returns:    NIL if *propName* is not found; otherwise *propName*.

---

**(AddCV class varName newValue)** [Function]

Purpose:    Adds a class variable to a class.

Behavior:    Varies according to the arguments.

- If *varName* is NIL, you are prompted to enter a name.
- If *varName* is already a class variable, its value is changed to *newValue*. NIL is returned.
- If *varName* is not a class variable of *class*, it is added to *class* with the value *newValue*. Also, a **doc** property is added with the following value:

```
`(* CV added by , (USERNAME NIL T))
```

*varName* is returned in this case.

Arguments:    *class*            A pointer to a class.

*varName*        Name of the new variable.

*newValue*        The new value.

Returns:    Value depends on the arguments; see Behavior.

---

**(← class AddCV varName newValue)** [Method of Class]

Purpose:    Adds a class variable to a class.

Behavior:    Provides a method version of the function **AddCV**.

Arguments:    See the function **AddCV**.

Returns:    NIL

Categories:    Class

---

**(DeleteCV class varName prop)** [Function]



Purpose: Deletes a class variable or one of its properties from a class.

Behavior: Marks *class* as changed.

Arguments: *class* Pointer to a class.  
*varName* Class variable name to be deleted.  
*prop* Property to be deleted.

Returns: NIL, if *varName* is not found, else *varName*.

---

(**AddCIV** *class varName defaultValue otherProps*) [Function]

---

Purpose: Adds an instance variable, and perhaps properties, to a class.

Behavior: If the length of *otherProps* is odd, an error occurs.

The remaining behavior varies according to the arguments.

- If *varName* is NIL, you are prompted to enter a name.
- If *varName* is already an instance variable of *class*, then change its default value to *defaultValue*. Properties on *otherProps* are added or changed as necessary. NIL is returned.
- If *varName* is not an instance variable of *class*, it is added to *class* and its default value is *defaultValue*. Properties on *otherProps* are also added. If there is no **doc** property, it is added and given the following value:  

```
`(* IV added by , (USERNAME NIL T))
```

*varName* is returned in this case.

Arguments: *class* Must be a pointer to a class.  
*varName* New instance variable name.  
*defaultValue* New default value.  
*otherProps* NIL or a list in property list format.

Returns: Value depends on the arguments; see Behavior.

---

(**← class AddIV** *varName defaultValue otherProps*) [Method of Class]

---

Purpose: Adds an instance variable to a class.

Behavior: Provides a method version of the function **AddCIV**.

Arguments: See the function **AddCIV**.

Returns: NIL

Categories: Object

Specializes: Object

Example: Define a new class **TestClass**, add an instance variable **testIV** with two properties **testProp1** and **testProp2**, all with initial values, and then prettyprint the class's variables.

```
64← (DefineClass 'TestClass)
#, ($C TestClass)
```

```

65←(← ($ TestClass) AddIV 'testIV 1234
'(testProp1 1 testProp2 2))
testIV

66←(← ($ TestClass) PPV! T)
#,$ TestClass)
MetaClass and its Properties
Class Edited: (* edited: 24-Sep-87 08:41 by mcgill)
Supers
(Object Tofu)
Instance Variable Descriptions
testIV 1234 doc (* IV added by MCGILL)
testProp2 2 testProp1 1
Class Variables

```

**(DeleteCIV class varName prop)**

[Function]

- Purpose:** Removes an instance variable or property from a class.
- Behavior:** If *class* does not have *varName*, a break occurs.  
Marks *class* as changed.
- Arguments:** *class*            Pointer to a class.  
*varName*       Instance variable to be deleted.  
*prop*           If non-NIL, property to be deleted.
- Returns:** Value depends on the arguments.
- NIL for removing an instance variable if successful.
  - *prop* for removing a property if successful.
  - NIL if *prop* is not a property.

**(← class ReplaceSupers supers)**

[Method of Class]

- Purpose:** Changes the super classes of a class.
- Behavior:** Checks that no circular lists can be made in the inheritance lattice.
- If the super class of *class* is Tofu, no change occurs.
  - If *supers* is different from the current *supers*, the *supers* list of *class* is changed and *class* is marked as changed.
- Arguments:** *class*            Pointer to a class.  
*supers*           A list of class names or classes.
- Returns:** NIL
- Categories:** Class

## 3.6 METHODS FOR MANIPULATING CLASS NAMES

## 3.6 METHODS FOR MANIPULATING CLASS NAMES

## 3.6 Methods for Manipulating Class Names

LOOPS classes must have one and only one LOOPS name. The following functions and methods allow you to change and rename class names.

| Name             | Type     | Description                                                                                |
|------------------|----------|--------------------------------------------------------------------------------------------|
| <b>Rename</b>    | Method   | Changes the name of a class. Prompts for name if not provided, then calls <b>SetName</b> . |
| <b>SetName</b>   | Method   | Changes the name of a class.                                                               |
| <b>UnSetName</b> | Method   | Unnames a class.                                                                           |
| <b>ClassName</b> | Function | Finds the class name of an object.                                                         |

(← *class* **Rename** *newName*) [Method of Class]

Purpose: Changes the name of a class. Prompts for name if not provided, then calls **SetName**.

Behavior: Varies according to the argument.

- If *newName* is NIL, this causes a break and prompts you for a name. **Rename** then sends the message **SetName** passing this name as an argument
- If *newName* is non-NIL, **Rename** sends the message **SetName** passing *newName* as an argument.

Arguments: *class*      Pointer to a class.  
*newName*      A litatom.

Returns: NIL

Categories: Object

Specializes: Object

Example: The following command renames class **Datum** to **Thing**:

```
(← ($ Datum) Rename 'Thing)
```

(← *class* **SetName** *newClassName*) [Method of Class]

Purpose: Changes the name of a class.

Behavior: Removes the old name of *self* from **ObjNameTable**.

**SetName** uses the Interlisp-D function **EDITCALLERS** to rename references to the class name or any file that contains the class. If **EDITCALLERS** cannot succeed, for example, when a file is not RANDACCESSP, a message is printed that the class cannot be renamed on that file. For complete information on **EDITCALLERS**, see the *Interlisp-D Reference Manual*.

The names of the method functions of *class* are changed to use *newClassName*.

Arguments: *class*      Pointer to a class.  
*newClassName*  
                     A litatom.

Returns: NIL  
 Categories: Object  
 Specializes: Object

(← *class* **UnSetName**)

[Method of Class]

Purpose: Unnames a class, but does not destroy it. Has limited usefulness for keeping a class name from being typed in.

Behavior: Removes *class* from the LOOPS name hash table and from any files on **FILELST**. This method is intended to be used internally only; it is not recommended to create an unnamed class.

Arguments: *class*      Pointer to a class.

Returns: NIL

Categories: Object

Specializes: Object

(**ClassName** *self*)

[Function]

Purpose: Finds the class name of an object.

Behavior: Varies according to the arguments.

- If *self* is a class, this returns the name of that class.
- If *self* is an instance, this returns the name of the class that describes that instance.
- If *self* is neither a class or an instance, this returns **Tofu**.

Arguments: *self*      See Behavior.

Returns: Value depends on the arguments; see Behavior.

Example: Given that

```
(← ($ Window) New 'w1)
```

the commands

```
(ClassName ($ w1))
(ClassName ($ Window))
```

both return **Window**.

### 3.7 QUERYING THE STRUCTURE OF A CLASS

#### 3.7 QUERYING THE STRUCTURE OF A CLASS

## 3.7 Querying the Structure of a Class

The following functions and methods allow you to query what is contained in a class.

| Name | Type | Description |
|------|------|-------------|
|------|------|-------------|

|                       |          |                                                                                     |
|-----------------------|----------|-------------------------------------------------------------------------------------|
| <b>GetClassProp</b>   | Method   | Obtains a class's metaclass or properties.                                          |
| <b>HasAttribute</b>   | Method   | Determines whether <i>self</i> has an attribute name.                               |
| <b>HasAttribute!</b>  | Method   | Recursive form of <b>HasAttribute</b> , but works only on classes.                  |
| <b>HasCV</b>          | Method   | Determines if a class has a class variable with a specified property.               |
| <b>HasItem</b>        | Method   | Determines if a class has an item of a given type.                                  |
| <b>HasIV</b>          | Method   | Determines if a class has an instance variable with a specified property.           |
| <b>HasIV!</b>         | Method   | Same as <b>HasIV</b> , except that <b>HasIV!</b> also searches up the supers chain. |
| <b>ListAttribute</b>  | Method   | Lists the elements of a class that are local to the class.                          |
| <b>ListAttribute!</b> | Method   | Lists all the items associated with a class.                                        |
| <b>WhoHas</b>         | Function | Determines what classes contains a specified item.                                  |

---

(← *class* **GetClassProp** *propname*) [Method of Class]

---

Purpose: Obtains a class's metaclass or properties by following metaclass links.

Behavior: Varies according to the arguments.

- If *propname* is NIL, this returns the *class*'s metaclass.
- If *propname* is non-NIL, this looks first in *class* for that property. If it cannot find it there, it looks through *class*'s metaclass links.
- If no property is found, the value of the variable **NotSetValue** is returned.

Arguments: *class* A pointer to a class.  
*prop* Property name.

Returns: Value depends on the arguments; see Behavior.

Categories: Class

Example: The following commands show the variety of responses.

```
51←(← ($ Window) GetClassProp)
#, ($C Class)

52←(← ($ Window) GetClassProp 'doc)
"A LOOPS object which represents a window"

53←(← ($ IconWindow) GetClassProp 'doc)
"An icon window that appears as an irregular shaped image
on the screen -- See the ICONW Library utility"
```

---

(← *self* **HasAttribute** *type name propname*) [Method of Class]

---

Purpose: Determines whether *self* has an attribute name, with a property *propname* if supplied.

Behavior: *self* can be an instance or a class. Remaining behavior depends on *type*, which is converted to uppercase on entry:

- If *type* is IV, IVPROP, or NIL, this returns T if *self* has an instance variable of *name*, with a property called *propname* (if *propname* is non-NIL), otherwise it returns NIL.
- If *type* is CV or CVPROP, this returns T if *self* has a CV called *name*, with a property of *propname* (if *propname* is non-NIL), otherwise it returns NIL.
- If *type* is METHOD or SELECTOR, this returns NIL or the name of the method implementing *name*.

**HasAttribute** applied to an instance reports on the actual state of the instance; it sees all instance variables and class variables whether local, inherited, or specially added to the instance. If only local attributes are required, use ( $\leftarrow$  (Class instance) HasAttribute ...).

Arguments: *self* Can be an instance or a class.  
*type* See Behavior.  
*name* A symbol which is looked up as the variable or method name.  
*propname* A symbol which is looked up as the property name.

Returns: See Behavior.

Categories: Object

Specializations: Class

Example: The command

```
($\leftarrow$ ($ LoopsIcon) HasAttribute 'IV 'icon)
```

returns T.

( $\leftarrow$  class **HasAttribute!** *type name propname*)

[Method of Class]

Purpose: Recursive form of **HasAttribute**; only works on classes

Behavior: Similar to **HasAttribute**, but will also search through *class* 's supers.

Arguments: *class* A class.  
*type* See Behavior under **HasAttribute**.  
*name* A symbol which is looked up as the variable or method name.  
*propname* A symbol which is looked up as the property name.

Returns: See Behavior.

Categories: Object

Specializations: Class

Example: The command

```
($\leftarrow$ ($ LoopsIcon) HasAttribute 'IV 'left)
```

returns NIL, but

```
($\leftarrow$ ($ LoopsIcon) HasAttribute! 'IV 'left)
```

returns T.

---

(← *class* **HasCV** *cvName prop*)

[Method of Class]

Purpose: Determines if a class has a class variable *cvName* with a property *prop*.

Note: The preferred form of this method is **HasAttribute** or **HasAttribute!**.

Behavior: Varies according to the arguments.

- If *prop* is NIL, this returns T if *class* contains a class variable called *cvName*, else NIL.
- If *prop* is non-NIL, this returns T if *class* contains a class variable called *cvName* with the property *prop*, else NIL.

Note: **HasCV** does not distinguish between locally defined class variables and inherited class variables. If you need to test a class to see if it has a class variable defined locally, you can use the **HasAttribute** method. For example, the form (←MyClass HasAttribute 'CV 'ABC) will return a non-NIL value if and only if the class **MyClass** has a local definition of the class variable ABC.

Arguments: *class*            A pointer to a class.  
*cvName*            A class variable name.  
*prop*            Property name.

Returns: NIL or T; see Behavior.

Categories: Object

Specializes: Object

Example: The command

```
(← ($ Window) HasCV 'TitleItems)
```

returns T.

---

(← *class* **HasItem** *itemName prop itemType*)

[Method of Class]

Purpose: Determines if a class has an item of a given type.

Note: The preferred form of this method is **HasAttribute** or **HasAttribute!**.

Behavior: Varies according to the arguments.

- If *itemType* is IV or IVS, this sends the message (← *class* **HasIV** *itemName prop*).
- If *itemType* is CV or CVS, this sends the message (← *class* **HasCV** *itemName prop*).
- If *itemType* is SELECTOR, METHOD, SELECTORS, or METHODS, this finds the corresponding local method of *class*.
- If *itemType* is not one of the above, this returns NIL.

Arguments: *class*            Pointer to a class.  
*prop*            Property name.  
*itemType*       See Behavior.

Returns: Value depends on the arguments; see Behavior.

Categories: Class

(← class **HasIV** *IVName prop*) [Method of Class]

Purpose: Determines if a class has an instance variable *IVName* with a property *prop*.

Note: The preferred form of this method is **HasAttribute** or **HasAttribute!**.

Behavior: *class* should point to a class.

- If *prop* is NIL, this returns T if *IVName* is contained in *class*.
- If *prop* is non-NIL, this returns T if *IVName* is contained in *class*, and *prop* is a property of *IVName* in *class* or one of its supers.

Arguments: *class* Pointer to a class.

*IVName* Instance variable name.

*prop* Property name.

Returns: Value depends on the arguments; see Behavior.

Categories: Object

Specializes: Object

(← class **HasIV!** *IVName prop*) [Method of Class]

Purpose/Behavior: Same as **HasIV**, except that **HasIV!** also searches up the supers chain.

Note: The preferred form of this method is **HasAttribute** or **HasAttribute!**.

Arguments: See the method **HasIV**.

Returns: Value depends on the arguments; see Behavior.

Categories: Class

(← class **ListAttribute** *type name*) [Method of Class]

Purpose: Lists the elements of a class that are local to the class.

Behavior: *type* is converted to uppercase on entry. The remaining behavior varies according to the arguments.

- If *type* is IVS, this returns the instance variable names (not values) local to *class*. *name* is ignored.
- If *type* is IV, IVPROPS, or NIL, *name* should be bound to an instance variable of *class*. This returns the property names (not values) of the instance variable *name*. If *name* is not an instance variable of *class*, this returns NIL.
- If *type* is CVS, this returns the class variables local to *class*. *name* is ignored.
- If *type* is CV or CVPROPS, *name* should be bound to a class variable of *class*. This returns the property names of the class variable *name*. If *name* is not a class variable of *class*, this returns NIL.



- If *type* is METHODS or SELECTORS, this returns the selectors for the class. *name* is ignored.

Arguments:    *class*            Pointer to a class.  
              *type*            See Behavior.  
              *name*            See Behavior.

Returns:      Value depends on the arguments; see Behavior.

Categories:   Object

Specializes:   Object

Example:      The following commands show the variety of responses.

```
55←(← ($ SupersBrowser) ListAttribute 'IVs')
(title)
```

```
56←(← ($ Window) ListAttribute 'iv 'menus)
(DontSave Title LeftButtonItem MiddleButtonItem TitleItems doc)
```

```
57←(← ($ IconWindow) ListAttribute 'METHODS')
(GetMenuItems)
```

---

(← *class* **ListAttribute!** *type name verboseFlg*)

[Method of Class]

Purpose:       Lists all items associated with a class.

Behavior:     Provides a recursive version of **ListAttribute**.

If *verboseFlg* is NIL, items that are inherited from Tofu, Object, or Class are omitted, unless *class* is one of Tofu, Object, or Class.

*type* is converted to uppercase on entry.

- If *type* is META or METAClass, this returns the same as **ListAttribute**.
- If *type* is IVS or NIL, this returns the instance variables an instance of *class* would have.
- If *type* is SUPERS or SUPERCLASSES, this returns the ordered list of super classes of *class*.
- If *type* is SUBS or SUBCLASSES, this returns all of the subclasses of *class*.
- If *type* is any other option that can be passed to **ListAttribute**, this returns all local and inherited values.

Arguments:    *class*            Pointer to a class.  
              *type*            See Behavior.  
              *name*            A litatom.  
              *verboseFlg*    See Behavior.

Returns:      Value depends on the arguments; see Behavior.

Categories:   Object

Specializes:   Object

---

(WhoHas *name type files editFlg*)

[Function]

|            |                                                                                                                                                                                       |                                                                                    |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| Purpose:   | Determines what classes contain a specified item.                                                                                                                                     |                                                                                    |
| Behavior:  | Returns a list of classes on <i>files</i> that contain <i>name</i> . If <i>editFlg</i> is non-NIL, then edit the methods (if <i>type</i> is METHOD), or the classes before returning. |                                                                                    |
| Arguments: | <i>name</i>                                                                                                                                                                           | The item specified.                                                                |
|            | <i>type</i>                                                                                                                                                                           | One of IV, CV, METHOD, or Method. If <i>type</i> is NIL, it defaults to METHOD.    |
|            | <i>files</i>                                                                                                                                                                          | A file or a list of files. If <i>files</i> is NIL, it defaults to <b>FILELST</b> . |
|            | <i>editFlg</i>                                                                                                                                                                        | T or NIL.                                                                          |
| Returns:   | A list of classes on <i>files</i> that contain <i>name</i> .                                                                                                                          |                                                                                    |

## 3.8 COPYING CLASSES AND THEIR CONTENTS

## 3.8 COPYING CLASSES AND THEIR CONTENTS

## 3.8 Copying Classes and Their Contents

Inheritance lets classes be described in terms of other classes in a hierarchical manner. When it is preferable to duplicate a class description in different parts of a lattice these methods provide the capability.

The following table shows the methods in this section.

| Name          | Type   | Description                                   |
|---------------|--------|-----------------------------------------------|
| <b>Copy</b>   | Method | Copies a class.                               |
| <b>CopyCV</b> | Method | Copies a class variable to another class.     |
| <b>CopyIV</b> | Method | Copies an instance variable to another class. |

(← class **Copy** *name*)

[Method of Class]

|             |                                                                                                                               |
|-------------|-------------------------------------------------------------------------------------------------------------------------------|
| Purpose:    | Makes a copy of a class.                                                                                                      |
| Behavior:   | If <i>name</i> is NIL, you are prompted to supply a name for the new class. This copies variables and properties and methods. |
| Arguments:  | <i>class</i> The class being copied.                                                                                          |
|             | <i>name</i> The name of the copy.                                                                                             |
| Returns:    | The new class.                                                                                                                |
| Categories: | Class                                                                                                                         |
| Example:    | Given that                                                                                                                    |

```
(DefineClass 'Datum)
(← ($ Datum) AddIV 'something)
```

the following command makes a copy of class **Datum** and names it **Thing**:

```
(← ($ Datum) Copy 'Thing)
```

(← class **CopyCV** *cvName toClass*)

[Method of Class]

|                   |                                                                                                                |                                         |
|-------------------|----------------------------------------------------------------------------------------------------------------|-----------------------------------------|
| Purpose/Behavior: | Copies a class variable to another class. This also copies the properties of <i>cvName</i> to <i>toClass</i> . |                                         |
| Arguments:        | <i>class</i>                                                                                                   | The source class.                       |
|                   | <i>cvName</i>                                                                                                  | The name of the class variable to copy. |
|                   | <i>toClass</i>                                                                                                 | The destination class.                  |
| Returns:          | NIL                                                                                                            |                                         |
| Categories:       | Class                                                                                                          |                                         |

---

(← *class* **CopyIV** *ivName toClass*)

[Method of Class]

|                   |                                                                                                                    |                                            |
|-------------------|--------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| Purpose/Behavior: | Copies an instance variable to another class. This also copies the properties of <i>ivName</i> to <i>toClass</i> . |                                            |
| Arguments:        | <i>class</i>                                                                                                       | The source class.                          |
|                   | <i>ivName</i>                                                                                                      | The name of the instance variable to copy. |
|                   | <i>toClass</i>                                                                                                     | The destination class.                     |
| Returns:          | NIL                                                                                                                |                                            |
| Categories:       | Class                                                                                                              |                                            |

3.9 ENUMERATING INSTANCES OF CLASSES

---

3.9 ENUMERATING INSTANCES OF CLASSES

---

---

## 3.9 Enumerating Instances of Classes

---

New instances may be created without names, or without being tracked. These methods allow you to produce a list of instances according to their classes. **Prototype** instances are a convenience used where the methods defined for a class must be used, but there is no logical instance for the class.

The following table shows the items in this section.

| Name                 | Type   | Description                                                                                    |
|----------------------|--------|------------------------------------------------------------------------------------------------|
| <b>AllInstances</b>  | Method | Finds all instances of a class.                                                                |
| <b>AllInstances!</b> | Method | Finds all instances of a class or its subclasses.                                              |
| <b>IndexedObject</b> | Class  | Keeps track of instances so that <b>AllInstances</b> searches can proceed more rapidly.        |
| <b>PrintOn</b>       | Method | Modifies how instances of <b>IndexedObject</b> that do not have LOOPS names will be printed.   |
| <b>Prototype</b>     | Method | Returns an instance of a class that is stored on the class's class variable <b>Prototype</b> . |

---

(← *class* **AllInstances**)

[Method of Class]

|          |                                 |
|----------|---------------------------------|
| Purpose: | Finds all instances of a class. |
|----------|---------------------------------|

|             |                                                                                                                                                                                                                                                                          |          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| Behavior:   | Checks if <i>class</i> is a subclass of <b>IndexedObject</b> . If so, a faster search is used to find all of the instances of <i>class</i> . If not, this checks if each object is an instance of <i>class</i> . Instances that do not yet have a UID will not be found. |          |
| Arguments:  | <i>class</i>                                                                                                                                                                                                                                                             | A class. |
| Returns:    | A list of the instances found.                                                                                                                                                                                                                                           |          |
| Categories: | Class                                                                                                                                                                                                                                                                    |          |
| Example:    | The following command produces a list of all the LOOPS window instances:<br><br>61← (← (\$ Window) AllInstances)                                                                                                                                                         |          |

(← *class* **AllInstances!**)

[Method of Class]

|             |                                                                                                                                                                                                                                                                                         |                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------|
| Purpose:    | Finds all instances of a class or its subclasses.                                                                                                                                                                                                                                       |                       |
| Behavior:   | Returns a list of instances that are instances of <i>class</i> or any of its subclasses. Instances that do not have the class <b>IndexedObject</b> as a super class, or that do not yet have a UID are not found. (See Chapter 18, Reading and Printing, for more information on UIDs.) |                       |
| Arguments:  | <i>class</i>                                                                                                                                                                                                                                                                            | A pointer to a class. |
| Returns:    | A list of the instances found.                                                                                                                                                                                                                                                          |                       |
| Categories: | Class                                                                                                                                                                                                                                                                                   |                       |

**IndexedObject**

[Class]

|                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                                                                                                                                                                                |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:         | Keeps track of instances so that <b>AllInstances</b> searches can proceed more rapidly.                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                |
| Behavior:        | This class is to be used as a <b>Mixin</b> (an addition superclass), and should be the first class on a supers list for a class.<br><br><b>IndexedObject</b> provides <b>NewInstance</b> and <b>Destroy</b> protocols that cause instances to be added to or removed from a global list when they are created or destroyed. This global list allows the <b>AllInstances</b> protocols to search more quickly.<br><br><b>IndexedObject</b> also provides a <b>PrintOn</b> protocol that modifies how instances will be printed if they have no LOOPS name. |                                                                                                                                                                                                                                                                                |
| MetaClass:       | Class                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                |
| Supers:          | Object                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                |
| Class Variables: | <b>IdentifierVar</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | The name of an instance variable which will contain a string which could provide some identification to the user. Used in <b>PrintOn</b> if variable is in object and filled. <b>shortName</b> , the value of this class variable, is the default variable name which is used. |

(← *self* **PrintOn**)

[Method of IndexedObject]

|           |                                                                                                                                                                                             |  |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| Purpose:  | Modifies how instances of <b>IndexedObject</b> that do not have LOOPS names will be printed.                                                                                                |  |
| Behavior: | If <i>self</i> has a LOOPS name, or if <i>self</i> does not have an instance variable with a name equal to (@ <i>self</i> ::IdentifierVar), then do a (← <b>Super</b> ). Otherwise, build a |  |

form that incorporates the value of the instance variable *IV* referenced by (*@self ::IdentifierVar*).

Arguments: *self*            An instance.

Returns:    A list ; see example

Categories:    Object

Specializes:    Object

Example:    Create a class, **IndexedObjectTest**, that has this structure.

```
62← (DefineClass 'IndexedObjectTest ' (IndexedObject))
#, ($C IndexedObjectTest)

63← (← ($ IndexedObjectTest) AddIV 'shortName 'ioTest)
shortName
```

Create an instance.

```
64← (SETQ test (← ($ IndexedObjectTest) New))
#, ($& IndexedObjectTest (YMW0.0X%:>T4.n18 . 36))

65← (←@ test shortName 'changeName)
changeName

66← (← test PrintOn)
("#, " $& IndexedObjectTest (changeName (YMW0.0X%:>T4.n18 . 36)))
```

---

(← *class* **Prototype** *newProtoFlg*)

[Method of Class]

Purpose:    Returns a prototype instance of a class.

Behavior:    Varies according to the arguments.

- If *class* has a class variable **Prototype** and the variable's value is an instance of *class*, return the value (assuming *newProtoFlg* is NIL).
- If there is no class variable **Prototype**, or if there is a class variable **Prototype** but its value is not an instance of *class*, or if *newProtoFlg* is non-NIL, then create a new instance of *class*, store the instance on the class variable **Prototype**, and return the instance.

See **Proto** in Chapter 7, Message Sending Forms, for more information.

Arguments: *class*            A class.

*newProtoFlg*  
If non-NIL, create a new prototype instance.

Returns:    The prototype.

Categories:    Class

Example:    LOOPS defines an icon to make it easy to bring up class browsers and file browsers. The icon is the **Prototype** instance of the class **LoopsIcon**.

To move the icon to the center of the bottom of the screen, enter

```
71← (←Proto ($ LoopsIcon) Move (QUOTIENT SCREENWIDTH 2) 0)
(576 . 0)
```

This places the left edge of the icon at the center of the screen. To move the icon to the center of the screen, enter

```

72←(LET ((icon (← ($ LoopsIcon) Prototype)))
 (← icon Move (QUOTIENT (DIFFERENCE SCREENWIDTH
 (@ icon width))
 2)
 0))
(544 . 0)

```

## 3.10 DEALING WITH INHERITANCE

## 3.10 DEALING WITH INHERITANCE

## 3.10 Dealing with Inheritance

The inheritance lattice for classes shows how methods and variables are shared (see Chapter 10, Browsers, for details on how to graph the lattice on the screen). To programmatically inspect and add to this lattice via **Specialize**, use the following functions and methods:

| Name                 | Type     | Description                                           |
|----------------------|----------|-------------------------------------------------------|
| <b>Fringe</b>        | Method   | Finds the leaves of a branch of an inheritance tree.  |
| <b>Specialize</b>    | Method   | Creates a subclass of a class.                        |
| <b>SubClasses</b>    | Method   | Returns a list of subclasses.                         |
| <b>Subclass</b>      | Method   | Determines if a class is a subclass of another class. |
| <b>AllSubClasses</b> | Function | Computes the subclasses of a class.                   |
| <b>SubsTree</b>      | Function | Computes all the names of the subclasses of a class.  |

(← *class* **Fringe**) [Method of Class]

**Purpose:** Finds the leaves of a branch of an inheritance tree.

**Behavior:** Returns a list of subclasses of *class*, whether close or distant, that have no subclasses.

**Arguments:** *class*      A class, the root of the tree to explore.

**Returns:** Names of subclasses of *class* that have no subclasses.

**Categories:** Class

**Example:** The following commands show the variety of responses.

```

73←(← ($ Window) Fringe)
(InstanceBrowser MetaBrowser SupersBrowser FileBrowser
LoopsIcon IconWindow)

```

```

74←(← ($ ClassBrowser) Fringe)
(MetaBrowser SupersBrowser FileBrowser)

```

(← *class* **Specialize** *newName*) [Method of Class]

**Purpose:** Creates a subclass of a class.

**Behavior:** Creates a class with *class* as its only super.

- If *newName* is non-NIL, this is the name of the new class.

- If *newName* is NIL, this creates a name consisting of the name of *class* followed by an integer.

Arguments: *class*            Pointer to a class.  
*newName*    Name of the new subclass.

Returns:    The new class.

Categories: Class

Example:    Given that

```
(DefineClass 'Datum)
```

the following command creates a specialization of the class **Datum** called **DatumX**:

```
(← ($ Datum) Specialize 'DatumX)
```

---

(← *class* **SubClasses**)

[Method of Class]

Purpose:     Returns a list of subclasses.

Behavior:   The classes returned by this are the immediate subclasses of *class*.

Arguments: *class*            A pointer to a class.

Returns:    A list of subclasses.

Categories: Class

Specializations: DestroyedClass

Example:    The following command gets a list of the subclasses of the class **Window**:

```
(← ($ Window) SubClasses)
```

---

(← *class* **Subclass** *super*)

[Method of Class]

Purpose:     Determines if a class is a subclass of another class.

Behavior:   If *class* is a subclass of *super*, *super* is returned, else NIL.

Arguments: *class*            Pointer to a class.

*super*        Either the LOOPS name of a class or a pointer to a class.

Returns:    Value depends on the arguments; see Behavior.

Categories: Class

Example:    The command

```
(← ($ DestroyedClass) Subclass 'Class)
```

returns

```
#, ($C Class)
```

---

(**AllSubClasses** *class* *currentSubs*)

[Function]

Purpose:     Computes the subclasses of a class.

Behavior: This is a recursive function that computes (without duplicates) all of the subclasses of *class*.

Arguments: *class* Must be a pointer to a class, for example, (\$ Window).  
*currentSubs* Used by LOOPS; NIL when called by the user.

Returns: A list of classes.

Example: The command  
 (AllSubClasses (\$ LatticeBrowser))  
 returns  
 (#, (\$C FileBrowser) #, (\$C SupersBrowser)  
 #, (\$C MetaBrowser) #, (\$C ClassBrowser)  
 #, (\$C InstanceBrowser))

**(SubsTree class currentList)**

[Function]

Purpose: Computes the names of the subclasses of a class.

Behavior: Provides a recursive function that computes (without duplicates) all of the names of the subclasses of *class*.

Arguments: *class* Can be a class name or a pointer to a class  
*currentList* Used internally by **SubsTree**; it should be NIL when called by the user.

Returns: A list of class names.

Example: The command  
 (SubsTree 'LatticeBrowser)  
 returns  
 (InstanceBrowser ClassBrowser MetaBrowser SupersBrowser  
 FileBrowser)



[This page intentionally left blank]

---

## Overview of the Manual

---

The *LOOPS Reference Manual* provides a detailed description of all the methods, functions, classes, and other items available in the Lisp Object-Oriented Programming System, LOOPS. This manual describes the Medley Release of LOOPS, which runs under Medley.

This manual is for people who are familiar with LOOPS programming principles, and is not intended to teach you LOOPS or how to use it. Please contact your LOOPS distributor for information about classes and training material.

---

## Organization of the Manual and How to Use It

---

This manual is divided into chapters, with most chapters focusing on a particular aspect of LOOPS. The organization of this manual is similar to the *Interlisp-D Reference Manual*.

A Table of Contents is included at the beginning of the manual to help you find specific material. At the end of the manual, a Glossary is included to define terms within the context of LOOPS.

All readers should review Chapter 1, Introduction, before referring to specific material.

---

## Conventions

---

This manual uses the following conventions:

- Case is significant in LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown. Typically, this means that method names are capitalized and variables are not.
- Arguments appear in italic type. Optional arguments are indicated by a dash (-).
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a message sending form appears as follows:

(← self **Selector** *Arg1 Arg2* -)

- Examples appear in the following typeface:

89← (←LOGIN)

- All examples are typed into an Interlisp Exec. This is the recommended Exec for all LOOPS expressions.

- Methods with an exclamation mark (!) suffix usually perform operations deeply into class structure instead of only on a given object.
- Methods with a question mark (?) suffix usually are predicates; that is, truth functions.
- Method names often appear in the form **ClassName.SelectorName**.
- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

This manual describes the LOOPS items (functions, methods, etc.) by using the following template:

|                  |                                                                                                                                                                                                                                                                                                                                                                     |                 |             |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|-------------|
| Purpose:         | Gives a short statement of what the item does.                                                                                                                                                                                                                                                                                                                      |                 |             |
| Behavior:        | Provides the details of how the item operates.                                                                                                                                                                                                                                                                                                                      |                 |             |
| Arguments:       | Describes each argument in the following format:<br><br><table> <tr> <td><i>argument</i></td><td>Description</td></tr> </table>                                                                                                                                                                                                                                     | <i>argument</i> | Description |
| <i>argument</i>  | Description                                                                                                                                                                                                                                                                                                                                                         |                 |             |
| Returns:         | States what the item returns, and does not appear if the item does not return a value. The phrase "Used as a side effect only." means that the purpose of the item is to perform a computation or action that is independent of any returned value, not to return a particular value.                                                                               |                 |             |
| Categories:      | A way to group related methods. For example, all the methods related to Masterscope on the class <b>FileBrowser</b> have the category Masterscope, not <b>FileBrowser</b> . This item appears only for methods.                                                                                                                                                     |                 |             |
| Specializes:     | The next higher class in the class hierarchy that contains a method with the same selector; only appear for methods. For example, <b>the manual entry for RectangularWindow.Open</b> would say that it specializes <b>Window.Open</b> , since <b>Window</b> is the first superclass of <b>RectangularWindow</b> that implements a method for <b>Open</b> .          |                 |             |
| Specializations: | The next lower class(es) in the class hierarchy that contains method(s) with the same selector; only appears for methods. For example, the manual entry for <b>Window.Open</b> would say that it has a specialization of <b>RectangularWindow.Open</b> since <b>RectangularWindow</b> is a subclass of <b>Window</b> and has its own version of <b>Open</b> method. |                 |             |
| Example:         | An example is often included to show how to use the item and what result it produces. Some examples may appear differently on your system, depending on the settings of various print flags. See Chapter 18, Reading and Printing, for details.                                                                                                                     |                 |             |

---

## References

---

The following books and manuals augment this manual.

*LOOPS Library Modules Manual*

*LOOPS Users' Modules Manual*

*Interlisp-D Reference Manual*

*Common Lisp: the Language* by Guy Steele

*Common Lisp Implementation Notes, Medley Release*

*Lisp Release Notes, Medley Release*

*Lisp Library Modules Manual, Medley Release*

[This page intentionally left blank]

In object-oriented programming, every object is described by a class. Instances are described by classes and classes are, in turn, described by metaclasses. The methods that an instance inherits are defined in the class definition of that instance and the methods that the class inherits are defined in the metaclass definition of that class's metaclass. Sending a message to an instance invokes method in class. Similarly, sending a message to class invokes method in metaclass.

The two classes **Class** and **MetaClass** are metaclasses of other classes. If **Class** or **MetaClass** refers to the metaclass, it appears in a bold typeface.

One method defined by a class's metaclass is **New**, which returns a new instance of a class. Different classes can initialize their instances in different ways. For example, one class may need to have certain values assigned to instance variables at creation, while another does not. The different forms of **New** are defined in separate metaclasses.

A class's metaclass is assigned when the class is created. A new class is created by sending a metaclass the message **New** or by specializing an already existing class. In the latter case, the metaclass defaults to the metaclass of the class's super class. The class's metaclass can be changed by directly editing the class definition.

This chapter discusses the metaclasses provided with LOOPS, describes pseudoclasses, explains how to define new metaclasses, and discusses the root class **Tofu**.

## 4.1 Specific Metaclasses

This section describes the metaclasses provided by LOOPS: **Class**, **MetaClass**, **AbstractClass**, and **DestroyedClass**. These metaclasses are shown in Figure 4-1.

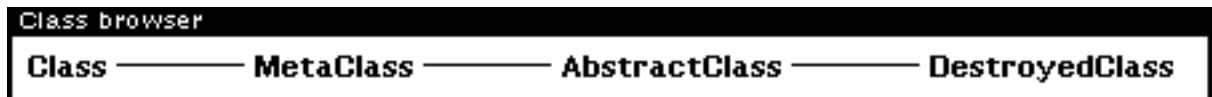


Figure 4-1. Class Browser Showing Metaclasses

### 4.1.1 Metaclass Class

**Class** is the default metaclass for LOOPS classes. When a class whose metaclass is **Class** receives the message **New**, it creates a new instance of itself and returns that instance. If this message is sent at the top level, the definition of the created instance is printed in the Executive window.

### 4.1.2 Metaclass MetaClass

---

**MetaClass** is the metaclass for all metaclasses and provides the message **New** to all metaclasses. For metaclasses, the result of sending the message **New** is the definition of a new metaclass. This is discussed in detail in Section 4.3, "Defining New Metaclasses."

### 4.1.3 Metaclass AbstractClass

---

If a class's metaclass is **AbstractClass**, then it cannot be instantiated. If an abstract class is sent the message **New**, the following message is printed to the TTY window.

```
#, ($C className) Abstract Class cannot be instantiated
```

To make a class an **AbstractClass**, either send the metaclass **AbstractClass** the message **New** or change the metaclass of the class definition directly using the editor.

Use an abstract class to define a class which should not have any instances. For example, consider mixin classes. Mixins are always used in conjunction with another class to create a subclass. Instances are created from the new subclass that has the mixin as one of its parents. Because mixins never have instances, they have **AbstractClass** as their metaclass.

As an example, consider a circuit simulation module that contains various classes such as **Resistors**, **Inductors**, **Batteries**, and **Wires**. A possibility is to define a super class for these classes called **AnalogDevice** to contain all the information common to all such classes: current, impedance, voltage drop, etc. This super class also holds all the methods common to the classes, such as **ApplyOhmsLaw**. Since **AnalogDevice** is not itself intended to be instantiated (only its subclasses are), its metaclass can be **AbstractClass** so that an error occurs if it is accidentally instantiated.

Note: Whenever **AnalogDevice** is specialized to create a new subclass, be sure to change its metaclass.

### 4.1.4 Metaclass DestroyedClass

---

**DestroyedClass** is the metaclass for classes that have been sent the message **Destroy** or **Destroy!** Trying to instantiate a **DestroyedClass** causes an error. Attempts to destroy a **DestroyedClass** have no effect.

4.2 PSEUDOCASSES

## 4.2 PSEUDOCASSES

---

## 4.2 Pseudoclasses

---

Pseudoclasses provide an object interface to Lisp data types, which are also known as Lisp objects. Pseudoclasses associate a class with the type name of a Lisp object. When messages are sent to Lisp objects of the named type, the messages are actually sent to the pseudoclass. Lisp objects which have pseudoclasses are considered pseudoinstances.

Pseudoclasses provide two special cases in the message-sending mechanism: for lists whose first element is a class, or for ordinary Lisp data types.

In the first case, the list's first element is used as the class to look up the method to be used.

In the second case, the class of the data type is found by using the **GetLispClass** function, which looks in an internal table based on the type name of the data type. If none is found, it is assumed to be **Tofu**. If found, the data type is considered a pseudoclass and instances of it pseudoinstances.

Pseudoclasses also provide special cases in the behavior of **GetValue** and **PutValue**, to allow simulation of variable or property access, as described below.

---

**(GetValue pseudoinstance varName propName)** [Function]

---

**Purpose:** A variation on the behavior of **GetValue** to simulate retrieving variable or property values on pseudoinstances.

**Behavior:** If **GetValue** is called with *self* bound to a pseudoinstance, then the method associated with the selector **GetValue** in the pseudoclass is called with the arguments:

*pseudoinstance varName propName*

**Arguments:**

*pseudoinstance* A Lisp object which has a pseudoclass.

*varName* The simulated variable name.

*propName* The simulated property name, or NIL.

**Returns:** The result of the call to the **GetValue** method in the pseudoclass.

---

**(PutValue pseudoinstance varName propName newValue)** [Function]

---

**Purpose:** A variation on the behavior of **PutValue** to simulate setting of variable or property values on pseudoinstances.

**Behavior:** If **PutValue** is called with *self* bound to a pseudoinstance, then the method associated with the selector **PutValue** in the pseudoclass is called with the arguments:

*instance varName newValue propName*

**Arguments:**

*pseudoinstance* A Lisp object which has a pseudoclass.

*varName* The simulated variable name.

*propName* The simulated property name, or NIL.

*newValue* The new value to be placed in the simulated slot.

**Returns:** The result of the call to the **PutValue** method in the pseudoclass.

---

**(GetLispClass obj)** [Function]

---

**Purpose:** Used by the system to compute a class corresponding to a Lisp data type.

**Behavior:** Gets the hash value for the key (**TYPENAME** *obj*) from an internal hash array.

- If this hash value is NIL, (\$ Tofu) is returned.
- If the hash value is not NIL and it is a class, it is returned.



- In all other cases, the hash value, which should be a function, is applied to *obj* and the result is returned.

Arguments: *obj*            A Lisp object.

Returns: Value depends on the hash value; see Behavior.

Example: The command

```
79← (GetLispClass (create annotatedValue))
```

returns

```
#, ($C AnnotatedValue)
```

---

**LispClassTable**

[Global Variable]

Purpose: Used by **GetLispClass** to map type names of Lisp objects to pseudoclasses.

Format: This hash table has EQ hashing. It contains pairs of symbol keys (a type name) and either classes, NIL, or a function object to be applied (see **GetLispClass**).

---

**4.2.1 Example**

This example creates a pseudoclass from the Lisp data type STRINGP.

1. Define a class **String** that receives its messages:

```
37← (DefineClass 'String)
#, ($C String)
```

2. Place an entry in the LispClass hash table to link the Lisp data type STRING to the **String** class.

```
38← (PUTHASH 'STRINGP ($ String) LispClassTable)
#, ($C String)
```

3. Add methods to **String** which will operate on Lisp STRINGPs, for example:

```
39← (DefineMethod ($ String) 'UpCase ' (self)
 ' (U-CASE self))
String.Upcase
```

This allows messages like the following:

```
40← (← "abc" UpCase)
"ABC"
```

4. Specialize **GetValue** and **PutValue** to allow element access in strings, for example:

```
41← (DefineMethod ($ String) 'GetValue ' (index)
 ' (NTHCHAR self index))
String.GetValue
```

```
42← (DefineMethod ($ String) 'PutValue ' (index value)
 ' (RPLSTRING self index value))
String.PutValue
```

This allows messages to access characters in strings, for example:

```
43← (← "abc" GetValue 2)
b

44← (← "abc" PutValue 2 'p)
"apc"
```

## 4.3 DEFINING NEW METACLASSES

## 4.3 DEFINING NEW METACLASSES

## 4.3 Defining New Metaclasses

A new metaclass must be defined if you want to create several classes for which a class message, such as **Destroy**, needs to be specialized. To create a new metaclass, an object of the class **MetaClass** must be instantiated. This is done by sending **MetaClass** the message **New**.

(← (\$ **MetaClass**) **New** *metaClassName* *supers*) [Method of Metaclass]

- Purpose:** Instantiates a new metaclass with **MetaClass** as its metaclass, *metaClassName* as its name, and *supers* as a list of its super classes.
- Behavior:** Evaluates *metaClassName*, which must evaluate to a symbol. The default for *supers* is (**Class**). If used, *supers* must evaluate to a list of classes. The message returns the new metaclass.
- Arguments:** *metaClassName*  
Name of the new metaclass; must evaluate to a symbol.  
*supers* List of classes.
- Categories:** **MetaClass**
- Specializes:** Class.New
- Specializations:** AbstractClass.New
- Example:** Assume the following **MetaClass** definition:

```
42← (← ($ MetaClass) New 'ListMetaClass' (Class))
#, ($C ListMetaClass)
```

The message **New** can then be defined for the metaclass, **ListMetaClass**. In this example, it saves all the instances created of a class with the metaclass **ListMetaClass**. The instances are stored as the value of the class property **AllInstances**. Define the message **New** using **DefineMethod** as follows:

```
DefineMethod ($ ListMetaClass) 'New' (name arg1 arg2 arg3
arg4 arg5)
' ((* * Create an instance and add it to
list in class)
(LET ((newObj (←Super)))

(* * newObj is the instance returned by
sending the New message provided by
the class CLASS.)

(PutClass
self
(CONS newObj (LISTP (GetClassHere
```

```
self 'AllInstances)))
'AllInstances)

(* * LISTP returns list or
 NIL if not a list)
(* * GetClassHere returns the value
 of a property of a class.)

newObj]
```

Now a new class can be defined by sending #,(\$C ListMetaClass) the message **New**. The result is a new class with **ListMetaClass** as the metaclass.

```
43←(←($ ListMetaClass) New 'Book)
#, ($C Book) [New class called Book]

44←(←($ Book) New 'Book1)
#, ($C Book1)

45←(←($ Book) New 'Book2)
#, ($C Book2)

46←(GetClass ($ Book) 'AllInstances)
#, ($C Book2) #, ($C Book1) [List of all instances created so far]
```

4.4 TOFU

4.4 TOFU

4.4 Tofu

The highest class in the LOOPS hierarchy is **Tofu**, which is an acronym for Top of the Universe. It is the simplest class, having no instance variables and only three defined messages:

- **MessageNotUnderstood**
- **MethodNotFound**
- **SuperMethodNotFound**

Figure 4-2 shows specializations of **Tofu**. The most familiar specialization of **Tofu** is the class **Object**, which is the root of the most of the other classes. Another specialization of **Tofu** is **AnnotatedValue**. **AnnotatedValue** is used with active values (see Chapter 8, Active Values).

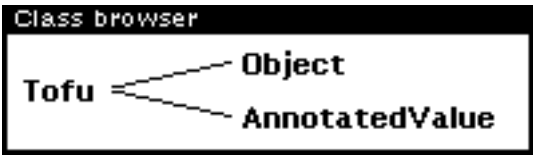


Figure 4-2. Specializations of Tofu

If another evaluation protocol or scheme for catching error conditions is needed, specialize **Tofu** and define all the methods required for handling data, usually some subset of the methods of **Object**. Specializing Tofu should only be necessary on very rare occasions.

The following table shows the methods in this section.

| Functionality | Type | Description |
|---------------|------|-------------|
|---------------|------|-------------|

|                             |        |                                                                                                                    |
|-----------------------------|--------|--------------------------------------------------------------------------------------------------------------------|
| <b>MessageNotUnderstood</b> | Method | Provides the error handling mechanism for when a message is sent to an object that cannot respond to that message. |
| <b>MethodNotFound</b>       | Method | Provides some intermediate checking before sending the message <b>MessageNotUnderstood</b> .                       |
| <b>SuperMethodNotFound</b>  | Method | Provides some intermediate checking before sending the message <b>MessageNotUnderstood</b> .                       |

(← *self* **MessageNotUnderstood** *selector messageArguments superFlg*) [Method of Tofu]

|                   |                                                                                                                                                                                                                               |                                                                   |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| Purpose/Behavior: | Provides the error handling mechanism for when a message is sent to an object that cannot respond to that message.<br><br>Calls <b>ERROR</b> with a list which includes <i>self</i> , <i>selector</i> , and "not understood." |                                                                   |
| Arguments:        | <i>self</i>                                                                                                                                                                                                                   | An object receiving a message with the selector <i>selector</i> . |
|                   | <i>selector</i>                                                                                                                                                                                                               | A selector.                                                       |
|                   | <i>messageArguments</i>                                                                                                                                                                                                       | A list of the arguments to the message.                           |
|                   | <i>superFlg</i>                                                                                                                                                                                                               | Used internally.                                                  |
| Returns:          | See Behavior.                                                                                                                                                                                                                 |                                                                   |
| Categories:       | Tofu                                                                                                                                                                                                                          |                                                                   |
| Specializations:  | Object                                                                                                                                                                                                                        |                                                                   |

(← *self* **MethodNotFound** *selector*) [Method of Tofu]

|                   |                                                                                              |                                                                   |
|-------------------|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| Purpose/Behavior: | Provides some intermediate checking before sending the message <b>MessageNotUnderstood</b> . |                                                                   |
| Arguments:        | <i>self</i>                                                                                  | An object receiving a message with the selector <i>selector</i> . |
|                   | <i>selector</i>                                                                              | A selector.                                                       |
| Returns:          | Used for side effect only.                                                                   |                                                                   |
| Categories:       | Tofu                                                                                         |                                                                   |

(← *self* **SuperMethodNotFound** *selector classOfSendingMethod*) [Method of Tofu]

|                   |                                                                                              |                                                                   |
|-------------------|----------------------------------------------------------------------------------------------|-------------------------------------------------------------------|
| Purpose/Behavior: | Provides some intermediate checking before sending the message <b>MessageNotUnderstood</b> . |                                                                   |
| Arguments:        | <i>self</i>                                                                                  | An object receiving a message with the selector <i>selector</i> . |
|                   | <i>selector</i>                                                                              | A selector.                                                       |
|                   | <i>classOfSendingMethod</i>                                                                  | The class with the method that contains a ← <b>Super</b> .        |
| Returns:          | Used for side effect only.                                                                   |                                                                   |
| Categories:       | Tofu                                                                                         |                                                                   |

[This page intentionally left blank]

This chapter discusses the various ways to access data:

- Generalized Get and Put functions
- Accessing data in instances
- Accessing data in classes

## 5.1 Generalized Get and Put Functions

These functions support generalized instance variable and property access for LOOPS objects. They can be very useful for implementing methods that support new types of conditional accessing; they have been used to simplify code in the active values system, for example.

This section deals with the following functions:

| Name             | Type     | Description                                                                                                                                                 |
|------------------|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GetIt</b>     | Function | Retrieves values from instance variables and properties.                                                                                                    |
| <b>GetItOnly</b> | Function | Like <b>GetIt</b> , but returns active values on a variable/property without triggering them.                                                               |
| <b>GetItHere</b> | Function | Like <b>GetIt</b> , but returns active values on a variable/property without triggering them; does not observe <b>NotSetValue</b> as <b>GetItOnly</b> does. |
| <b>PutIt</b>     | Function | Stores values into instance variables and properties.                                                                                                       |
| <b>PutItOnly</b> | Function | Like <b>PutIt</b> , but stores by smashing active values on a variable/property without triggering them.                                                    |

(GetIt self varOrSelector propName type)

[Function]

Purpose: Retrieves values from instance variables and properties.

Behavior: Varies according to the arguments.

- If *type* is 'IV or NIL
  - If *self* is an instance, this is equivalent to **(GetValue self varOrSelector propName)**
  - If *self* is a class, this is equivalent to **(GetClassIV self varOrSelector propName)**
- If *type* is 'CV, this is equivalent to **(GetClassValue self varOrSelector propName)**

- If *type* is 'CLASS, this is equivalent to (**GetClass** *self* (**OR** *varOrSelector* *propName*))
- If *type* is 'METHOD, this is equivalent to (**GetMethod** *self* *varOrSelector* *propName*)

Arguments: *self* A class or an instance.  
*varOrSelector* An instance variable name or the name of a method.  
*propName* Property name.  
*type* Specifies the type of the object *self*.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetIt ($ Window) 'doc NIL 'CLASS)
```

returns

```
"A Loops object that represents a window"
```

---

**(GetItOnly** *self* *varOrSelector* *propName* *type*)

[Function]

Purpose: Retrieves values from instance variables and properties without triggering active values.

Behavior: Varies according to the arguments.

- If *type* is 'IV or NIL
  - If *self* is an instance, this is equivalent to (**GetValueOnly** *self* *varOrSelector* *propName*)
  - If *self* is a class, this is equivalent to (**GetClassIV** *self* *varOrSelector* *propName*)
- If *type* is 'CV, this is equivalent to (**GetClassValueOnly** *self* *varOrSelector* *propName*)
- If *type* is 'CLASS, this is equivalent to (**GetClassOnly** *self* (**OR** *varOrSelector* *propName*))
- If *type* is 'METHOD, this is equivalent to (**GetMethodOnly** *self* *varOrSelector* *propName*)

Arguments: *self* A class or an instance.  
*varOrSelector* An instance variable name or the name of a method.  
*propName* Property name.  
*type* Specifies the type of the object *self*.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetItOnly (GetClassValue ($ LoopsIcon) 'Prototype) 'window)
```

returns the LoopsWindowAV that holds the image of the LOOPS icon. Calling **GetIt** with similar arguments returns the Lisp window object held by that LoopsWindowAV.

**(GetItHere self varOrSelector propName type)**

[Function]

**Purpose:** Retrieves values from instance variables and properties without triggering active values; does not observe **NotSetValue** like **GetItOnly**.

**Behavior:** Varies according to the arguments.

- If *type* is 'IV or NIL
- If *self* is an instance, this is equivalent to **(GetIVHere self varOrSelector propName)**
- If *self* is a class, this is equivalent to **(GetClassIVHere self varOrSelector propName)**
- If *type* is 'CV, this is equivalent to **(GetCVHere self varOrSelector propName)**
- If *type* is 'CLASS, this is equivalent to **(GetClassHere self (OR varOrSelector propName))**
- If *type* is 'METHOD, this is equivalent to **(GetMethodHere self varOrSelector propName)**

**Arguments:** *self* A class or an instance.  
*varOrSelector* An instance variable name or the name of a method.  
*propName* Property name.  
*type* Specifies the type of the object *self*.

**Returns:** Value depends on the arguments; see Behavior.

**Example:** The command

```
(GetItHere (GetClassValue ($ LoopsIcon) 'Prototype) 'title)
```

returns the value of **NotSetValue**. Calling **GetIt** with similar arguments returns the default value for this instance variable, NIL.

**(PutIt self varOrSelector newValue propName type)**

[Function]

**Purpose:** Stores values into instance variables and properties.

**Behavior:** Varies according to the arguments.

- If *type* is 'IV or NIL
- If *self* is an instance, this is equivalent to **(PutValue self varName newValue propName)**
- If *self* is a class, this is equivalent to **(PutClassIV self varName newValue propName)**
- If *type* is 'CV, this is equivalent to **(PutClassValue self varName newValue propName)**
- If *type* is 'CLASS, this is equivalent to **(PutClass self newValue (OR varName propName))**

**Arguments:** *self* A class or an instance.



*varName*      An instance variable name.

*propName*    Property name.

*type*          Specifies the type of the object *self*.

Returns:      Value depends on the arguments; see Behavior.

Example:      The command

```
(PutIt (GetClassValue ($ LoopsIcon) 'Prototype) 'title "foo")
```

sets the instance variable **title** of the LOOPS icon prototype to "foo". This can be verified by inspecting (GetClassValue (\$ LoopsIcon) 'Prototype) and examining the title slot.

**(PutItOnly self varOrSelector newValue propName type)** [Function]

---

Purpose:        Stores values into instance variables and properties and smashes any active values it finds in its way without triggering them.

Behavior:     Varies according to the arguments.

- If *type* is 'IV or NIL
- If *self* is an instance, this is equivalent to  
  **(PutValueOnly self varName newValue propName)**
- If *self* is a class, this is equivalent to  
  **(PutClassIV self varName newValue propName)**
- If *type* is 'CV, this is equivalent to  
  **(PutClassValueOnly self varName newValue propName)**
- If *type* is 'CLASS, this is equivalent to  
  **(PutClassOnly self newValue (OR varName propName))**

Arguments:    *self*            A class or an instance.

*varName*      An instance variable name.

*propName*    Property name.

*type*          Specifies the type of the object *self*.

Returns:      Value depends on the arguments; see Behavior.

Example:      If the inspector from the **PutIt** example is used to set a break on the the instance variable **title** of the LOOPS icon prototype, then doing

```
(PutItOnly (GetClassValue ($ LoopsIcon) 'Prototype) 'title "mumble")
```

will set the instance variable **title** to "mumble" while smashing the trace active value.

## 5.2 ACCESSING DATA IN INSTANCES

## 5.2 ACCESSING DATA IN INSTANCES

---

## 5.2 Accessing Data in Instances

Two kinds of variables are associated with an instance:

- Its local instance variables, also referred to as IVs.
- The class variables, also referred to as CVs, that it shares with all instances of the class.

The data contained within instances are the values of instance variables and associated properties as well as a pointer to the class that describes the instance. Details of the LOOPS implementation determine exactly when the values of instance variables are stored within an instance. In some cases, the system must look to the class to find the values of instance variables. In general, you do not need to be concerned with this distinction; however, the details of it are covered in Chapter 2, Instances.

The types of data that an instance may contain is not limited. The values for an instance variable or a class variable can be any Lisp or LOOPS data structure.

The active value is a special case of data. When you try to access a variable with an active value as its value, the active value may be returned, depending upon the type of access. Normally, however, data computed by the active value is returned, not the active value. The details of how this computation is performed is described in Chapter 8, Active Values.

Instance variable names and class variable names are symbols and are not necessarily unique to each class. Although it is possible to use the same symbol for both a class variable name and an instance variable name, it is advisable not to do this since some of the LOOPS functionality examines both the instance variables and class variables in the search for data. See the method **IVMissing** in the class **Object**.

This section deals with the following functions and methods. See the *LOOPS Library Modules Manual* for information on how these interact with Masterscope.

| Name                     | Type     | Description                                                                                                                                                                           |
|--------------------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GetValue</b>          | Function | Finds the value of an instance variable.                                                                                                                                              |
| <b>Get</b>               | Method   | Finds the value of an instance variable.                                                                                                                                              |
| <b>PutValue</b>          | Function | Writes the value of an instance variable.                                                                                                                                             |
| <b>Put</b>               | Method   | Writes the value of an instance variable.                                                                                                                                             |
| <b>GetValueOnly</b>      | Function | Finds the value of an instance variable without triggering active values.                                                                                                             |
| <b>PutValueOnly</b>      | Function | Writes the value of an instance variable without triggering active values.                                                                                                            |
| <b>GetClassValue</b>     | Function | Returns the value of a class variable.                                                                                                                                                |
| <b>PutClassValue</b>     | Function | Changes the value of a class variable. The change occurs within the class and therefore causes all instances to access the new value of the variable.                                 |
| <b>GetClassValueOnly</b> | Function | Returns the value of a class variable; does not trigger active values.                                                                                                                |
| <b>PutClassValueOnly</b> | Function | Changes the value of a class variable. The change occurs within the class and therefore causes all instances to access the new value of the variable. Does not trigger active values. |

**GetIVHere**

Function

Gets the value stored in an instance variable without invoking active values.

---

**(GetValue self varName propName)**

[Function]

Purpose: Finds the value of an instance variable when *varName* and *propName* are to be computed.

Behavior: Varies according to the arguments.

- If *self* is an instance and *propName* is NIL, this returns the value of the instance variable *varName*. If there is no instance variable of the name *varName* and there is a class variable of that name, this returns the value of the class variable. See the **IVMissing** message for a complete discussion of this behavior. If there is neither an instance variable or class variable of that name, a break occurs.
- If *self* is an instance and *propName* is non-NIL, this returns the value of the property *propName* of the instance variable or class variable *varName*. If there is no property of the name, *propName*, this returns the value of the variable **NoValueFound**.
- If the value of *varName* (or *propName* if it is non-NIL) is an active value, the active value is activated.
- If *self* is not an instance, this calls **(GetIt self varName propName 'IV)**

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments: *self*            A class or an instance.  
*varName*        Instance or class variable name.  
*propName*      Property name.

Returns: Value depends on the arguments; see Behavior.

Example: Given that

```
32←(← ($ window1) Shape '(100 200 300 400))
(100 200 300 400)
```

then

```
33←(GetValue ($ window1) 'width)
300
```

```
34←(GetValue ($ window1) 'LeftButtonItem)
((Update ...))
```

---

**(← self Get varName propName)**

[Method of Object]

Purpose/Behavior: Method version of **GetValue**.

Arguments: See **GetValue**.

Categories: Object

---

**(PutValue self varName newValue propName)**

[Function]

Purpose: Writes the value of an instance variable when *varName* and *propName* are to be computed.

Behavior: Varies according to the arguments.

- If *self* is an instance and *propName* is NIL, this changes the value of the instance variable *varName* to *newValue*. This returns *newValue*. If *varName* is not an instance variable of *self*, this causes a break.
- If *self* is an instance and *propName* is non-NIL, this changes the value of the property *propName* of the instance variable *varName* to *newValue*. If *propName* is not already a property of *varName*, it is added. This returns *newValue*.
- If the value of *varName* (or *propName* if it is non-NIL) is an active value, the active value is activated.
- If *self* is a class, this calls  
(**PutIt** *self varName newValue propName* 'IV)

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments: *self* A class or an instance.  
*varName* Instance name or class name.  
*newValue* The new value for *varName* or *propName*.  
*propName* Property name.

Returns: Value depends on the arguments; see Behavior.

Example: (PutValue (\$ window1) 'width 120)

---

(← *self* **Put** *varName newValue propName*) [Method of Object]

---

Purpose/Behavior: Method version of the function **PutValue**.

Arguments: See **PutValue**.

Categories: Object

Specializations: Class

---

(**GetValueOnly** *self varName propName*) [Function]

---

Purpose: Similar to **GetValue**, except that it overrides the active value mechanism.

Behavior: See **GetValue**. If the value found is an active value, it is returned without triggering its side effects.

Arguments: See **GetValue**.

Returns: See Behavior.

Example: The following expressions compare **GetValue** and **GetValueOnly**

```
35←(GetValue ($ window1) 'window)
{WINDOW}#nn, mmmm

36←(GetValueOnly ($ window1) 'window)
#,$AV LispWindowAV ...)
```

---

(**PutValueOnly** *self varName newValue propName*) [Function]

---

Purpose: Similar to **PutValue**, except that it overrides the active value mechanism.

|            |                                                                                                                   |
|------------|-------------------------------------------------------------------------------------------------------------------|
| Behavior:  | See <b>PutValue</b> . The argument <i>newValue</i> overwrites any active value on the slot without triggering it. |
| Arguments: | See <b>PutValue</b> .                                                                                             |
| Returns:   | Used for side effect only.                                                                                        |

---

**(GetClassValue** *self varName propName*) [Function]

---

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Returns the value of a class variable.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Behavior:  | <p>Varies according to the arguments.</p> <ul style="list-style-type: none"><li>• If <i>propName</i> is NIL, this returns the value of the class variable <i>varName</i>. If <i>varName</i> is not a class variable, a break occurs.</li><li>• If <i>propName</i> is non-NIL, this returns the value of the property, <i>prop</i>, of the class variable <i>varName</i>. If <i>varName</i> has no property of that name, the value of the variable <b>NoValueFound</b> is returned.</li></ul> <p>See the <i>LOOPS Library Modules Manual</i> about interaction with Masterscope.</p> |
| Arguments: | <p><i>self</i>            An instance or a class.</p> <p><i>varName</i>        Class variable name of <i>self</i>.</p> <p><i>propName</i>       Property name for class variable <i>varName</i>; may be NIL.</p>                                                                                                                                                                                                                                                                                                                                                                     |
| Returns:   | Value depends on the arguments; see Behavior.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Example:   | <p>The following commands show a variety of returned values.</p> <pre>37←(GetClassValue (\$ window1) 'window)</pre> <p>This breaks, since window is not a class variable of <b>Window</b>.</p> <pre>38←(GetClassValue (\$ window1) 'LeftButtonItem) ((Update ...))</pre> <pre>39←(GetClassValue (\$ window1) 'LeftButtonItem 'qwerty) NIL</pre>                                                                                                                                                                                                                                      |

---

**(PutClassValue** *self varName newValue propName*) [Function]

---

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Changes the value of a class variable. The change occurs within a class and therefore causes a class variable lookup by other instances to find the new value.                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Behavior:  | <p>Varies according to the arguments.</p> <ul style="list-style-type: none"><li>• If <i>propName</i> is NIL, this changes the value of the class variable <i>varName</i> to <i>newValue</i>. If <i>varName</i> is not a class variable, this breaks.</li><li>• If <i>propName</i> is non-NIL, this changes the value of the property, <i>propName</i>, of the class variable <i>varName</i> to <i>newValue</i>. If <i>varName</i> has no property of that name, the property is added.</li></ul> <p>See the <i>LOOPS Library Modules Manual</i> about interaction with Masterscope.</p> |
| Arguments: | <p><i>self</i>            An instance or a class.</p> <p><i>varName</i>        Class variable name of <i>self</i>.</p> <p><i>newValue</i>       Value to be assigned to class variable or property name.</p>                                                                                                                                                                                                                                                                                                                                                                            |

*propName* Property name for class variable *varName*; may be NIL.

Returns: *newValue*

Example: The following command breaks since **left** is not a class variable name of **Window**.

```
40←(PutClassValue ($ window1) 'left 1234)
```

The command

```
41←(PutClassValue ($ window1) 'TitleItems 1234)
```

changes the value of **TitleItems**. The command

```
42←(PutClassValue ($ window1) 'TitleItems 123 'asdf)
```

adds the property **asdf** with the value 123 to **TitleItems**.

**(GetClassValueOnly self varName propName)**

[Function]

Purpose: Gets the value of a class variable without triggering active values.

Behavior: Varies according to the arguments.

- If *propName* is NIL, this returns the value of the class variable *varName* without triggering active values. If *varName* is not a class variable, this breaks.
- If *propName* is non-NIL, this returns the value of the property, *propName*, of the class variable *varName* without triggering active values. If *varName* has no property of that name, the value of the variable **NotSetValue** is returned.

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments: *self* An instance or a class.

*varName* Class variable name for *self*.

*propName* Property name of class variable *varName*; may be NIL.

Returns: Value depends on the arguments; see Behavior.

Example: The following command returns the value of the variable **NotSetValue** since **LeftButtonItem** has no property of the name **qwerty**.

```
43←(GetClassValueOnly ($ window1) 'LeftButtonItem 'qwerty)
#,NotSetValue
```

**(PutClassValueOnly self varName newValue propName)**

[Function]

Purpose: Changes the value of a class variable without triggering active values. The change occurs within a class and therefore causes a class variable lookup by other instances to find the new value.

Behavior: The behavior is the same as **PutClassValue** except that the value stored does not trigger an active value, but overwrites it instead.

Arguments: *self* An instance or a class.

*varName* Class variable name of *self*.

*newValue* Value to be assigned to class variable or property name.

*propName*    Property name for class variable *varName*; may be NIL.

Returns:    *newValue*

---

**(GetIVHere self *varName propName*)** [Function]

---

Purpose:    Gets the value stored in an instance without invoking active values.

Behavior:    Returns the value of *varName* (or the property, *propName*, if it is non-NIL) without triggering active values. If the value of *varName* (or *propName*) is not yet stored in *self*, the value of the variable **NotSetValue** is returned.

See the *LOOPS Library Modules Manual* about interaction with Masterscope.

Arguments:    *self*            Must be an instance.

*varName*       Instance variable of *self*.

*propName*      Property name for variable *varName*; may be NIL.

Returns:    Value depends on the arguments; see Behavior.

Example:    Given that

```
44←(← ($ Window) New 'w2)
#,$(& Window (NEW0.1Y%.:;h.eN6 . 496))
```

then

```
45←(GetIVHere ($ w2) 'left)
,NotSetValue
```

After entering the command

```
46←(PutValue ($ w2) 'left 123)
123
```

then

```
47←(GetIVHere ($ w2) 'left)
123
```

---

### 5.2.1 Compact Accessing Forms

---

When you write methods for classes that you have defined, there are a number of accesses to the data contained in the object bound to the method argument *self*. The following forms have been created to allow a more concise notation for these accesses.

| Name | Type  | Description                                                                              |
|------|-------|------------------------------------------------------------------------------------------|
| @    | Macro | Provides compact <b>GetValue</b> and <b>GetClassValue</b> forms.                         |
| @*   | Macro | Provides compact <b>GetValue</b> forms.                                                  |
| ←@   | Macro | Provides compact <b>PutValue</b> and <b>PutClassValue</b> forms and assigns a new value. |

---

**(@ *accessPath*)** [Macro]

---

Purpose:    Provides compact **GetValue** or **GetClassValue** forms.

Behavior: The *accessPath* can be one, two, or three arguments.

- If the *accessPath* is one argument, *self* is assumed to be the object and the argument points to an instance variable. This is the most common usage in methods in which you need to get the value of an instance variable contained in *self*. For example,

```
(@ iv1)
```

translates to

```
(GetValue self 'iv1).
```

- If the *accessPath* is two arguments, the first argument is an object and the second argument is an instance variable. For example,

```
(@ ($ w) left)
```

translates to

```
(GetValue ($ w) 'left).
```

- If the *accessPath* is three arguments, the first argument is an object, the second argument is an instance variable, and the third argument is a property. For example,

```
(@ ($ w) menus DontSave)
```

translates to

```
(GetValue ($ w) 'menus 'DontSave).
```

When programming using objects, one object often points to another object. For example, the value of an instance variable is another object. Using different *accessPath* forms allows you to write accesses into objects that are nested inside of other objects. As an example, assume an object (\$ pipe) has an instance variable named *output* with a value (\$ tank), which has an instance variable named *level*. The command

```
(@ ($ pipe) output:level)
```

which is equivalent to

```
(@ (@ ($ pipe) output) level)
```

gets the value of the instance variable *level* of (\$ tank).

The ":" is a delimiter that indicates instance variable access. The following table shows all the delimiters.

| Delimiter | Description                                                                                                              |
|-----------|--------------------------------------------------------------------------------------------------------------------------|
| :         | Indicates instance variable access.                                                                                      |
| ::        | Accesses the value of a class variable whose name follows the double colon.                                              |
| :,        | Accesses the value of a property whose name follows the colon-comma.                                                     |
| .         | Sends a message to the object with the selector following the period.                                                    |
| !         | Evaluates the next expression.                                                                                           |
| \         | States that the next symbol refers to a Lisp symbol. This is often used in conjunction with the exclamation mark, above. |



**\$** States that the next expression is a LOOPS object.

You can test forms using these delimiters by evaluating  
(Parse@ (LIST accessPath) 'IV) .

Arguments: *accessPath* One, two, or three arguments; refer to Behavior.

Returns: See Behavior.

Example: The following examples show the (@ accessPath) form, the Parse@ test, and the translation.

1. (@ foo)  
(Parse@ (LIST 'foo) 'IV)  
(GetValue self 'foo)
2. (@ ::fie:foe)  
(Parse@ (LIST '::(GetValue (GetClassValue self 'fie) 'foe)

The following three examples are rarely seen in code, but they are additional examples of the expressions that can be interpreted by the system.

3. (@ foo::!::fum)  
(Parse@ (LIST 'foo::!::fum) 'IV)  
(GetClassValue (GetValue self 'foo) (GetClassValue self 'fum))
4. (@ (\$ w) fie:,foe.fum)  
(Parse@ (LIST '(\$ w) 'fie:,foe.fum) 'IV)  
(← (GetValue (\$ w) 'fie 'foe) fum)
5. (@ \$fie.foe:!\fum.!foo)  
(Parse@ (LIST '\$fie.foe:!\fum.!foo) 'IV)  
(←! (GetValue (← (GetObjectRec 'fie) foe) fum) (GetValue self 'foo))

---

**(@\* accessPath)**

[Macro]

Purpose/Behavior: Provides a concise form for writing embedded **GetValue** forms.

Arguments: *accessPath* An object followed by an arbitrary number of instance variable names.

Returns: The value of a nested instance variable.

Example: The command

```
(@* ($ foo) a b c)
```

translates to

```
(GetValue (GetValue (GetValue ($ foo) 'a) 'b) 'c)
```

---

**(←@ accessPath newValue)**

[Macro]

Purpose/Behavior: Similar to the @ macro, but used to assign a new value instead of reading a value. Evaluates *newValue*.

Arguments: *accessPath* See Behavior in the @ macro.

*newValue* Value to be assigned to variable indicated by *accessPath*.

Returns: *newValue*

Example: The following examples show the ( $\leftarrow$ @ accessPath) form, the Parse@ test, and the translation.

1. ( $\leftarrow$ @ foo 1234)  
 (ParsePut@ (LIST 'foo 1234) 'IV)  
 (PutValue self 'foo 1234)
2. ( $\leftarrow$ @ (\$ w) ::left 1234)  
 (ParsePut@ (LIST (\$ w) ' ::left 1234) 'IV)  
 (PutClassValue #.(\$ w) 'left 1234)
3. ( $\leftarrow$ @ (\$ w) menus DontSave 'Any)  
 (ParsePut@ (LIST (\$ w) 'menus 'DontSave '(QUOTE Any)) 'IV)  
 (PutValue #.(\$ w) 'menus 'Any 'DontSave)

## 5.2.2 Support for Changetran

Interlisp uses Changetran to provide an extensive set of facilities for expressing changes to structures, such as push, pushnew, pop, add, change, by using access expressions. You can use any LOOPS access expression in a Changetran context, so that you can now write expressions such as:

```
(push (@ v1) newTop)
(change (@ x) newValue)
(pushnew (@ colors:,truck) 'red)
(pop (@ ::cv17))
(add (@ x:y:z) 37)
```

The first two are equivalent to:

```
(PushValue self 'v1 (CONS newTop(@ V1)))
(_@ x newValue)
```

This uniform interface allows simpler expressions for changes, and arbitrary extensions through Changetran. See the *Interlisp-D Reference Manual* for more information on Changetran.

## 5.3 ACCESSING DATA IN CLASSES

### 5.3 ACCESSING DATA IN CLASSES

## 5.3 Accessing Data in Classes

A number of functions and methods are available for reading and storing data within classes. Some of these change existing data, and others change the structure of the class by adding variables or properties.

When reading or storing data, some of these functions trigger any active values that are associated with that data. See Chapter 8, Active Values, for a discussion of their behavior.

### 5.3.1 Metaclass and Property Access

Associated with a class are a metaclass and properties. This section describes the following functions to manipulate their values.

| Name            | Type     | Description                                |
|-----------------|----------|--------------------------------------------|
| <b>GetClass</b> | Function | Obtains a class's metaclass or properties. |

|                     |          |                                                                                        |
|---------------------|----------|----------------------------------------------------------------------------------------|
| <b>PutClass</b>     | Function | Changes the metaclass or class properties of a class.                                  |
| <b>GetClassOnly</b> | Function | Obtains a class's metaclasses or properties without triggering active values.          |
| <b>PutClassOnly</b> | Function | Changes the metaclass or class properties of a class without triggering active values. |
| <b>GetClassHere</b> | Function | Obtains a property local to the class.                                                 |

---

**(GetClass *classRec* *propName*)** [Function]

---

Purpose: Obtains a class's metaclass or properties by following metaclass links.

Behavior: Sends the message **GetClassProp** to *classRec* and passes *propName* as an argument.

Varies according to the arguments.

- If *propName* is NIL, this returns the *class*'s metaclass.
- If *propName* is non-NIL, this looks first in *class* for that property. If it cannot find it there, it looks through *class*'s metaclass links.
- If no property is found, the value of the variable **NotSetValue** is returned.

Arguments: *classRec*    Pointer to a class.  
*propName*    Property name.

Returns: See Behavior.

Example: The following commands show the variety of returned values.

```
31←(GetClass ($ Window))
#,$C Class)

32←(GetClass ($ Window) 'doc)
" A LOOPS object which represents a window"

33←(GetClass ($ IconWindow) 'doc)
"An icon window that appears as an irregular shaped image
on the screen -- See the ICONW Library utility"
```

---

**(PutClass *classRec* *newValue* *propName*)** [Function]

---

Purpose: Changes the metaclass or class properties of a class.

Behavior: Varies according to the arguments.

- If *propName* is NIL, this changes the metaclass of *classRec* to *newValue*. If *newValue* is not a class or the name of a class, this causes a break.
- If *propName* is non-NIL and *classRec* already has this property, this triggers an active value on *propName* if it exists and changes the value of *propName* to *newValue*.
- If *propName* is non-NIL and *classRec* does not have this property, the property is added with the value *newValue*.

Marks the class *classRec* as changed.

Arguments: *classRec*    Pointer to a class.

*newValue* See Behavior.

*propName* Property name.

Returns: Newly created class object.

Example: The following command changes the **doc** property of class **Datum**:

```
66←(DefineClass 'Datum)
#,($C Datum)
```

```
67←(PutClass ($ Datum) '(* this is the updated doc for class Datum) 'doc)
(* this is the updated doc for class Datum)
```

**(GetClassOnly** *classRec propName*)

[Function]

Purpose: Obtains a class's metaclass or properties by following superclass links, without triggering active values.

Behavior: Varies according to the arguments.

- If *propName* is NIL, this returns the *classRec*'s metaclass.
- If *propName* is non-NIL, this looks first in *classRec* for that property. If it cannot find it there, it looks through *classRec*'s supers links. This returns the value of the property found without triggering active values.
- If no property is found, the value of the variable **NotSetValue** is returned.

Arguments: *classRec* Pointer to a class.

*propName* Property name.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetClassOnly ($ IconWindow) 'doc)
```

returns

```
"An icon window that appears as an irregular shaped image
on the screen -- See the ICONW Library utility"
```

**(PutClassOnly** *classRec newValue propName*)

[Function]

Purpose: Changes the metaclass or class properties without triggering active values.

Behavior: Varies according to the arguments:

- If *propName* is NIL, this changes the metaclass of *classRec* to *newValue*. If *newValue* is not a class or the name of a class this causes a break.
- If *propName* is non-NIL and *classRec* already has this property, this changes the value of *propName* to *newValue*. Any active values are replaced.
- If *propName* is non-NIL and *classRec* does not have this property, the property is added with the value *newValue*.

The class *classRec* is marked as changed.

Arguments: *classRec* Pointer to a class.

*newValue*    A class or the name of a class.

*propName*    NIL or the name of a class property.

Returns:    *newValue*

---

**(GetClassHere classRec propName)** [Function]

---

Purpose:    Obtains property local to class.

Behavior:    Gets the class property without triggering active values or inheritance. If there is no local property the value of **NotSetValue** is returned.

Arguments:    *classRec*    Pointer to a class.

*propName*    NIL or the name of a class property.

Returns:    *newValue*

Example:    The command

```
(GetClassHere ($ ActiveValue) 'doc)
```

returns

```
#,NotSetValue
```

---

### 5.3.2 Class Variable Access

---

A class variable can be thought of as being shared by all instances of that class and by all instances of any of its subclasses. This section describes how to access class variables with the functions shown in the following table.

| Name                     | Type     | Description                                                                                       |
|--------------------------|----------|---------------------------------------------------------------------------------------------------|
| <b>GetClassValue</b>     | Function | Returns the value of a class variable or property.                                                |
| <b>PutClassValue</b>     | Function | Stores a value in a class variable or property.                                                   |
| <b>GetClassValueOnly</b> | Function | Returns the value of a class variable or property, without triggering active values.              |
| <b>PutClassValueOnly</b> | Function | Stores a value in a class variable or property, without triggering active values.                 |
| <b>GetCVHere</b>         | Function | Returns the value of a class variable in a particular class without looking for inherited values. |
| <b>PutCVHere</b>         | Function | Stores a class variable locally with a value if it is not local.                                  |

---

**(GetClassValue self varName prop)** [Function]

---

Purpose:    Returns the value of a class variable or property.

Behavior:    Varies according to the arguments.

If *self* is an instance, the lookup begins at the class of the instance, since instances do not have class variables stored locally. If *self* is a class, the lookup is in that class.

- If *prop* is NIL, **GetClassValue** returns the value of the class variable *varName*. If *varName* is not found, this breaks.
- If *prop* is non-NIL, **GetClassValue** returns the value of the property *prop*, associated with the class variable *varName*. If the value is an active value, it is activated. If *varName* has no property *prop*, this returns the value of the variable **NoValueFound**.

If the class does not have a class variable *varName*, **GetClassValue** searches through the super classes of the class until it finds *varName*. Since this is rare, class variables are stored only in the class in which they are defined, and the runtime search is necessary.

Arguments: *self*            An instance or a class.  
               *varName*        A class variable name.  
               *prop*            Property name.

Returns: Value depends on the arguments; see Behavior.

Example: Given that

```
(← ($ Window) New 'window1)
```

then the command

```
(GetClassValue ($ window1) 'LeftButtonItem)
```

returns the same value as the command

```
(GetClassValue ($ Window) 'LeftButtonItem)
```

The command

```
(GetClassValue ($ Window) 'abcde)
```

breaks. The command

```
(GetClassValue ($ Window) 'LeftButtonItem 'wxyz)
```

returns the value of **NoValueFound**.

**(PutClassValue self varName newValue propName)**

[Function]

Purpose: Stores a value in a class variable or property.

Behavior: Varies according to the arguments.

If *self* is an instance, the lookup begins at the class of the instance, since instances do not have class variables stored locally. If *self* is a class, the lookup is in that class.

- If *prop* is NIL, **PutClassValue** changes the value of the class variable *varName*.
- If *prop* is non-NIL, **PutClassValue** stores *newValue* as the value of the property, *prop*. If an active value is the current value, it is triggered.

If *varName* is not local to the class, the value is put in the first class in the inheritance list in which *varName* is found. If *varName* is not found, a break occurs.

Arguments: *self*            An instance or a class.  
               *varName*        A class variable name.

*newValue*    A new value.

*propName*    Property name.

Returns: *newValue*

Example: Given that

```
(← ($ Window) New 'window1)
```

then the command

```
(PutClassValue ($ window1) 'LeftButtonItem 2 'number)
```

adds the property number with the value 2 to the class variable **LeftButtonItem** of the class **Window**. The following command performs the same action.

```
(PutClassValue ($ Window) 'LeftButtonItem 2 'number)
```

---

**(GetClassValueOnly** *classRec varName prop*) [Function]

---

Purpose: Returns the value of a class variable or property, without triggering active values.

Behavior: Similar to **GetClassValue**, with the following exceptions:

- If **GetClassValueOnly** finds that the value is an active value, the active value is returned without being triggered.
- If *prop* is non-NIL and is not found, **GetClassValueOnly** returns the value of the variable **NotSetValue**.

Arguments: See **GetClassValue**.

Returns: Value depends on the arguments; see Behavior.

Example: The command

```
(GetClassValueOnly ($ Window) 'abcde)
```

breaks. The command

```
(GetClassValueOnly ($ Window) 'LeftButtonItem 'wxyz)
```

returns the value of **NotSetValue**.

---

**(PutClassValueOnly** *self varName newValue propName*) [Function]

---

Purpose: Stores the value of a class variable or property, without triggering active values.

Behavior: Similar to **PutClassValue**, except that **PutClassValueOnly** does not trigger an active value, but replaces it with *newValue*.

Arguments: See **PutClassValue**.

Returns: Used for side effect only.

---

**(GetCVHere** *classRec varName propName*) [Function]

---

Purpose: Returns the value of a class variable in a particular class without looking for inherited values.

**Behavior:** Returns the value of the class variable *varName*, or the *propName* property if *propName* is non-NIL.

If the value is an active value, it is returned without being triggered.

If there is no *varName* (or *propName*), this returns the value of the variable **NotSetValue**.

**Arguments:** *classRec* Must be a class.  
*varName* A class variable name.  
*propName* Property name.

**Returns:** Value depends on the arguments; see Behavior.

**Example:** The command

```
(GetCVHere ($ NonRectangularWindow) 'LeftButtonItem)
```

returns

```
#,NotSetValue
```

The command

```
(GetCVHere ($ Window) 'LeftButtonItem)
```

returns

```
((Update (QUOTE Update)...))
```

**(PutCVHere self varName value)**

[Function]

**Purpose:** Puts a class variable locally with a value if it is not local.

**Behavior:** Calls **(AddCV self varName value)**.

**Arguments:** *self* An instance or a class.  
*varName* A class variable name.  
*value* Value for the class variable.

**Returns:** *value*

### 5.3.3 Instance Variable Access

An instance variable can be thought of as being local to each instance of a class. The class defines what instance variables and their default values will be in an instance. This section describes the functions that manipulate the default values in the class.

See the *LOOPS Library Modules Manual* for interaction with Masterscope.

| Name                  | Type     | Description                                                                                                       |
|-----------------------|----------|-------------------------------------------------------------------------------------------------------------------|
| <b>GetClassIV</b>     | Function | Gets the default value of an instance variable or associated property as defined in a class or one of its supers. |
| <b>GetClassIVHere</b> | Function | Gets the default value of an instance variable or associated property as defined in a class.                      |



|                   |          |                                                                |
|-------------------|----------|----------------------------------------------------------------|
| <b>PutClassIV</b> | Function | Changes the default value for an instance variable in a class. |
|-------------------|----------|----------------------------------------------------------------|

---

|                                              |            |
|----------------------------------------------|------------|
| <b>(GetClassIV <i>self varName prop</i>)</b> | [Function] |
|----------------------------------------------|------------|

---

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                     |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| Purpose:   | Gets the default value of an instance variable or associated property as defined in a class or one of its supers.                                                                                                                                                                                                                                                                                                                    |                                                     |
| Behavior:  | If <i>self</i> is not bound to a class, an error occurs.<br><br>Searches through the supers of <i>self</i> to find <i>varName</i> or <i>prop</i> . <ul style="list-style-type: none"><li>• If <i>prop</i> is NIL, this returns the default value for <i>varName</i>.</li><li>• If <i>prop</i> is non-NIL, this returns its default value.</li></ul> If the default value is an active value, it is returned without being triggered. |                                                     |
| Arguments: | <i>self</i>                                                                                                                                                                                                                                                                                                                                                                                                                          | Must be bound to a class.                           |
|            | <i>varName</i>                                                                                                                                                                                                                                                                                                                                                                                                                       | The name of an instance variable.                   |
|            | <i>prop</i>                                                                                                                                                                                                                                                                                                                                                                                                                          | Name of a property associated with <i>varName</i> . |
| Returns:   | Value depends on the arguments; see Behavior.                                                                                                                                                                                                                                                                                                                                                                                        |                                                     |
| Example:   | The commands<br><br><pre>(GetClassIV (\$ Window) 'window) (GetClassIV (\$ NonRectangularWindow) 'window)</pre><br>both return<br><br><pre>#, (\$AV LispWindowAV ...)</pre>                                                                                                                                                                                                                                                           |                                                     |

---

|                                                  |            |
|--------------------------------------------------|------------|
| <b>(GetClassIVHere <i>self varName prop</i>)</b> | [Function] |
|--------------------------------------------------|------------|

---

|            |                                                                                                                                                                                                                           |                                                     |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|
| Purpose:   | Gets the default value of an instance variable or associated property as defined in a class.                                                                                                                              |                                                     |
| Behavior:  | Similar to <b>GetClassIV</b> . This does not search the super classes of <i>self</i> for <i>varName</i> . If <i>varName</i> or <i>prop</i> is not local to <i>self</i> , this returns the value of <b>NotSetValue</b> .   |                                                     |
| Arguments: | <i>self</i>                                                                                                                                                                                                               | Pointer to a class.                                 |
|            | <i>varName</i>                                                                                                                                                                                                            | Name of an instance variable.                       |
|            | <i>prop</i>                                                                                                                                                                                                               | Name of a property associated with <i>varName</i> . |
| Returns:   | The default value of <i>varName</i> or <i>prop</i> or <b>NotSetValue</b> .                                                                                                                                                |                                                     |
| Example:   | The command<br><br><pre>(GetClassIVHere (\$ Window) 'window)</pre><br>returns<br><br><pre>#, (\$AV LispWindowAV ...)</pre><br>the command<br><br><pre>(GetClassIVHere (\$ NonRectangularWindow) 'window)</pre><br>returns |                                                     |

#,NotSetValue

**(PutClassIV** *self varName newValue propName*)

[Function]

Purpose: Changes the default value for an IV in a class.

Behavior: If *self* is not a class that contains the instance variable *varName*, an error occurs.

- If *propName* is NIL, the default value for the instance variable *varName* is changed to *newValue*.
- If *propName* is non-NIL, the default value for it is changed to *newValue*.

Arguments: *self* Must be a class that contains the instance variable *varName*.

*varName* An instance variable name.

*newValue* The new default value.

*propName* Property name.

Returns: *newValue* (used for side effect only).

Example: After the commands

```
68←(DefineClass 'Datum)
#,($C Datum)
```

```
69←(← ($ Datum) AddIV 'id# NIL)
id#
```

the following command changes the default value of the instance variable **id#** to '(7) for all new instances of the class **Datum**:

```
70←(PutClassIV ($ Datum) 'id# '(7))
(7)
```

[This page intentionally left blank]

|                        |                                                                                                                                                                                                                              |
|------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| abstract class         | A class which cannot be instantiated, for example, <b>ActiveValue</b> .                                                                                                                                                      |
| active value           | The mechanism that carries out access-oriented programming for variables in LOOPS. Active values send messages as a side effect of having an object's variable referenced.                                                   |
| activeValue            | The previous implementation of the active value concept.                                                                                                                                                                     |
| <b>ActiveValue</b>     | An abstract class that defines the general protocol followed by all active value objects.                                                                                                                                    |
| annotatedValue         | A special Interlisp-D data type that wraps each <b>ActiveValue</b> instance.                                                                                                                                                 |
| <b>AnnotatedValue</b>  | An abstract class that allows an annotatedValue to be treated as an object.                                                                                                                                                  |
| browser                | A window that allows you to examine and change items in a data structure.                                                                                                                                                    |
| class                  | A description of one or more similar objects; that is, objects containing the same types of data fields and responding to the same messages.                                                                                 |
| class inheritance      | The means by which a class inherits variables, values, and methods from its super class(es).                                                                                                                                 |
| class lattice          | A network showing the inheritance relationship among classes.                                                                                                                                                                |
| class variable (CV)    | A variable that contains information shared by all instances of the class. A class variable is typically used for information about a class taken as a whole.                                                                |
| inheritance            | The means by which you can organize information in objects, create objects that are similar to other objects, and update objects in a simplified way.                                                                        |
| Inspector              | A Lisp display program that has been modified to allow you to view classes, objects, and active values.                                                                                                                      |
| instance               | An object described by a particular class. Every object within LOOPS is an instance of exactly one class.                                                                                                                    |
| instance variable (IV) | A variable that contains information specific to an instance.                                                                                                                                                                |
| instantiate            | To make a new instance of a class.                                                                                                                                                                                           |
| lattice                | An arrangement of nodes in a hierarchical network, which allows for multiple parents of each node.                                                                                                                           |
| Masterscope            | A Lisp Library Module program analysis tool that has been modified to allow analysis of LOOPS files.                                                                                                                         |
| message                | A command sent to an object that activates a method defined in the object's class. The object responds by computing a value that is returned to the sender of the message.                                                   |
| metaclass              | Classes whose instances are classes or abstract classes.                                                                                                                                                                     |
| method                 | What an object applies to the arguments of a message it receives. This is similar to a procedure in procedure-oriented programming, except that here, you determine the message to send and the object receiving the message |

|                         |                                                                                                                                                                                                 |
|-------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                         | determines the method to apply, instead of the calling routine determining which procedure to apply.                                                                                            |
| mixin                   | A class that is used in conjunction with another class to create a subclass. Mixins never have instances, and hence have <b>AbstractClass</b> as their metaclass.                               |
| object                  | A data structure that contains data and a pointer to functionality that can manipulate the data.                                                                                                |
| property list           | A place for storing additional information on classes, their variables, and their methods.                                                                                                      |
| selector                | Part of a message that is sent to an object. The object uses the selector to determine which method is appropriate to apply to the message arguments.                                           |
| self                    | A method argument that represents the receiver of the message.                                                                                                                                  |
| specialization          | The process of creating a subclass from a class, or the result of that process.                                                                                                                 |
| subclass                | A class that is a specialization of another class.                                                                                                                                              |
| super class             | A class from which a given class inherits variables, values, and methods.                                                                                                                       |
| Tofu                    | An acronym for Top of the universe, which is the highest class in the LOOPS hierarchy.                                                                                                          |
| Unique Identifier (UID) | An alphanumeric identifier that LOOPS uses to store and retrieve objects. Objects do not have UIDs unless they are named, are instances of indexed objects, or are instances printed to a file. |
| wrap                    | Objects have fields that can contain data. Some <b>ActiveValue</b> can be added so this data is stored within it. When this occurs, the <b>ActiveValue</b> wraps the data.                      |

Methods are the expressions that evaluate when a message is sent to an instance or a class. Methods are analogous to Interlisp-D functions, except that they are defined by a LOOPS class and invoked by sending a message to an instance of that class.

This chapter presents the basic constructs used to create and implement methods. Also included are important methods and functions relevant to the definition and maintenance of methods.

---

## 6.1 Categories

---

LOOPS methods can be divided into categories. This section contains a brief description of each method category. These categories serve as additional documentation only; they do not imply differences in implementation.

Any symbol can be used as a category. Categories can be used as a tool for the organization of methods. Methods may belong to more than one category.

---

### **Class** [Category]

---

Messages associated with a class method can only be sent to an object of type class. Methods associated with the class **Class** have this category. See Chapter 3, Classes, for more information on classes.

---

### **Object** [Category]

---

The message associated with an object method can only be sent to an object of type object. Methods associated with the class **Object** have this category.

---

### **Internal** [Category]

---

Internal methods are low-level system methods, and should not be specialized by users.

---

### **Public** [Category]

---

Public methods are defined by the user or the system. These methods can be specialized by users.

---

### **Any** [Category]

---

Methods that have not been categorized belong to this category by default.

---

### **Masterscope** [Category]

---

Masterscope is an interactive program analysis tool. Methods that are predefined for Masterscope are local only to Masterscope and can be used only when Masterscope has been invoked. Refer to the *Lisp Library Modules Manual* for more information on Masterscope.

---

(← *self* **AllMethodCategories**) [Method of Class]

---

Purpose/Behavior: Extracts and lists the categories of all methods defined by the class *self*.

Arguments: *self*            Pointer to a class.

Returns: The categories of the methods defined by the class of *self*.

Categories: Class

Example: Line 98 shows the categories of all methods defined in the class *self*.

```
98←(← ($ Class) AllMethodCategories)
(Class Object Masterscope)
```

---

(← *self* **CategorizeMethods** *categorization*) [Method of Class]

---

Purpose: Allows you to change how methods are categorized.

Behavior: Varies according to the arguments.

- If *categorization* is NIL, this opens a display editor window with a form that represents the current categorizations. After you have exited from the editor, these new categorizations are installed.

- If *categorization* is non-NIL, it must be of the following form:

*(category1 (selector1 ... selectorN)) (category2 (selector ...)).*

A categorization specified by **CategorizeMethods** deletes any previous categorization; i.e., if method Print for class Thing was in categories Internal and I/O, after doing

```
(← ($ Thing) CategorizeMethods ' ((Output
(Print)) (Printing (Print))))
```

Print will be only in categories Output and Printing.

Arguments: *self*            Pointer to a class.

*categorization*

A list in the form as described in Behavior, or NIL.

Categories: Class

Example: This example shows how to use **CategorizeMethods** with categorization NIL.

```
1←(← ($ MetaClass) CategorizeMethods)
```

The following display editor window appears:

```
SEdit Package; INTERLISP
((Any (CreateClass DestroyInstance New NewWithValues))
 (Public (CreateClass DestroyInstance New NewWithValues))
 (Internal NIL)
 (MetaClass (CreateClass))
 (Class (DestroyInstance New NewWithValues)))
```

(← *self* **ChangeMethodCategory** *selector newCategory*) [Method of Class]

Purpose: Changes the category of a selected method.

Behavior: Varies according to the arguments.

- If *selector* is NIL, a menu appears showing the selectors for the class of *self*. This is done using the message **PickSelector** to determine the *selector* that is to have its category changed.
- If *selector* is supplied, but not associated with *self*, this message returns NIL.
- If *newCategory* is an atom, adds *selector* to the category. If *newCategory* is a list of atoms, removes *selector* from all its current categories, then adds it to the categories in the list. If *newCategory* is NIL, pops up a menu showing all of the known categories and an additional item, **\*other\***. If **\*other\*** is selected, you are prompted to enter a new category name.

Arguments: *self* Pointer to a class.

*selector* Method selector for class of *self* or NIL.

*newCategory*  
An atom, a list of atoms, or NIL.

Returns: The new category if there was a change made; else NIL.

Categories: Class

Example: The following command changes the categories of the method associated with **Shape1**.

```
2←(← ($ Window) ChangeMethodCategory
 'Shape1 ' (Window Internal))
(Window Internal)
```

## 6.2 STRUCTURE OF METHOD FUNCTIONS

### 6.2 STRUCTURE OF METHOD FUNCTIONS

## 6.2 Structure of Method Functions

This section discusses the structure of a LOOPS method.

(**Method** :FUNCTION-TYPE *type* ((*class selector*) *self args ...*) *body...*) [Definer]

Purpose: Similar to **DefineMethod**, but gives more control over the argument list and body syntax. Allows use of Common Lisp lambda argument lists, and Common Lisp syntax in the body of the method. This is the form you will see when editing methods.



**Behavior:** Defines a method whose argument list is either Interlisp (default) or Common Lisp style. The body of the method may likewise contain either Common Lisp or Interlisp syntax. Common Lisp syntax is distinguished by lexical scoping, etc. (see the *Common Lisp Implementation Notes* for more information).

**Arguments:**

|                 |                                                                                                                                                                                                                                              |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>type</i>     | The :FUNCTION-TYPE type clause is optional and defaults to :IL.<br><br>:IL - The body of the method uses Interlisp syntax, allows CLISP expressions, etc.<br><br>:CL - The body of the method uses Common Lisp syntax (is lexically scoped). |
| <i>class</i>    | The class to which the method will be attached.                                                                                                                                                                                              |
| <i>selector</i> | The new method's selector.                                                                                                                                                                                                                   |
| <i>self</i>     | This argument must be present and first.                                                                                                                                                                                                     |
| <i>args</i>     | If type was given as :CL this argument list may contain Common Lisp keys like &OPTIONAL, &KEY and &REST.                                                                                                                                     |
| <i>body</i>     | The body of the method. If the type was given as :CL it will be treated as the body of a Common Lisp lambda is, e.g. scoping will be lexical.                                                                                                |

**Returns:** The name of the method function.

**Example:**

```
12← (Method :FUNCTION-TYPE :CL ((Window Foo) self bar
&OPTIONAL baz &REST glorp)
 (CL:FORMAT T "Bar ~s baz ~s glorp ~s~%" bar baz
glorp))
13← (← ($ Window) New 'Flarb)
14← (← ($ Flarb) Foo 1 2 3 4)
Bar 1 baz 2 glorp (3 4)
```

---

## 6.3 CREATING, EDITING, AND DESTROYING METHODS

---

### 6.3 CREATING, EDITING, AND DESTROYING METHODS

---

---

## 6.3 Creating, Editing, and Destroying Methods

---

This section describes the methods and functions which are used to create, rename, delete, and edit LOOPS methods.

| Name                          | Type     | Description                                           |
|-------------------------------|----------|-------------------------------------------------------|
| <b>DefineMethod</b>           | Function | Defines a new method on a class.                      |
| <b>DeleteMethod</b>           | Function | Deletes a method from a class.                        |
| <b>EditMethod</b>             | Method   | Invokes the editor on a method of a class.            |
| <b>SubclassResponsibility</b> | Macro    | Appears in the template when you create a new method. |

**(DefineMethod class selector args expr file - )**

[Function]

Purpose: Defines a new method on a class.

Behavior: Varies according to the arguments.

- If *args* is a non-NIL symbol and *expr* is NIL, its function definition is installed as the method for (class selector). This definition must accept an appropriate number of arguments and otherwise work as a LOOPS method. Also, *args* must be a symbol of the form **Name1.Name2** for many of the LOOPS internal routines to handle it properly.
- If *args* is a list of arguments and *expr* is a function, its body will be installed as the definition of **class.selector**.

Arguments: *class* Class in which method is defined.

*selector* Method selector (message).

*args* List of arguments.

*expr* Function definition or NIL.

*file* Place where method is stored.

Example: The following expression shows how to add a method called **Increment** to a class called **Documentation**.

```
(DefineMethod ($ Documentation) 'Increment ' (Number) ' (PLUS number 1)
```

**(DeleteMethod class selector prop )**

[Function]

Purpose: Deletes a method from a class.

Behavior: Varies according to the arguments.

- If *prop* is NIL or T, the method is deleted from the class.
- If *prop* is T, the function definition is also deleted.

Note: You may also delete methods by using the **ClassInheritance Browser**. Position the mouse on the appropriate class, press the middle mouse button, and select **DeleteMethod** from the resulting menu.

Arguments: *class* Class in which method is defined.

*selector* Method selector (message).

*prop* T or NIL; determines whether the function definition is deleted.

Example: The following command deletes the method associated with **'MyOpen** from **LatticeBrowser**.

```
(DeleteMethod ($ LatticeBrowser) 'MyOpen)
```

**(← self EditMethod selector commands okCategories)**

[Method of Class]

Purpose: Invokes the display editor on a method of a class.

Behavior: Varies according to the arguments.

- If *selector* is NIL, a menu of selectors is presented using the message **PickSelector** in *okCategories*. This can be a list or a symbol.

- If *selector* is non-NIL, and if it corresponds to a method that is in not *self*'s class, you are asked whether the method should be created.
- If *selector* cannot be found, the spelling corrector is invoked to find a correct local selector. If it can be corrected, the local method is used, or an inherited method that is made local is used. When the method is finally determined, **EDITF** (refer to the *Lisp Release Notes* and the *Interlisp-D Reference Manual*) is invoked with *commands* passed as the second argument.

Note: You may also edit methods by using the **ClassInheritance Browser**. Position the mouse on the appropriate class, press the middle mouse button, and select **EditMethod** from the resulting menu.

|             |                     |                                           |
|-------------|---------------------|-------------------------------------------|
| Arguments:  | <i>self</i>         | Class name.                               |
|             | <i>selector</i>     | Refers to the method.                     |
|             | <i>commands</i>     | List of <b>EDITF</b> commands.            |
|             | <i>okCategories</i> | Atom or list specifying valid categories. |
| Categories: | Class               |                                           |

(SubclassResponsibility) [Macro]

Purpose/Behavior: Appears in the template when you create a new method. It is used to make sure you specialize a method.

6.4 ESCAPING FROM MESSAGE SYNTAX

6.4 ESCAPING FROM MESSAGE SYNTAX

---

---

## 6.4 Escaping from Message Syntax

---

The methods described in the previous section manipulate methods in a specific order. Sometimes it may be necessary to invoke multiple inherited methods in some other order. The more general functions in this section have been provided to do this.

---

**CAUTION**

These functions do not conform to the conventions of method inheritance and should be used as a last resort and with extreme caution.

---

The following table shows the items in this section.

| Name               | Type     | Description                                                                                                         |
|--------------------|----------|---------------------------------------------------------------------------------------------------------------------|
| <b>DoMethod</b>    | Function | Computes the action which should be a method associated with a class and applies it to an object and arguments.     |
| <b>ApplyMethod</b> | Function | Computes the action which should be a method associated with a class and applies it to an object and argument list. |

|                       |          |                                                                                            |
|-----------------------|----------|--------------------------------------------------------------------------------------------|
| <b>DoFringeMethod</b> | Function | Invokes a method in the class of an object or in each of the super classes for that class. |
|-----------------------|----------|--------------------------------------------------------------------------------------------|

---

**(DoMethod** *object selector class arg1 ... argn*) [Function]

---

|            |                                                                                                                                                                                                            |                                            |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------|
| Purpose:   | Computes the action which should be a method associated with <i>class</i> and applies it to <i>object</i> .                                                                                                |                                            |
| Behavior:  | All of the arguments are evaluated. If <i>class</i> is NIL, <b>DoMethod</b> uses the class of <i>object</i> . If no method from <i>class</i> can be computed from <i>selector</i> , an error is generated. |                                            |
| Arguments: | <i>object</i>                                                                                                                                                                                              | Instance to which action is applied.       |
|            | <i>selector</i>                                                                                                                                                                                            | Evaluates to a method selector.            |
|            | <i>class</i>                                                                                                                                                                                               | NIL or class in which method name resides. |
|            | <i>arg1...argn</i>                                                                                                                                                                                         | The arguments for the method.              |

---

**(ApplyMethod** *object selector argList class*) [Function]

---

|            |                                                                                                                                     |                                      |
|------------|-------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|
| Purpose:   | Same as <b>DoMethod</b> .                                                                                                           |                                      |
| Behavior:  | Applies the selected method to the already evaluated arguments in <i>argList</i> ; otherwise, this is the same as <b>DoMethod</b> . |                                      |
| Arguments: | <i>object</i>                                                                                                                       | Instance to which action is applied. |
|            | <i>selector</i>                                                                                                                     | Evaluates to a method name.          |
|            | <i>arglist</i>                                                                                                                      | The arguments for the method.        |
|            | <i>class</i>                                                                                                                        | Class in which method name resides.  |

Example: This example illustrates the MessageNotUnderstood protocol, the function **ApplyMethod**, and the macro **\_Super**. This is a specialization of the default **MessageNotUnderstood** message that tries to correct the spelling of the selector. (See Chapter 11, Errors and Breaks, for more information on **MessageNotUnderstood**.)

```
(Method ((DwimObject MessageNotUnderstood)
 self selector messageArguments superFlg)
 (LET ((correctSelector (FixSelectorSpelling selector)))
 (COND ((correctSelector (ApplyMethod correctSelector messageArguments))
 (T (_Super))))))
```

Note: *self* is included in the list of **messageArguments**.

---

**(DoFringeMethods** *object selector arg1 ... argn*) [Function]

---

|            |                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                  |
|------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Purpose:   | Invokes method for <i>selector</i> in the class of <i>object</i> or in each of the super classes for that class.                                                                                                                                                                                                                                                                                                                                             |                  |
| Behavior:  | Evaluates all of the arguments. If the method for <i>selector</i> in the class of <i>object</i> is defined in that class (not through inheritance), <b>DoFringeMethods</b> invokes the local method. If there is no local method, <b>DoFringeMethods</b> goes down the class of <i>object</i> , and for each super invokes its method for selector if one exists. If the supers share supers this can result in the same method being called more than once. |                  |
| Arguments: | <i>object</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                | Class instance.  |
|            | <i>selector</i>                                                                                                                                                                                                                                                                                                                                                                                                                                              | Method selector. |

*arg1...argn* Arguments to *selector*.

Returns: NIL

6.5 MOVEMENT BETWEEN CLASSES

6.5 Movement between Classes

This section describes functions and methods that are used in moving methods between classes, as well as stack method macros.

6.5.1 Movement of Methods

The following functions and methods are used to move methods, instance variables, and class variables between classes.

| Name                    | Type     | Description                                                                            |
|-------------------------|----------|----------------------------------------------------------------------------------------|
| <b>RenameMethod</b>     | Function | Renames a function used as a method.                                                   |
| <b>MoveMethod</b>       | Function | Moves a method from one class to another.                                              |
| <b>MoveMethod</b>       | Method   | Moves a method from one class to another.                                              |
| <b>MoveMethodToFile</b> | Function | Moves a method to this file if it has the same name as a function on a specified file. |
| <b>CalledFns</b>        | Function | Finds names of all functions called from a set of classes.                             |

**(RenameMethod** *classOrName oldSelector newSelector*) [Function]

Purpose: Renames a function used as a method in *classOrName*.

Behavior: This changes the selector for a method. If no method is associated with *oldSelector* or *newSelector*, this generates an error. Explicit references to *oldSelector* such as

`(←Super self oldSelector)`

will not be fixed by **RenameMethod**.

Arguments: *classOrName* Class in which function is defined.

*oldSelector* Old name of method; invokes method before this function is called.

*newSelector* New name of method; invokes method after this function is called.

Returns: If successful, returns *newSelector* in the form **ClassName.Selector**.

Example: The following command renames a method named **Foo** to **Fie** in the class **MyClass**.

```
24←(RenameMethod ($ MyClass) 'Foo 'Fie)
```

**(MoveMethod** *oldClassName newClassName selector newSelector files*) [Function]

Purpose: Moves a method from *oldClassName* to *newClassName*. The method is deleted from *oldClassName*.

Behavior: If *newSelector* is a different name than *selector*, **MoveMethod** renames the method. Explicit references to *oldSelector* such as

```
(←Super self oldSelector))
```

will not be fixed by **RenameMethod**.

Note: You may also move methods by using the **ClassInheritance Browser**. Position the mouse on the appropriate class, press the middle mouse button, and select **MoveMethod** from the resulting menu.

Arguments: *oldClassName*  
Source class.

*newClassName*  
Destination class.

*selector* Method selector to be moved.

*newSelector*  
New name; if NIL, the existing *selector* is preserved.

*files* Files in which the change is to occur.

Example: The following command moves the method **Buy** from class **Car** to class **Boat** and renames the method to **Purchase**.

```
25←(MoveMethod ($ Car) ($ Boat) 'Buy 'Purchase)
Boat.Purchase
```

**(← self MoveMethod** *newClassName selector*) [Method of Class]

Purpose: Moves a method from the class associated with *self* to *newClassName*.

Behavior: Same as the function **MoveMethod**, except that you cannot rename *selector*.

Arguments: *self* Pointer to a class from which the method is taken.

*newClassName*  
Destination class; must be a class, not a class name.

*selector* Method selector to be moved.

Returns: **NewsClass.Selector**

**(MoveMethodsToFile** *file*) [Function]

Purpose/Behavior: Moves a method to this file if it has the same name as a function on *file*.

Arguments: *file* Name of a file to which methods are moved.

Returns: Normally T; NIL if a method does not have the same name as a function on *file*.

**(CalledFns classes definedFlg )** [Function]

Purpose: Finds names of all functions called from a set of *classes*.

Behavior: Varies according to the arguments.

- If *definedFlg* is NIL, all the functions associated with *classes* are returned.
- If *definedFlg* is T, the defined functions are returned.
- If *definedFlg* is 1, the undefined functions are returned.

Arguments: *classes*      List of classes to search.  
*definedFlg*    NIL, 1, or T.

Returns: NIL or the list of functions.

Example: The following command finds all functions called from the class **Method**.

`(CalledFns ' (Method) )`

## 6.5.2 Stack Method Macros

---

This section describes macros that access methods on the stack.

**(ClassNameOfMethodOwner)** [Macro]

Purpose: Uses the stack to perform a help check. Returns the name of the class to which the method on top of the stack belongs.

**(SelectorOfMethodBeingCompiled)** [Macro]

Purpose: Uses the stack to perform a help check. Returns the name of the method being compiled.

**(ArgsOfMethodBeingCompiled)** [Macro]

Purpose: Uses the stack to perform a help check. Returns all arguments associated with the method being compiled.

[This page intentionally left blank]



## A

access-oriented programming 1-1,9  
 accessing  
   a class, errors 11-6  
   an instance, errors 11-6  
   class variable 5-16  
   data in classes 5-13  
   instance variable 5-19  
 active value 1-9; 5-5; 8-1  
   adding 8-17  
   breaking and tracing 8-10  
   bypassing 8-22  
   creating 8-23  
   deleting 8-17  
   errors 11-9  
   in class structures 8-27  
   printing 17-11  
   replacing 8-17  
 ActiveValue 8-1  
   methods 8-16  
   shared 8-22  
   specializations 8-2  
**Add** (*Inspector Submenu Option*) 18-4,10  
**Add** (*Method of Class*) 3-12  
**Add** (**AddMethod**) (*Browser Menu Option*) 10-20,32  
**Add Category Menu** (*Browser Menu Option*) 10-10,25  
**Add file to browser** (*Browser Submenu Option*) 10-26  
**Add/Delete** (*Inspector Menu Option*) 18-4,10  
**AddActiveValue** (*Method of ActiveValue*) 8-17  
**AddCIV** (*Function*) 3-14  
**AddCV** (*Browser Submenu Option*) 10-20  
**AddCV** (*Function*) 3-13  
**AddCV** (*Method of Class*) 3-13  
**AddIV** (*Browser Submenu Option*) 10-20  
**AddIV** (*Function*) 2-10  
**AddIV** (*Method of Class*) 3-14  
**AddIV** (*Method of Object*) 2-11  
**AddMethod** (*Browser Submenu Option*) 10-21  
 adding an active value 8-17  
**AddRoot** (*Browser Menu Option*) 10-10,25  
**AddRoot** (*Method of LatticeBrowser*) 10-36  
**AddSubs** (*Browser Menu Option*) 10-31  
**AddSubs!** (*Browser Submenu Option*) 10-31  
**AddSuper** (*Browser Submenu Option*) 10-21  
**AfterMove** (*Method of Window*) 19-3  
**AfterReshape** (*Method of Window*) 19-3  
**all** (*Browser Submenu Option*) 10-26  
**All** (*Inspector Menu Option*) 18-8  
**AllInstances** (*Method of Class*) 3-24  
**AllInstances!** (*Method of Class*) 3-25  
**AllMethodCategories** (*Method of Class*) 6-2  
**AllSubClasses** (*Function*) 3-28  
**AllValues** (*Inspector Menu Option*) 18-3,10  
**AnalyzeFile** (*Browser Submenu Option*) 10-28  
 annotated value 8-24  
   errors 11-9  
   explicit control over 8-25  
   restoring 8-26  
   saving 8-26  
**AnnotatedValue** (*Class*) 8-24  
**AnnotatedValue?** (*Macro*) 9-2

**Any** (*Category*) 6-1  
**AppendSuperValue** (*Class*) 8-11  
**ApplyMethod** (*Function*) 6-7  
**ArgsOfMethodBeingCompiled** (*Macro*) 6-10  
**associatedFiles** (*Browser Submenu Option*) 10-26  
**AttachLispWindow** (*Method of Window*) 19-22  
**AVPrintSource** (*Method of ActiveValue*) 8-26; 17-11  
**AVPrintSource** (*Specialization of ActiveValue*) 8-16

## B

background menu 10-3  
**Blink** (*Method of Window*) 19-3  
**Box/UnBoxNode** (*Browser Menu Option*) 10-17  
**BoxNode** (*Browser Menu Option*) 10-32  
**BoxNode** (*Method of LatticeBrowser*) 10-37  
**Break on Access** (*Inspector Submenu Option*) 18-6  
**Break on Put** (*Inspector Submenu Option*) 18-6  
**BreakFunction** (*Browser Submenu Option*) 10-29  
 breaking 12-1  
**BreakIt** (*Function*) 12-4  
**BreakIt** (*Inspector Menu Option*) 18-6  
**BreakIt** (*Method of Object*) 12-3  
**BreakMethod** (*Browser Menu Option*) 10-20  
**BreakMethod** (*Method of Class*) 12-1  
**BreakOnPut** (*Class*) 8-10  
**BreakOnPutOrGet** (*Class*) 8-10  
**BrokenVariables** (*Global Variable*) 12-6  
**Browse** (*Function*) 10-7  
**Browse** (*Inspector Menu Option*) 18-8  
**Browse** (*Method of LatticeBrowser*) 10-6,37  
**Browse Class** (*Submenu Option*) 10-3,4  
**Browse File** (*Submenu Option*) 10-3,5  
**Browse Supers** (*Submenu Option*) 10-4  
**BrowseFile** (*Method of FileBrowser*) 10-6  
 browser 10-1  
**BrowserObjects** (*Method of LatticeBrowser*) 10-38  
**BrowseSupers** (*Inspector Submenu Option*) 18-8  
**Bury** (*Method of Window*) 20-4  
**ButtonEventFn** (*Method of Window*) 19-15

## C

cache 15-2,3  
**CalledFns** (*Function*) 6-9  
**CallsFunction** (*Browser Submenu Option*) 10-28  
 categories of a method 6-1  
**CategorizeMethods** (*Browser Submenu Option*) 10-19  
**CategorizeMethods** (*Method of Class*) 6-2  
**Change display mode** (*Browser Menu Option*) 10-25  
**ChangeClass** (*Method of Object*) 2-16  
**ChangeFontSize** (*Browser Submenu Option*) 10-8  
**ChangeFontSize** (*Method of LatticeBrowser*) 10-38  
**ChangeFormat** (*Method of LatticeBrowser*) 10-38  
**ChangeMaxLabelSize** (*Method of LatticeBrowser*) 10-39  
**ChangeMethodCategory** (*Browser Submenu Option*) 10-19  
**ChangeMethodCategory** (*Method of Class*) 6-3  
 Changetran 5-13  
**Check** (*Browser Submenu Option*) 10-27  
**CheckFile** (*Browser Submenu Option*) 10-28  
 class 1-3,6; 3-1  
   contents 3-1

- copying 3-23
- creating 3-1
- destroying 3-5
- editing 3-10; 13-1
- enumerating instances 3-24
- manipulating 2-16; 3-16
- modifying 3-11
- printing 17-4
- querying the structure 3-17
- reading data in 5-13
- renaming 3-16
- storing data in 5-13
- Class** (Category) 6-1
- Class** (Inspector Menu Option) 18-3
- Class** (Macro) 2-17
- Class** (Method of Object) 2-17
- class browser 10-2
  - automatic update 10-52
  - menu interface 10-8
- class inspector 18-7
- class variable 1-4; 5-5
  - accessing 5-16
  - for class LatticeBrowser 10-34
- Class?** (Macro) 9-2
- ClassDoc** (Browser Submenu Option) 10-14
- CLASSES** (File Package Command) 14-4
- ClassIV inspector 18-9
- ClassName** (Function) 2-18; 3-17
- ClassName** (Method of Object) 2-18
- ClassNameOfMethodOwner** (Macro) 6-10
- CLEANUP** file (Browser Menu Option) 10-30
- CleanUp File** (Submenu Option) 10-5
- Clear** (Method of Window) 19-4
- ClearAllCaches** (Function) 15-3; 20-3
- ClearAllCaches** (Variable) 20-3
- ClearLabelCache** (Method of LatticeBrowser) 10-40
- ClearMenuCache** (Method of Window) 19-15
- ClearPromptWindow** (Method of Window) 19-10
- Close** (Method of Window) 19-4
- ClosePromptWindow** (Method of Window) 19-10
- compact accessing forms 5-10
- ConformToClass** (Method of Object) 2-12
- Copy** (Method of Class) 3-23
- Copy** (CopyMethodTo) (Browser Menu Option) 10-23,32
- CopyActiveValue** (Method of ActiveValue) 8-22
- CopyActiveValue** (Specialization of ActiveValue) 8-16
- CopyCV** (Method of Class) 3-23
- CopyCVTo** (Browser Submenu Option) 10-23
- CopyDeep** (Method of Object) 2-19
- copying
  - class 3-23
  - instance 2-19
- CopyIV** (Method of Class) 3-24
- CopyIVTo** (Browser Submenu Option) 10-23
- CopyMethodTo** (Browser Submenu Option) 10-23
- CopyShallow** (Method of Object) 2-20
- create annotatedValue** (Macro) 8-25
- CreateClass** (Method of Metaclass) 3-3
- CreateWindow** (Method of NonRectangular Window) 19-19
- CreateWindow** (Method of Window) 19-22
- creating
  - active value 8-23
  - class 3-1
  - instance 2-4

- method 6-4
- record 18-17
- CursorInside?** (Method of Window) 19-4
- CV, see class variable
- CVDoc** (Browser Submenu Option) 10-14
- CVMissing** (Method of Class) 11-2
- CVValueMissing** (Method of Class) 11-3
- D**
- data type predicate 9-1
- DefaultActiveValueClassName** (Variable) 17-12
- DEFCLASS** (NLambda NoSpread Function) 14-4
- DEFCLASSES** (NLambda NoSpread Function) 14-4
- DefineClass** (Function) 3-2
- DefineMethod** (Function) 6-5
- defining a metaclass 4-5
- DEFINST** (NLambda NoSpread Function) 14-6
- DEFINSTANCES** (NLambda NoSpread Function) 14-6
- Delete** (Inspector Submenu Option) 18-4,10
- Delete** (Method of Class) 3-12
- Delete** (DeleteMethod) (Browser Menu Option) 10-21,32
- DeleteActiveValue** (Method of ActiveValue) 8-18
- DeleteCIV** (Function) 3-15
- DeleteClass** (Browser Submenu Option) 10-22
- DeleteClassProp** (Function) 3-13
- DeleteCV** (Browser Submenu Option) 10-21
- DeleteCV** (Function) 3-14
- DeleteFromBrowser** (Browser Menu Option) 10-16,31
- DeleteFromBrowser** (Browser Submenu Option) 10-16
- DeleteFromBrowser** (Method of LatticeBrowser) 10-40
- DeleteIV** (Browser Submenu Option) 10-21
- DeleteIV** (Function) 2-11
- DeleteIV** (Method of Object) 2-12
- DeleteMethod** (Browser Submenu Option) 10-22
- DeleteMethod** (Function) 6-5
- DeleteSubtreeFromBrowser** (Browser Submenu Option) 10-16
- DeleteSubtreeFromBrowser** (Method of LatticeBrowser) 10-40
- deleting an active value 8-17
- DelFromFile** (Method of Object) 14-9
- Destroy** (Method of Class) 3-5
- Destroy** (Method of Object) 2-15
- Destroy** (Method of Window) 19-4
- Destroy!** (Method of Class) 3-6
- Destroy!** (Method of Object) 2-15
- DestroyClass** (Method of Class) 3-6
- destroying
  - class 3-5
  - instance 2-15
  - method 6-4
- DestroyInstance** (Method of Class) 2-15
- DetachLispWindow** (Method of Window) 19-22
- Doc** (ClassDoc) (Browser Menu Option) 10-13,30
- DoFringeMethods** (Function) 6-7
- DoMethod** (Function) 6-7
- DontSave** (Instance Variable Property Name) 14-10
- dynamic mixin 3-4

**E**

- Edit** (Browser Submenu Option) 10-27

**Edit** (*Inspector Menu Option*) 18-4,8  
**Edit** (*Method of Class*) 3-10; 13-1  
**Edit** (*Method of Object*) 13-5  
**Edit (EditClass)** (*Browser Menu Option*) 10-24,32  
**Edit FileComs** (*Browser Menu Option*) 10-28  
**Edit Filecoms** (*Submenu Option*) 10-5  
**Edit Functions** (*Browser Submenu Option*) 10-29  
**Edit!** (*Method of Class*) 13-3  
**EditCategory** (*Browser Submenu Option*) 10-19  
**EditClass** (*Browser Submenu Option*) 10-24  
**EditClass!** (*Browser Submenu Option*) 10-24  
**EditComs** (*Browser Submenu Option*) 10-29  
**EditFns** (*Browser Submenu Option*) 10-29  
**EditIcon** (*Method of NonRectangular Window*) 19-19  
 editing 13-1  
     a class 13-1  
     class 3-10  
     description of window 13-2  
     method 6-4  
**EditInstances** (*Browser Submenu Option*) 10-29  
**EditMacros** (*Browser Submenu Option*) 10-29  
**EditMask** (*Method of NonRectangular Window*) 19-20  
**EditMethod** (*Browser Submenu Option*) 10-18  
**EditMethod** (*Method of Class*) 6-5  
**EditMethod!** (*Browser Submenu Option*) 10-18  
**EditMethodObject** (*Browser Submenu Option*) 10-18  
**EditRecords** (*Browser Submenu Option*) 10-29  
**EditVars** (*Browser Submenu Option*) 10-29  
 enumerating instances of a class 3-24  
 error handling 11-1  
 error message 11-5  
**ErrorOnNameConflict** (*Variable*) 2-4; 11-2  
 escaping from message syntax 6-6  
**ExplicitFnActiveValue** (*Class*) 8-8

## F

**fetch annotatedValue of** (*Macro*) 8-25  
**FetchMethod** (*Method of Class*) 7-5  
 file  
     storing 14-10  
 file browser 10-2  
     menu interface 10-24  
 file manager 1-11; 14-1  
     commands 14-3  
**FileBrowse** (*Function*) 10-7  
**FileIn** (*Method of Class*) 14-6  
**FileOut** (*Method of Class*) 17-4  
**FileOut** (*Method of Object*) 14-11; 17-8  
**FILES?** (*Function*) 14-7  
**FirstFetchAV** (*Class*) 8-12  
**FlashNode** (*Method of LatticeBrowser*) 10-41  
**FlipNode** (*Method of LatticeBrowser*) 10-41  
**Fringe** (*Method of Class*) 3-27  
 function calling 3-2

## G

garbage collection 15-1  
**Get** (*Method of Object*) 5-6  
**GetClass** (*Function*) 5-14  
**GetClassHere** (*Function*) 5-16  
**GetClassIV** (*Function*) 5-20  
**GetClassIVHere** (*Function*) 5-20  
**GetClassOnly** (*Function*) 5-15  
**GetClassProp** (*Method of Class*) 3-18

**GetClassValue** (*Function*) 5-8,16  
**GetClassValueOnly** (*Function*) 5-9,18; 8-22  
**GetCVHere** (*Function*) 5-18  
**GetDisplayLabel** (*Method of LatticeBrowser*) 10-41  
**GetIt** (*Function*) 5-1  
**GetItHere** (*Function*) 5-3  
**GetItOnly** (*Function*) 5-2  
**GetIVHere** (*Function*) 5-10  
**GetLabel** (*Method of LatticeBrowser*) 10-41  
**GetLispClass** (*Function*) 4-3  
**GetObjectNames** (*Function*) 2-4  
**GetObjFromUID** (*Function*) 17-15  
**GetPromptWindow** (*Method of Window*) 19-10  
**GetProp** (*Method of Window*) 19-5  
**GetSubs** (*Method of InstanceBrowser*) 10-50  
**GetSubs** (*Method of LatticeBrowser*) 10-42  
**GettingWrappedValue** (*Message*) 1-9  
**GetValue** (*Function*) 4-3; 5-6  
**GetValue** (*Macro*) 15-2  
**GetValueOnly** (*Function*) 5-7; 8-22  
**GetWrappedValue** (*Method of ActiveValue*) 8-20  
**GetWrappedValue** (*Method of LispWindowAV*) 19-23  
**GetWrappedValue** (*Specialization of ActiveValue*) 8-16  
**GetWrappedValueOnly** (*Method of ActiveValue*) 8-20  
 global cache 15-2,3  
 Grapher 1-11; 10-1,33  
**GraphFits** (*Method of LatticeBrowser*) 10-42

## H

**Hardcopy** (*Method of Window*) 19-5  
**Hardcopy file** (*Browser Submenu Option*) 10-30  
**HardcopyToFile** (*Method of Window*) 19-5  
**HardcopyToPrinter** (*Method of Window*) 19-5  
**HasAttribute** (*Method of Class*) 3-18  
**HasAttribute!** (*Method of Class*) 3-19  
**HasCV** (*Method of Class*) 3-19  
**HasCV** (*Method of Object*) 2-21  
**HasItem** (*Method of Class*) 3-20  
**HasIV** (*Method of Class*) 3-21  
**HasIV** (*Method of Object*) 2-21  
**HasIV!** (*Method of Class*) 3-21  
**HasLispWindow** (*Method of Window*) 19-23  
**HasObject** (*Method of LatticeBrowser*) 10-42  
**HasUID?** (*Function*) 17-14  
**HELPCHECK** (*Function*) 11-1  
**HighlightNode** (*Method of LatticeBrowser*) 10-42

## I

**IconTitle** (*Method of LatticeBrowser*) 10-43  
**IconWindow** (*Class*) 19-20  
**ImplementsMethod** (*Browser Submenu Option*) 10-28  
**in-supers-of** (*Iterative Statement Operator*) 9-3  
**IndexedObject** (*Class*) 3-25  
**IndirectVariable** (*Class*) 8-3  
 inheritance 1-6; 3-7,27  
**InheritedValue** (*Method of InheritingAV*) 8-14  
**InheritingAV** (*Class*) 8-14  
**InPlace** (*Browser Submenu Option*) 10-8  
**Inspect** (*Inspector Menu Option*) 18-7,10  
**Inspect** (*Method of Object*) 2-22; 18-2  
**InspectClass** (*Browser Submenu Option*) 10-24  
**InspectFetch** (*Method of Object*) 18-12  
 inspector 1-10; 18-1

customizing 18-15  
**InspectPropCommand** (*Method of Object*) 18-13  
**InspectProperties** (*Method of Object*) 18-13  
**InspectStore** (*Method of Object*) 18-13  
**InspectTitle** (*Method of Object*) 18-14  
**InspectValueCommand** (*Method of Object*) 18-14  
**InstallEditSource** (*Method of Class*) 13-3  
**InstallEditSource** (*Method of Object*) 13-6  
 instance 1-4  
   accessing data in 5-4  
   copying 2-19  
   creating 2-4  
   data storage at creation time 2-8  
   destroying 2-15  
   editing 13-5  
   naming 2-1  
   querying structure 2-21  
 instance browser 10-3,50  
 instance inspector 18-2  
 instance variable 1-4; 5-5  
   access 15-1  
   accessing 5-19  
   changing number of 2-10  
   delimiters 2-11  
   for class InstanceBrowser 10-50  
   for class LatticeBrowser 10-33  
**Instance?** (*Macro*) 9-2  
**INSTANCES** (*File Package Command*) 14-5  
**InstOf** (*Method of Object*) 2-18  
**InstOf!** (*Method of Object*) 2-19  
**Internal** (*Category*) 6-1  
**Invert** (*Method of NonRectangular Window*) 19-20  
**Invert** (*Method of Window*) 19-5  
**ItemMenu** (*Method of Window*) 19-15  
 iterative operator 9-3  
 IV, see instance variable  
**IVDoc** (*Browser Submenu Option*) 10-14  
**IVMissing** (*Method of Object*) 2-12; 11-3  
**IVs** (*Inspector Menu Option*) 18-4,10  
**IVValueMissing** (*Method of Object*) 2-8; 11-4  
  
**L**  
 lattice 1-1  
 lattice browser 10-2  
**Lattice/Tree** (*Browser Submenu Option*) 10-9  
 left column menu  
   class inspector 18-8  
   ClassIV inspector 18-10  
   instance inspector 18-4  
 left menu  
   class, meta, and supers browsers 10-11  
   file browser 10-30  
   instance browser 10-51  
**LeftSelection** (*Method of LatticeBrowser*) 10-43  
**LeftSelection** (*Method of Window*) 19-16  
**LeftShiftSelect** (*Method of LatticeBrowser*) 10-43  
 Lisp window 19-22  
**LispClassTable** (*Global Variable*) 4-4  
**LispUserFilesForLoops** (*Variable*) 20-2  
**LispWindowAV** (*Class*) 8-10  
**ListAttribute** (*Method of Class*) 3-21  
**ListAttribute** (*Method of Object*) 2-22  
**ListAttribute!** (*Method of Class*) 3-22  
**ListAttribute!** (*Method of Object*) 2-23  
**LOAD** (*Function*) 14-2  
**Load PROP file** (*Browser Submenu Option*) 10-30  
**LOADFNS** (*Function*) 14-3  
 loading a file 14-2

**LoadLoopsForms** (*Variable*) 20-2  
**Local** (*Inspector Menu Option*) 18-8  
 local cache 15-2,3  
**LocalStateActiveValue** (*Class*) 8-6  
**LocalValues** (*Inspector Menu Option*) 18-4,10  
 LOOPS icon 10-4  
**Loops Icon** (*Menu Option*) 10-3  
**LoopsDate** (*Variable*) 20-2  
**LoopsDebugFlg** (*Variable*) 11-2  
**LOOPSDIRECTORY** (*Variable*) 20-2  
**LOOPFILES** (*Variable*) 20-2  
**LoopsHelp** (*NoSpread Function*) 11-2  
**LoopsIcon** (*Class*) 19-21  
**LOOPSLIBRARYDIRECTORY** (*Variable*) 20-2  
**LoopsPatchFiles** (*Variable*) 20-2  
**LOOPUSERSDIRECTORY** (*Variable*) 20-2  
**LOOPUSERSRULESDIRECTORY** (*Variable*) 20-2  
**LoopsVersion** (*Variable*) 20-1

## M

**MakeEditSource** (*Method of Class*) 13-4  
**MakeEditSource** (*Method of Object*) 13-5  
**MAKEFILE** (*Function*) 14-11  
**MakeFileSource** (*Method of Object*) 14-11  
**MakeFullEditSource** (*Method of Class*) 13-4  
**MakeFunctionMenu** (*Browser Submenu Option*) 10-29  
 manipulating  
   a file 14-1  
   a class 2-16; 3-16  
**MapObjectUID** (*Function*) 17-15  
 Masterscope 1-10  
**Masterscope** (*Category*) 6-2  
**MaxLatticeHeight** (*Variable*) 10-48  
**MaxLatticeWidth** (*Variable*) 10-48  
 menu 19-14  
   caching 19-18  
   item structure 19-17  
 message 1-3  
 message sending 3-2  
 message sending form 7-1; 18-16  
 message syntax, escaping from 6-6  
**MessageNotUnderstood** (*Method of AnnotatedValue*) 8-26  
**MessageNotUnderstood** (*Method of Object*) 11-5  
**MessageNotUnderstood** (*Method of Tofu*) 4-7  
 metaclass 1-6; 5-13  
   AbstractClass 4-2  
   Class 4-1  
   defining 4-5  
   DestroyedClass 4-2  
   MetaClass 4-2  
 metaclass browser 10-2  
   menu interface 10-8  
**METH** (*NLambda NoSpread Function*) 14-5  
 method 1-3; 6-1  
   creating 6-4  
   destroying 6-4  
   editing 6-4  
   for class InstanceBrowser 10-50  
   for window 19-2  
   printing 17-12  
**method** (*Definer*)  
   structure 6-3  
 method lookup 15-3  
**MethodDoc** (*Browser Submenu Option*) 10-14  
**MethodDoc** (*Method of Class*) 17-13  
**MethodMenu** (*Browser Submenu Option*) 10-19

**MethodNotFound** (Method of Tofu) 4-7  
**METHODS** (File Package Command) 14-5  
**Methods (EditMethod)** (Browser Menu Option) 10-18,32  
**MethodSummary** (Browser Submenu Option) 10-13  
**MethodSummary** (Method of Class) 17-13  
 middle menu  
   class, meta, and supers browsers 10-17  
   file browser 10-31  
   instance browser 10-52  
**MiddleSelection** (Method of Window) 19-16  
**MiddleShiftSelect** (Method of LatticeBrowser) 10-44  
 modifying a class 3-11  
**MousePackage** (Method) 19-6  
**MouseReadtable** (Method) 19-6  
**Move** (Method of Window) 19-5  
**Move (MoveMethodTo)** (Browser Menu Option) 10-22,32  
**MoveClassVariable** (Function) 2-14  
**MoveCVTo** (Browser Submenu Option) 10-22  
**MoveIVTo** (Browser Submenu Option) 10-22  
**MoveMethod** (Function) 6-8  
**MoveMethod** (Method of Class) 6-9  
**MoveMethodsToFile** (Function) 6-9  
**MoveMethodTo** (Browser Submenu Option) 10-22  
**MoveSuperTo** (Browser Submenu Option) 10-22  
**MoveToFile** (Browser Submenu Option) 10-22  
**MoveToFile** (Method of Class) 14-9  
**MoveToFile!** (Browser Submenu Option) 10-22  
**MoveToFile!** (Method of Class) 14-9  
**MoveVariable** (Function) 2-14  
 moving a variable 2-13  
 multiple references of objects 15-1

## N

naming an instance 2-1  
 naming an object, errors 11-8  
**NestedNotSetValue** (Class) 8-16  
**New** (Method of Class) 2-5  
**New** (Method of Metaclass) 3-3; 4-5  
**NewClass** (Method of Class) 3-3  
**NewInstance** (Browser Submenu Option) 10-21,32  
**NewInstance** (Method of Object) 2-6  
**NewItem** (Method of LatticeBrowser) 10-44  
**NewPath** (Method of InstanceBrowser) 10-51  
**NewWithValues** (Method of Class) 2-7  
**NiceMenu** (Function) 19-13  
**NodeRegion** (Method of LatticeBrowser) 10-45  
**NonRectangularWindow** (Class) 19-19  
**NotSetValue** (Class) 8-15  
**NotSetValue** (Macro) 2-9  
**NoUpdatePermittedAV** (Class) 8-9  
**NoValueFound** (Macro) 2-24  
**NoValueFound** (Variable) 2-24

## O

object 1-2  
   multiple references 15-1  
   printing 17-8  
   saving on a file 14-6  
   storing data in 1-4  
**Object** (Category) 6-1  
 object-oriented programming 1-1,3  
**Object?** (Macro) 9-1  
**ObjectAlwaysPPFlag** (Variable) 17-3

**ObjectDontPPFlag** (Variable) 17-3  
**ObjectFromLabel** (Method of LatticeBrowser) 10-45  
**ObjectModified** (Method of Object) 14-8  
**OldInstance** (Method of Object) 14-10  
**OnFile** (Method of Class) 14-8  
**Open** (Method of Window) 19-6  
 opening a browser 10-3  
**OptionalLispuserFiles** (Variable) 9-2  
**OverridesMethod** (Browser Submenu Option) 10-28

## P

**Paint** (Method of Window) 19-6  
**PositionNode** (Method of LatticeBrowser) 10-45  
**PP** (Browser Submenu Option) 10-12  
**PP** (Method of Class) 17-5  
**PP** (Method of Object) 17-9  
**PP!** (Browser Submenu Option) 10-12  
**PP!** (Method of Class) 17-6  
**PP!** (Method of Object) 17-9  
**PPDefault** (Variable) 17-12  
**PPMethod** (Browser Submenu Option) 10-13  
**PPMethod** (Method of Class) 17-12  
**PPV!** (Browser Submenu Option) 10-12  
**PPV!** (Method of Class) 17-7  
**PPV!** (Method of Object) 17-10  
**PrettyPrintClass** (Function) 14-11  
**PrettyPrintInstance** (Function) 14-11  
**PrintCategories** (Browser Submenu Option) 10-12  
 printing 17-1  
   variables affecting 17-2  
**PrintOn** (Method of IndexedObject) 3-25  
**PrintOn** (Method of Object) 17-8  
**PrintSummary** (Browser Menu Option) 10-12,30  
**PrintSummary** (Browser Submenu Option) 10-13  
 procedure-oriented programming 1-1,2,3  
 programming paradigms 1-1  
 prompt window 19-9  
**PromptEval** (Function) 19-11  
**PromptForList** (Method of Window) 19-11  
**PromptForString** (Method of Window) 19-12  
**PromptForWord** (Method of Window) 19-12  
**PromptPrint** (Method of Window) 19-13  
**PromptRead** (Function) 19-13  
**Properties** (Inspector Menu Option) 18-5,11  
**Prototype** (Method of Class) 3-26  
 pseudoclass 8-24  
 pseudoclasses 4-2  
 pseudoinstances 4-2; 8-24  
**Public** (Category) 6-1  
**Put** (Method of Object) 5-7  
**PutClass** (Function) 5-14  
**PutClassIV** (Function) 5-21  
**PutClassOnly** (Function) 5-15  
**PutClassValue** (Function) 5-8,17  
**PutClassValueOnly** (Function) 5-9,18; 8-22  
**PutCVHere** (Function) 5-19  
**PutIt** (Function) 5-3  
**PutItOnly** (Function) 5-4  
**PutSavedValue** (Function) 19-21  
**PuttingWrappedValue** (Message) 1-9  
**PutValue** (Function) 4-3; 5-6  
**PutValue** (Inspector Menu Option) 18-5  
**PutValue** (Macro) 15-2  
**PutValueOnly** (Function) 5-7; 8-22  
**PutValueOnly** (Inspector Submenu Option) 18-5  
**PutWrappedValue** (Method of ActiveValue) 8-21

**PutWrappedValue** (Method of *LispWindowAV*) 19-23  
**PutWrappedValue** (Specialization of *ActiveValue*) 8-16  
**PutWrappedValueOnly** (Method of *ActiveValue*) 8-21

## Q

querying  
     structure of a class 3-17  
     structure of an instance 2-21

## R

reading 17-1  
**Recompute** (Browser Menu Option) 10-8,25  
**Recompute** (Browser Submenu Option) 10-8  
**Recompute** (Method of *LatticeBrowser*) 10-46  
**RecomputeInPlace** (Method of *LatticeBrowser*) 10-46  
**RecomputeLabels** (Browser Submenu Option) 10-8  
**RecomputeLabels** (Method of *LatticeBrowser*) 10-46  
 record  
     creating 18-17  
**Refetch** (Inspector Menu Option) 18-4,8,10  
**RemoveFromBadList** (Browser Submenu Option) 10-10  
**RemoveHighlights** (Method of *LatticeBrowser*) 10-46  
**RemoveShading** (Method of *LatticeBrowser*) 10-47  
**Rename** (Method of Class) 3-16  
**Rename** (Method of Object) 2-3  
**Rename** (**RenameMethod**) (Browser Menu Option) 10-23,32  
**RenameClass** (Browser Submenu Option) 10-23  
**RenameCV** (Browser Submenu Option) 10-23  
**RenameIV** (Browser Submenu Option) 10-23  
**RenameMethod** (Browser Submenu Option) 10-23  
**RenameMethod** (Function) 6-8  
**RenameVariable** (Function) 2-14  
**replace annotatedValue of** (Macro) 8-25  
**ReplaceActiveValue** (Method of *ActiveValue*) 8-19  
**ReplaceMeAV** (Class) 8-15  
**ReplaceSupers** (Method of Class) 3-15  
 replacing an active value 5-17  
**RetireMethod** (Browser Submenu Option) 10-23  
 right column menu  
     class inspector 18-8  
     ClassIV inspector 18-10  
     instance inspector 18-6  
**RightSelection** (Method of Window) 19-16  
 rule-oriented programming 1-1

## S

**Save Value** (Inspector Menu Option) 18-4,7,11  
**SavedValue** (Function) 19-21  
**SaveInIT** (Method of *LatticeBrowser*) 10-47  
**SaveInstance** (Method of Object) 14-8  
**SaveInstance?** (Method of Object) 14-9  
**SaveValue** (Browser Submenu Option) 10-8  
 saving  
     an object on a file 14-6  
**ScrollWindow** (Method of Window) 19-7  
 SEdit 1-10  
**Select file** (Browser Submenu Option) 10-26  
**selectedFile** (Browser Submenu Option) 10-25

**SelectFile** (Lambda NoSpread Function) 19-14  
 selector 1-3  
**SelectorOfMethodBeingCompiled** (Macro) 6-10  
**SelectorsWithBreak** (Method of Class) 12-3  
**SEND** (Function) 7-2  
**SEND** (Macro) 7-2  
 sending  
     a message, errors 11-7  
**SendsMessage** (Browser Submenu Option) 10-28  
**SetName** (Method of Class) 3-16  
**SetName** (Method of Object) 2-2  
**SetProp** (Method of Window) 19-7  
**ShadeNode** (Method of *LatticeBrowser*) 10-47  
**Shape** (Method of *NonRectangular Window*) 19-20  
**Shape** (Method of Window) 19-7  
**Shape?** (Method of Window) 19-8  
**ShapeToHold** (Browser Submenu Option) 10-8  
**ShapeToHold** (Method of *LatticeBrowser*) 10-48  
**Show** (Method of *LatticeBrowser*) 10-48  
**Shrink** (Method of *LatticeBrowser*) 10-48  
**Shrink** (Method of Window) 19-8  
**Snap** (Method of Window) 19-8  
**Specialize** (Method of Class) 3-27  
**SpecializeClass** (Browser Submenu Option) 10-21  
**SpecializedClass** (Browser Submenu Option) 10-32  
**SpecializeMethod** (Browser Submenu Option) 10-21  
**SpecializesMethod** (Browser Submenu Option) 10-28  
 stack method macros 6-10  
 storing  
     a file 14-10  
     data in objects 1-4  
**SubBrowser** (Browser Menu Option) 10-16,31  
**SubBrowser** (Method of *LatticeBrowser*) 10-49  
 subclass 1-6  
**Subclass** (Method of Class) 3-28  
**SubClasses** (Method of Class) 3-28  
**SubclassResponsibility** (Macro) 6-6  
**Substitute** (Browser Submenu Option) 10-27  
**SubsTree** (Function) 3-29  
 superclass 1-7,8  
**SuperMethodNotFound** (Method of *Tofu*) 4-7  
 supers browser 10-2  
     menu interface 10-8  
 system function 20-1  
 system variable 20-1

## T

**THESE-INSTANCES** (File Package Command) 14-5  
 title  
     class inspector 18-7  
     ClassIV inspector 18-9  
     instance inspector 18-2  
 title bar menu  
     class inspector 18-8  
     class, meta, and supers browsers 10-8  
     ClassIV inspector 18-9  
     file browser 10-25  
     instance browser 10-51  
     instance inspector 18-3  
**TitleCommand** (Method of Object) 18-15  
**TitleSelection** (Method of *LatticeBrowser*) 10-49  
**TitleSelection** (Method of Window) 19-16  
 Tofu 4-6  
 tools 1-10

**ToTop** (*Method of Window*) 19-8  
**Trace on Access** (*Inspector Submenu Option*) 18-6  
**Trace on Put** (*Inspector Submenu Option*) 18-6  
**TraceFunction** (*Browser Submenu Option*) 10-29  
**TraceIt** (*Function*) 12-6  
**TraceIt** (*Inspector Menu Option*) 18-6  
**TraceIt** (*Method of Object*) 12-5  
**TraceMethod** (*Browser Submenu Option*) 10-20  
**TraceMethod** (*Method of Class*) 12-2  
**TraceOnPut** (*Class*) 8-10  
**TraceOnPutOrGet** (*Class*) 8-10  
tracing 12-1  
**type? annotatedValue** (*Macro*) 8-25  
**TypeInName** (*Browser Menu Option*) 10-16,31

## U

UID, see Unique Identifier  
**UID** (*Function*) 17-15  
**UnbreakFunction** (*Browser Submenu Option*) 10-29  
**UnBreakIt** (*Function*) 12-6  
**UnBreakIt** (*Inspector Menu Option*) 18-6  
**UnbreakMethod** (*Browser Submenu Option*) 10-20  
**UnbreakMethod** (*Method of Class*) 12-2  
**Understands** (*Method of Object*) 9-3  
**UNDO** (*Program Assistant Command*) 14-3  
Unique Identifier 15-1; 17-14  
**UnmarkNodes** (*Method of LatticeBrowser*) 10-49  
**UnSetName** (*Method of Class*) 3-17  
**UnSetName** (*Method of Object*) 2-3  
**Update** (*Method of Window*) 19-9  
**UpdateClassBrowsers** (*Function*) 10-52  
**UpdateClassBrowsers?** (*Variable*) 10-52  
**Use saved value** (*Inspector Submenu Option*) 18-5  
user interface to inspector 18-2  
**Uses IV?** (*Browser Menu Option*) 10-26  
**UsesCV** (*Browser Submenu Option*) 10-28  
**UsesIV** (*Browser Menu Option*) 10-33  
**UsesIV** (*Browser Submenu Option*) 10-27  
**UsesLispVar** (*Browser Submenu Option*) 10-28  
**UsesObject** (*Browser Submenu Option*) 10-28

## V

**ValueFound** (*Macro*) 2-25  
variable 1-4

## W

**WhenMenuItemHeld** (*Method of Window*) 19-17  
**WhereIs** (*Browser Menu Option*) 10-14  
**WhereIs** (*Method of Object*) 2-24  
**WhereIs (WhereIsMethod)** (*Browser Menu Option*) 10-30  
**WhereIsCV** (*Browser Submenu Option*) 10-15  
**WhereIsIV** (*Browser Submenu Option*) 10-15  
**WhereIsMethod** (*Browser Submenu Option*) 10-15  
**WhoHas** (*Function*) 3-22  
**Window** (*Class*) 19-1  
**WindowAfterMoveFn** (*Function*) 19-9  
**WindowButtonEventFn** (*Function*) 19-17  
**WindowRightButtonFn** (*Function*) 19-17  
**WindowShapeFn** (*Function*) 19-9  
wrapped value 8-19  
**WrappingPrecedence** (*Method of ActiveValue*) 8-18  
**WrappingPrecedence** (*Specialization of ActiveValue*) 8-16

←  
← (*Function*) 7-1  
← (*Macro*) 7-1  
←! (*Function*) 7-2  
←! (*Macro*) 7-2  
←@ (*Macro*) 5-12  
←AV (*Macro*) 8-26  
←IV (*Macro*) 7-2  
←New (*Macro*) 2-6  
←New (*NLambda NoSpread Macro*) 7-5  
←Process (*Macro*) 16-1  
←Process! (*Macro*) 16-2  
←Proto (*Macro*) 7-3  
←Super (*Macro*) 7-3  
←Super (*Function*) 7-3  
←Super? (*Macro*) 7-5  
←SuperFringe (*Macro*) 7-5  
←SuperFringe (*Function*) 7-5  
←Try (*Macro*) 7-3

## #

#, 18-1

## \$

\$ (*Macro*) 2-2  
\$ (*NLambda Function*) 2-2; 17-1  
\$! (*Function*) 2-2  
\$! (*Lambda Function*) 17-2  
\$AV (*NLambda NoSpread Function*) 8-27  
\$C (*NLambda Function*) 17-2

## \*

\*any\* (*Browser Submenu Option*) 10-27  
\*EditAll\* (*Browser Submenu Option*) 10-27; 27  
\*FEATURES\* (*Variable*) 20-2  
\*hiddenFile\* (*Submenu Option*) 10-5  
\*loadFile\* (*Submenu Option*) 10-5  
\*newFile\* (*Submenu Option*) 10-5  
\*NewFunction\* (*Browser Submenu Option*) 10-29  
\*other\* (*Browser Submenu Option*) 10-27  
\*SubstituteAll\* (*Browser Submenu Option*) 10-27

## :

:initForm (*Property*) 2-9

## ?

?= 18-16

## @

@ (*Macro*) 5-10  
@\* (*Macro*) 5-12

ntionally left blank]

[This page inte



## 7. MESSAGE SENDING FORMS

Objects in LOOPS communicate with each other by sending messages. This chapter describes the standard message sending forms used in LOOPS.

The following table shows the macros in this section.

| Name                      | Type                   | Description                                                                                                                                      |
|---------------------------|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>←</code>            | Macro and Function     | Sends a message to an object.                                                                                                                    |
| <b>SEND</b>               | Macro and Function     | Sends a message to an object.                                                                                                                    |
| <code>←!</code>           | Macro and Function     | Evaluates the selector and sends a message to an object.                                                                                         |
| <code>←IV</code>          | Macro                  | Invokes the function stored in an instance variable of the object.                                                                               |
| <code>←Try</code>         | Macro                  | Sends a message to an object only if it has a corresponding method.                                                                              |
| <code>←Proto</code>       | Macro                  | Sends a message to the prototype instance of a class.                                                                                            |
| <code>←Super</code>       | Macro and Function     | Combines an inherited method with local code; must appear in the body of a method.                                                               |
| <code>←Super?</code>      | Macro                  | Combines an inherited method with local code; must appear in the body of a method. This does not cause an error if there is no inherited method. |
| <code>←SuperFringe</code> | Macro and Function     | Invokes general methods for objects with more than one super class from which to inherit methods; must appear in the body of a method.           |
| <code>←New</code>         | NLambda NoSpread Macro | Creates an instance of a class and then sends a message to that instance.                                                                        |
| <b>FetchMethod</b>        | Macro                  | Finds the function name which implements the method invoked by a selector.                                                                       |

In addition, Chapter 8, Active Values, contains a description of `←AV`, and Chapter 15, Performance Issues, contains a description of `←Process` and `←Process!`.

`(← self sel arg1 ... argn )` [Macro and Function]

Purpose: Sends the message with the selector *sel* to an object *self*. This is the standard way to send a message.

Behavior: Evaluates all arguments except *sel*.

When an object receives a message, it tries to match the selector *sel* with the names of its methods. If the object or the message does not recognize the message, a **Not Understood** error occurs.

The function version does more error checking than the macro and also attempts to convert unbound symbols into names for classes and instances.

Arguments: *self*            Pointer to an object.  
*sel*            Selector; not evaluated.  
*arg1...argn*   Arguments associated with *sel*.

Returns:    The value returned by the method associated with *sel*.

Example:    In this example, the message **New** is sent to the class **Window**. This returns the newly created instance.

```
76←(← ($ Window) New 'Window1)
#,$(& Window (|OZW0.1Y:.;h.Qm:| . 495))
```

---

**(SEND *self sel arg1 ... argn*)** [Macro and Function]

---

Purpose:    Same as ←, above.

Example:    The expression

```
(SEND ($ Window) 'New 'Window1)
```

is equivalent to

```
(← ($ Window) New 'Window1)
```

---

**(←! *self sel arg1 ... argn*)** [Macro and Function]

---

Purpose/Behavior:    Sends a message with the selector *sel* to an object *self*. It differs from ← in that it evaluates all of its arguments, including *sel*.

Arguments:    *self*            Pointer to an object.  
*sel*            Selector, which is evaluated.  
*arg1...argn*   Arguments associated with *sel*.

Example:    This example illustrates the fact that ←! evaluates the *sel* argument.

The code

```
(for sel in '(Shape Invert)
 do (←! ($ Window1 sel))
```

is equivalent to

```
(←Window1 Shape) (←Window1 Invert)
```

---

**(←IV *self IVName arg1...argn*)** [Macro]

---

Purpose:    Invokes the function stored in the instance variable *IVName* of the object *self*.

Behavior:    Gets a function from *IVName* of *self* and applies the function to *self* with the arguments *args*. Returns the value of the function or breaks.

            ←IV does not evaluate *IVName*.

Arguments:    *self*            Pointer to an object.  
*IVName*       Instance variable name, which is not evaluated.

*arg1...argn* Arguments associated with *sel*; bound to arguments specified in the call.

**(←Try self sel arg1 ... argn)** [Macro]

**Purpose:** Sends the message with the selector *sel* to *self*, but only if there is a corresponding method.

**Behavior:** If *sel* is in fact a selector of *self*, the method is applied and the appropriate value is returned. If the method is not a selector of *self*, the symbol **NotSent** is returned.

**Arguments:** *self* Pointer to an object.  
*sel* Selector; not evaluated.  
*arg1...argn* Arguments associated with *sel*.

**Example:** The expression (←(\$ Window1) abcd) normally causes a break.

```
79←(←Try ($ Window1) Update)
NIL

80←(←Try ($ Window1) abcd)
NotSent
```

**(←Proto class sel arg1 ... argn)** [Macro]

**Purpose:** Sends a message to the prototype instance of a class.

**Behavior:** Creates an instance of a class, if necessary, and puts that instance on the class variable **Prototype** of *class*, marking the class as changed. This instance is referred to as the prototype instance. **Proto** then sends the message *sel* to that instance.

**Arguments:** *class* Pointer to a class.  
*sel* Selector; not evaluated.  
*arg1...argn* Arguments associated with *sel*.

**Example:** Usually only one instance of **LoopsIcon** is needed at a time, so the class **LoopsIcon** keeps one in its class variable **Prototype**.

```
81←(←Proto ($ LoopsIcon) Open)
```

**(←Super self sel arg1 ... argn)** [Macro and Function]

**Purpose:** Can invoke an inherited method within a method. ←**Super** must appear in the body of a method; it cannot be invoked directly.

**Behavior:** Searches up the class hierarchy and invokes the next more general method of the same name, even if a specialized method is inherited over a distance. It returns the value from that super method. You can use the form (←**Super**) when the arguments are not changed. If no arguments are provided, ←**Super** uses the arguments of the method from which it was called.

←**Super** and the other similar functions are now lexically scoped; that is, it is illegal to call ←**Super** anywhere but within a method body, and any selector given must be the same as the selector for that method.

**Arguments:** *self* Pointer to an object.  
*sel* Selector; not evaluated. Must be the same as the selector of the method in which the ←**Super** appears.

*arg1...argn* Arguments associated with *sel*.

Example: Two examples of **←Super** are included:

- One example shows where the arguments are not changed.
- One example shows where the arguments are changed.

*Example 1:* A use of **←Super** where the arguments are not changed.

Define a subclass of **Window** that will call **RINGBELLS** before a window is shaped.

```
(DefineClass 'RingingWindow ' (Window))
```

Through the browser interface, specialize the method **Shape**, to create the following method.

```
(RingingWindow.Shape
 (Method ((RingingWindow Shape)
 self newRegion noUpdateFlg)
 COMMENT **COMMENT**
 (RINGBELLS)
 (←Super)))
```

Executing the following command calls **RINGBELLS** before the new window is shaped.

```
(←New ($ RingingWindow) Shape)
```

In the method above, if the positions of **RINGBELLS** and **(←Super)** were reversed, **RINGBELLS** would be called after the window was shaped.

*Example 2:* A use of **←Super** where the arguments are changed.

Define a subclass of **Window** that will be square.

```
(DefineClass 'SquareWindow ' (Window))
```

Through the browser interface, specialize the method **Shape**, to create the following method.

```
(SquareWindow.Shape
 (Method ((SquareWindow Shape)
 self newRegion noUpdateFlg)
 COMMENT **COMMENT**
 (←Super self Shape
 (create REGION
 using
 (SETQ newRegion
 (OR newRegion
 (GETREGION)))
 HEIGHT ← (fetch WIDTH
 of
 newRegion)
 noUpdateFlg)))
```

Executing the following command creates a square window:

```
(←New ($ SquareWindow) Shape)
```

---

(←**Super?** *self sel arg1 ... argn*) [Macro]

---

- Purpose: Invokes the single next most general method; must appear in the body of a method. This does not cause an error if no inherited method matches.
- Behavior: Analogous to ←**Super**. The difference between ←**Super?** and ←**Super** is that ←**Super?** does not break if the *sel* does not have a more general method, whereas ←**Super** generates a break if there is not a more general method.
- Arguments: *self* Pointer to an object.
- sel* Selector; not evaluated. Must be the same as the selector of the method in which the ←**Super?** appears.
- arg1...argn* Arguments associated with *sel*.

---

(←**SuperFringe** *self sel arg1 ... argn*) [Macro and Function]

---

- Purpose: Invokes general methods for objects with more than one super class from which you wish to inherit methods; must appear in the body of a method.
- Behavior: It invokes and executes the next more general method of the same name from each of the classes on the super's list *object's* class. Calling ←**SuperFringe** is analogous to sending ←**Super** up through each item on the super's list. If no arguments are provided ←**SuperFringe** uses the arguments of the method from which it was called.
- Arguments: *self* Pointer to an object.
- sel* Selector; not evaluated. Must be the same as the selector of the method in which the ←**SuperFringe** appears.
- arg1...argn* Arguments associated with *sel*.

---

(←**New** *class selector arg1 ... argn*) [NLambda NoSpread Macro]

---

- Purpose: Creates an instance of class and then sends *sel* and arguments to that instance.
- Behavior: Creates a new instance of a class and sends a message to that instance. It returns the instance as a value and discards any value that may be returned by invoking the method specified by selector. ←**New** is equivalent to (← (← *ClassName New*) *selector arg1 ... argn*).
- Arguments: *class* Pointer to a class.
- sel* Selector; not evaluated.
- arg1...argn* Arguments associated with *sel*.
- Returns: The new instance.
- Example: This example shows an example of ←**New** that creates a new instance of the class **Window** and asks you to shape it.

```
99← (←New ($ Window) Shape)
#, ($& Window (|OZW0.1Y:.;h.Qm:| . 497))
```

---

(← *class* **FetchMethod** *sel*) [Method of Class]

---

- Purpose: Finds the function name which implements the method invoked by sending a message with the selector *sel* to an instance of *class*. The function can be found in either *class* or its supers.

Behavior: Calls the function **FetchMethod**.

Arguments: *class*            Pointer to a class.  
*sel*                    Selector; evaluated.

Returns: The function for *sel* or NIL.

Example: Line 100 shows that the class **Window** implements the method **Update**.

```
100←(← ($ Window) FetchMethod 'Update)
Window.Update
```

Line 1 shows that neither the class **Window** nor any of its supers implements the method **abcd**.

```
1←(← ($ Window) FetchMethod 'abcd)
NIL
```

Line 2 shows that the class **Object** implements the method **PP** which will be triggered when instances of the class **Window** receive the **PP** message.

```
2←(← ($ Window) FetchMethod 'PP)
Object.PP
```

[This page intentionally left blank]

Active values are used in LOOPS to transpose the access of data within an object to a message being sent to a different object. Typical uses include:

- Causing side effects to occur when data is accessed
- Debugging
- Initializing variables
- Maintaining constraints between variables

An **ActiveValue** is an instance of a subclass of the LOOPS class **ActiveValue**. When an **ActiveValue** instance is installed in the value of a variable, further references to that variable cause messages to be sent to the instance.

LOOPS provides several kinds of active values which are described in this chapter. You can obtain new behavior by specializing one of the existing LOOPS **ActiveValue** subclasses.

When **GetValue** notices that an **ActiveValue** is installed on the variable, it sends the **GetWrappedValue** message to the **ActiveValue**. Similarly, when **PutValue** notices that an **ActiveValue** is installed on the variable, it sends the **PutWrappedValue** message to the **ActiveValue**. The value returned from the get or put is the value returned from the message send. Each specialization of **ActiveValue** either inherits these methods from its superclasses or specializes them to call user code. The messages are received and processed by the **ActiveValue** instances.

For example, assume that you are modeling a simulation that requires the value of an instance variable called `windSpeed` to be a random value. You can make the value of `windSpeed` into an active value called (`$ RandomWindSpeedAV1`). Now, if you try to determine the value of `windSpeed` by entering

```
(@ ($ SomeAirport) windSpeed)
```

the value returned from this expression is the value returned from

```
(← ($ RandomWindSpeedAV1) GetWrappedValue . otherArgs)
```

This returns the required random value.

The variable containing the **ActiveValue** may still have a current value. Most system **ActiveValue** subclasses are specializations of **LocalStateActiveValue**, which uses an instance variable `localState` in the **ActiveValue** to hold the value.

For efficient implementation, LOOPS uses a special Interlisp data type, the `annotatedValue` data type, to "wrap" each **ActiveValue** instance when it is installed as a value within an object; the `annotatedValue` contains the **ActiveValue** instance. That is, if the value of an instance variable is said to be an active value, in actuality, the value of the instance variable is an `annotatedValue` which contains the active value. This allows every **GetValue** or **PutValue** to use Interlisp's microcoded type checking mechanism to see if it should be processed normally or via the **ActiveValue** mechanism. This extra layer is largely invisible in application programs. LOOPS also contains a class



**AnnotatedValue** to handle the occasional accident when a user forgets about the distinction between **annotatedValue** and **ActiveValue**, and attempts to treat an **annotatedValue** as a **LOOPS** object.

The class **ActiveValue** defines the general protocol followed by all active value objects. Methods setting up the basic functionality of **ActiveValues** are defined in this class and inherited by all its specializations. Methods defined in this class include **AVPrintSource** to specify how **annotatedValues** print, **AddActiveValue** to install an **ActiveValues**, and **DeleteActiveValue** to delete an installed **ActiveValue**.

The class **ActiveValue** itself is an abstract class; that is, it is a placeholder in the class hierarchy that is not intended to be instantiated. When this documentation refers to an active value object, it is referring to an instantiation of a specialization of the class **ActiveValue**.

Note: The current **ActiveValue** is different from the **activeValue** implementation in the Buttruss version of **LOOPS**. See Appendix A, Active Values in Buttruss **LOOPS**, for more information.

---

## 8.1 Using Active Values

---

A general template is available when using active values. As with all templates, you should not blindly follow it. A good understanding of the active value mechanism is necessary to avoid errors in more complicated situations.

- Determine the functionality that you want the active value to provide. For example, will it cause a side effect to occur on access of data? Will it maintain constraints between two pieces of data? The required functionality will give an indication of which active value class you should use.
- Specialize one of the active value classes to satisfy your specific requirements, if necessary.
- Create an instance of the active value class that you have chosen or created.
- Initialize the contents of that instance, if necessary.
- Install that active value instance on the data that you want to become active. This is accomplished by using the **AddActiveValue** message.

In a number of situations, you may want to install an active value on an instance variable for every instance of a class. One technique for doing this is discussed in Section 8.5, "Active Values in Class Structures."

### 8.2 SPECIALIZATIONS OF THE CLASS ACTIVEVALUE

---

## 8.2 SPECIALIZATIONS OF THE CLASS ACTIVEVALUE

---

---

## 8.2 Specializations of the Class ActiveValue

---

The class **ActiveValue** is an abstract class, and therefore cannot be instantiated. This class contains a number of methods, described in Section 8.3, "ActiveValue Methods," that are necessary for the active value mechanism to function. As a user, you will be making active values which are instances of some subclass of **ActiveValue**, either one of those already provided or one that you created. Figure 8-1 shows the class **ActiveValue** and its specializations. This section describes the subclasses of **ActiveValue**

in order of their appearance in this figure. Also included is information on specializing active values.

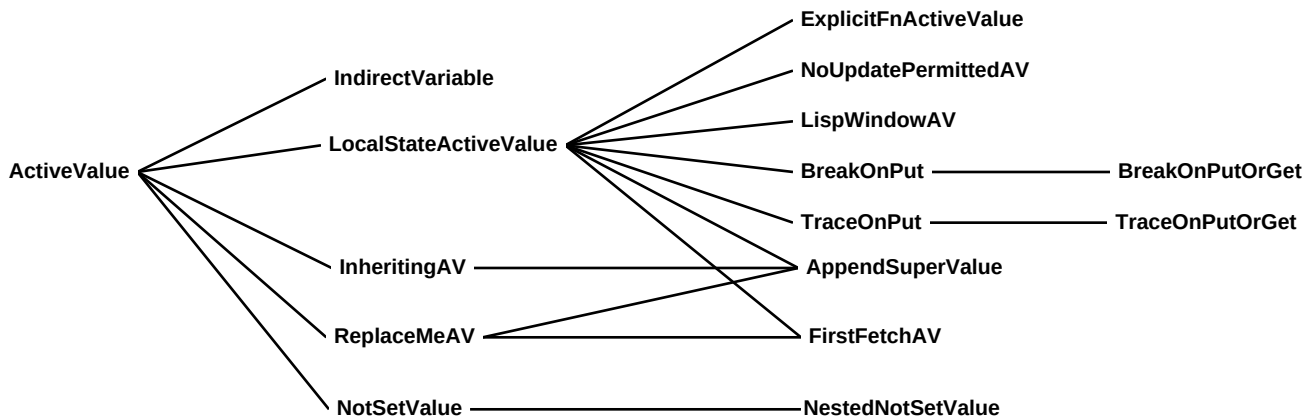


Figure 8-1. The Class **ActiveValue** and its Specializations

### 8.2.1 IndirectVariable

This specialization sets up the functionality of an **ActiveValue** to return the value of another variable as its value. It is analogous to the concept of indirect addressing in other computer languages.

Note: Indirect variables must be the innermost of nested active values. Wrapping precedence (see Section 8.3, "Active Value Methods") insures this.

#### IndirectVariable [Class]

**Purpose:** Enables variable values to be accessed indirectly from other variables. This simulates two variables sharing the same memory location. This is a useful technique for implementing simulations and enforcing constraints.

**Behavior:** When a Fetch is made on the variable containing the **IndirectVariable** instance, this active value retrieves and returns the value of the tracked variable. If a Store is made with the variable containing the **IndirectVariable** instance, this active value stores the new value in the tracked variable. Essentially, this forces the two variables to share the same data.

**Instance Variables:**

- object** Object instance containing the tracked value.
- varName** The name of the variable being tracked.
- propName** If non-NIL, the name of the variable property being tracked.
- type** Type of variable being tracked. Value can be CV, IV or NIL, which defaults to IV.

**Examples:** Several examples are included to show the use of **IndirectVariable**.

**Example 1:** Consider a chemical reactor simulation where you have a tank draining into a pipe. The output pressure of the tank needs to equal the input pressure of the pipe. The following demonstrates this.

First, build the appropriate pipe and tank classes and make instances of them.

```

78← (DefineClass 'Tank)
 #, ($C Tank)

79← (DefineClass 'Pipe)
 #, ($C Pipe)

```

```
80← (← ($ Tank) AddIV 'outputPressure)
outputPressure
```

```
81← (← ($ Pipe) AddIV 'inputPressure)
inputPressure
```

```
82← (← ($ Tank) New 'tank1)
#,$& Tank (NYW0.0X%:.aF4.6R8 . 5))
```

```
83← (← ($ Pipe) New 'pipe1)
#,$& Pipe (NYW0.0X%:.aF4.6R8 . 6))
```

Create an instance of **IndirectVariable** and initialize its contents to point to the tank's output pressure.

```
84← (← ($ IndirectVariable) New 'indVar1)
#,$& IndirectVariable (NYW0.0X%:.aF4.6R8 . 7))
```

```
85← (←@ ($ indVar1) object ($ tank1))
#,$& Tank (NYW0.0X%:.aF4.6R8 . 5))
```

```
86← (←@ ($ indVar1) varName 'outputPressure)
outputPressure
```

Install the active value instance as the value of the pipe's input pressure.

```
87← (← ($ indVar1) AddActiveValue ($ pipe1) 'inputPressure)
#,$AV IndirectVariable (indVar1 (NYW0.0X%:.aF4.6R8 . 7)) (object
#,$& Tank (NYW0.0X%:.aF4.6R8 . 5))) (varName outputPressure))
```

Accesses to either the pipe's input pressure or the tank's output pressure produce the same result.

```
90← (@ ($ pipe1) inputPressure)
NIL
```

```
92← (←@ ($ pipe1) inputPressure 100)
100
```

```
94← (@ ($ tank1) outputPressure)
100
```

```
95← (←@ ($ tank1) outputPressure 200)
200
```

```
96← (@ ($ pipe1) inputPressure)
200
```

An inspector window of (\$ pipe1) appears as follows:

```
All Values of Pipe ($ pipe1).
inputPressure #,$AV IndirectVariable (indVar1 (
```

**Example 2:** Consider a conveyor that must be three feet above a bin. Assume both have an instance variable named height.

First, create the classes and instances.

```
53← (DefineClass 'Bin)
#,$C Bin)
```

```
54← (DefineClass 'Conveyor)
#,$C Conveyor)
```

```
55← (← ($ Bin) AddIV 'height 0)
height
```

```

56← (← ($ Conveyor) AddIV 'height 0)
height

57← (← ($ Bin) New 'bin1)
#,$& Bin (|DAW0.1Y:.H53.]99| . 506))

58← (← ($ Conveyor) New 'conveyor1)
#,$& Conveyor (|DAW0.1Y:.H53.]99| . 507))

```

Create a specialization of **IndirectVariable** and specialize the methods **GetWrappedValue** and **PutWrappedValue**. You need to specialize **IndirectVariable** because you do not want to maintain equality between the two variables, but instead want to maintain a different mathematical relationship. The **\_Supers** are used to use the default behavior of **IndirectVariable** which takes care of retrieving or storing the data into the tracked variable.

```

59← (DefineClass '3FeetAbove
' (IndirectVariable))
#,$C 3FeetAbove)

```

```

SEdit 3FeetAbove.GetWrappedValue Package: INTERLISP
(Method ((3FeetAbove GetWrappedValue) self
 containingObj varName propName type)
 ;; Compute value of indirect variable and add 3 to it
 (PLUS 3
 (←Super self GetWrappedValue
 containingObj varName propName
 type))))

```

```

SEdit 3FeetAbove.PutWrappedValue Package: INTERLISP
(Method ((3FeetAbove PutWrappedValue) self
 containingObj varName newValue propName type)
 ;; Instead of placing newValue here, subtract 3
 ;; from it and put it in the indirect variable
 (←Super self PutWrappedValue containingObj
 varName (DIFFERENCE newValue 3)
 propName type)
 ;; return newValue
 newValue)

```

Create an instance of **3FeetAbove** and initialize its contents to point to the bin's height.

```

65← (← ($ 3FeetAbove) New '3fa1)
#,$& 3FeetAbove (|DAW0.1Y:.H53.]99| . 505))

66← (←@ ($ 3fa1) object ($ bin1))
#,$& Bin (|DAW0.1Y:.H53.]99| . 506))

67← (←@ ($ 3fa1) varName 'height)
height

```

Install this instance of **3FeetAbove** as the value of the conveyor's height.

```

68← (← ($ 3fa1) AddActiveValue ($ conveyor1) 'height)
#,$AV 3FeetAbove (3fa1 (|DAW0.1Y:.H53.]99| . 505))
 (object #,$& Bin (|DAW0.1Y:.H53.]99| . 506))
 (varName height))

```

The height of **bin1** defaults to 0, what is the height of **conveyor1**?

```

69← (@ ($ bin1) height)
0

```

```
70← (@ ($ conveyor1) height)
3
```

Set either **bin1's** height or **conveyor1's** height and notice how they track each other.

```
71← (←@ ($ bin1) height 15)
15
```

```
72← (@ ($ conveyor1) height)
18
```

```
73← (←@ ($ conveyor1) height 21)
21
```

```
74← (@ ($ bin1) height)
18
```

---

## 8.2.2 LocalStateActiveValue

---

This specialization sets up a class of **ActiveValue** that contains the instance variable **localState**, which is used primarily for storing the value of the referenced variable.

If you need an active value that will produce your own specific side-effect, you will most likely use your own specialization of **LocalStateActiveValue**. The data that would have been accessed, had an active value not been installed, is stored in the **localState** instance variable.

---

### LocalStateActiveValue

[Class]

**Purpose:** Creates a subclass of **ActiveValue** with an instance variable to hold the current value of the referenced variable.

**Behavior:** Holds the data that normally is stored in the variable where it is installed. At installation time, the current variable value is placed in the **localState** instance variable of the **ActiveValue**. Subclasses of **LocalStateActiveValues** are the most common **ActiveValue** instances.

The class **LocalStateActiveValue** is commonly specialized. In particular, it is usually desirable to specialize the methods **GetWrappedValue** and **PutWrappedValue** associated with new subclasses of **LocalStateActiveValue**. These methods implement the active value messages sent when the variable is accessed.

**Instance Variable:** **localState** Stores the value of the referenced variable.

**Examples:** Several examples are included to show the use of **LocalStateActiveValue**.

**Example 1:** In this example, an active value will print a message if an attempt is made to store an out-of-range value in an instance variable.

Define a subclass of **LocalStateActiveValue** and give it two instance variables that will store the values of the limits.

```
99← (DefineClass 'WarningAV' (LocalStateActiveValue))
#, ($C WarningAV)

100← (← ($ WarningAV) AddIV 'lowTrigger 0)
lowTrigger

101← (← ($ WarningAV) AddIV 'highTrigger 100)
highTrigger
```

Specialize **LocalStateActiveValue**'s **PutWrappedValue** method to create one for **WarningAV**.

```
SEdit WarningAV.PutWrappedValue Package: INTERLISP
(Method ((WarningAV PutWrappedValue) self containingObj
 varName newValue propName type)
 ;; If newValue is greater than highTrigger or lower
 ;; than lowTrigger, then print a message, but store
 ;; the data anyway
 (if (OR (< newValue (@ lowTrigger))
 (> newValue (@ highTrigger)))
 then (PRINTOUT T "The value " newValue
 " is out of range." T))
 (<Super self PutWrappedValue containingObj
 varName newValue propName type))
```

Create an instance of a Bin and attach an instance of a **WarningAV** to its height.

```
4← (← ($ Bin) New 'bin3)
#, ($& Bin (|DAW0.1Y:.H53.]99| . 513))

5← (←New ($ WarningAV) AddActiveValue ($ bin3) 'height)
#, ($& WarningAV (|DAW0.1Y:.H53.]99| . 514))
```

Attempt to store various values into the bin's height.

```
7← (←@ ($ bin3) height 10)
10

8← (←@ ($ bin3) height -10)
The value -10 is out of range.
-10

9← (←@ ($ bin3) height 110)
The value 110 is out of range.
110

10← (@ ($ bin3) height)
110
```

**Example 2:** In this example, an active value will return a random number when it is read from. Puts to it will change the range of the random value returned on gets. This will use **localState** for something other than storing the data for active values that provide only pure side-effect behavior.

```
99← (DefineClass 'RandomAV '(LocalStateActiveValue))
#, ($C RandomAV)

100← (← ($ RandomAV) AddIV 'localState '(0 100))
localState
```

Specialize **LocalStateActiveValue**'s **PutWrappedValue** and **GetWrappedValue** methods to create them for **RandomAV**.

```
SEdit RandomAV.GetWrappedValue Package: INTERLISP
(Method ((RandomAV GetWrappedValue) self containingObj
 varName propName type)
 ;; Get the value of the localState.
 ;; We'll assume for this example that the value returned
 ;; is a list of two numbers. If it is not, we get an error.
 (APPLY #'RAND (@ localState)))
```

```
SEdit RandomAV.PutWrappedValue Package: INTERLISP
(Method ((RandomAV PutWrappedValue) self containingObj
 varName newValue propName type)
 ;; We'll assume that the new value is a list of two
 ;; numbers.
 (←@ localState newValue))
```

Now, try to test this.

```
36← (DefineClass 'RandomTest)
#,$C RandomTest)

39← (← IT AddIV 'randomIV)
randomIV

40← (← ($ RandomTest) New 'rt1)
#,$& RandomTest (DCW0.0X%:.aF4.5S; . 518))

41← (←New ($ RandomAV) AddActiveValue ($ rt1) 'randomIV)
#,$& RandomAV (DCW0.0X%:.aF4.5S; . 519))

42← (@ ($ rt1) randomIV)
24

43← redo
32

44← redo
9

45← redo
49

46← (←@ ($ rt1) randomIV '(4.0 5.0))
(4.0 5.0)

47← (@ ($ rt1) randomIV)
4.190201

48← REDO
4.1129

49← REDO
4.380234

50← REDO
4.397278
```

---

### 8.2.2.1 ExplicitFnActiveValue

**ExplicitFnActiveValue** emulates the activeValue implementation from the Buttress version of LOOPS. Users are discouraged from using this particular form of active values within new projects.

See the *LOOPS Users' Modules* for details on *LOOPSBACKWARDS*, which describes **ExplicitFnActiveValue**. See Appendix A, Active Values in Buttriss *LOOPS*, for details on the compatibility of **ActiveValue** with *activeValue*.

### 8.2.2.2 NoUpdatePermittedAV

This specialization sets up a class of **ActiveValue** that prevents the value of a variable from being replaced.

#### NoUpdatePermittedAV

[Class]

- Purpose:** Prevents the value of a variable from being replaced using the **PutValue** method.
- Behavior:** Stores the current value of the variable in **localState**, then prevents it from being updated. **GetWrappedValue** requests return the value found in **localState**, but **PutWrappedValue** requests cause a break with the break message **NoUpdatePermitted!**, or a message if sent from the Exec.
- Example:** Suppose an identification number for a piece of data should never be changed. Installing a **NoUpdatePermittedAV** in the data's ID number will cause a break if a replacement attempt is made.

Start with a user-defined class named **Datum**. Make a **Datum** instance named **Datum1**. Set the instance variable named **idNumber** to the value 999. Look at the instance. Make a new instance of **NoUpdatePermittedAV**, and name it **NumberGuard**. Install the **ActiveValue** in the instance variable **idNumber** of the instance **Datum1**. Look at the **ActiveValue** instance; the **localState** instance variable contains the previous value of **idNumber**. To test this **ActiveValue**, attempt to replace the **idNumber** of **Datum1** with a new value.

```
67← (DefineClass 'Datum)
#, ($C Datum)

68← (← ($ Datum) AddIV 'idNumber 0)
idNumber

69← (← ($ Datum) New 'Datum1)
#, ($& Datum (|DAW0.1Y:.H53.]99| . 524))

70← (←@ ($ Datum1) idNumber 999)
999

71← (← ($ Datum1) PP)
(DEFINST Datum (Datum1 (|DAW0.1Y:.H53.]99| . 524)) (idNumber 999))
#, ($& Datum (|DAW0.1Y:.H53.]99| . 524))

74← (← ($ NoUpdatePermittedAV) New 'NumberGuard)
#, ($& NoUpdatePermittedAV (|DAW0.1Y:.H53.]99| . 525))

75← (← ($ NumberGuard) AddActiveValue ($ Datum1) 'idNumber)
#, ($AV NoUpdatePermittedAV (NumberGuard (|DAW0.1Y:.H53.]99| . 525))
(localState 999))

76← (← ($ Datum1) PP)
(DEFINST Datum (Datum1 (|DAW0.1Y:.H53.]99| . 524))
(idNumber #, ($AV NoUpdatePermittedAV (NumberGuard
(|DAW0.1Y:.H53.]99| . 525)) (localState 999))))
#, ($& Datum (|DAW0.1Y:.H53.]99| . 524))

77← (←@ ($ Datum1) idNumber 888)
No update permitted!
NIL
```



### 8.2.2.3 LispWindowAV

---

This specialization sets up a class of **ActiveValue** used by the system to guarantee that the **window** instance variable within a LOOPS **Window** instance contains an Interlisp window. This class provides functionality required by the LOOPS system, and should not generally be used by LOOPS users.

---

|                     |         |
|---------------------|---------|
| <b>LispWindowAV</b> | [Class] |
|---------------------|---------|

---

- |           |                                                                                                                                                                                                                                                                                                                                                  |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:  | Guarantees that a variable contains a window which has been made into a LOOPS window.                                                                                                                                                                                                                                                            |
| Behavior: | Meant to be installed only in the instance variable <b>window</b> of instances of class <b>Window</b> . A specialization of <b>LocalStateActiveValue</b> . Checks to see if its <b>localState</b> is a window, and assures that other instance variables of the window instance are set correctly. See Chapter 19, Windows, for further details. |

### 8.2.2.4 Breaking and Tracing Active Values

---

The following active values are all specializations of **LocalStateActiveValue** and are used for debugging, as described in Chapter 12, Breaking and Tracing. This chapter also describes **UnbreakIt**, which unbreaks or untraces a method of a class. These classes provide functionality required by the LOOPS system, and are not generally used by LOOPS users.

Note: All breaks and traces occur before the variable is read or modified.

---

|                   |         |
|-------------------|---------|
| <b>BreakOnPut</b> | [Class] |
|-------------------|---------|

---

- |           |                                                                                                                                                             |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:  | Breaks when a replacement attempt is made.                                                                                                                  |
| Behavior: | Breaks when a replacement attempt is made. Local variables bound at the time of the break are <b>containingObj</b> , <b>varName</b> , and <b>propName</b> . |

---

|                        |         |
|------------------------|---------|
| <b>BreakOnPutOrGet</b> | [Class] |
|------------------------|---------|

---

- |           |                                                                                                                                                                                        |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:  | Breaks when a retrieval or replacement of a variable is made. This is a specialization of <b>BreakOnPut</b> .                                                                          |
| Behavior: | Break occurs before any access to the variable where it is installed. Local variables bound at the time of the break are <b>containingObj</b> , <b>varName</b> , and <b>propName</b> . |

---

|                   |         |
|-------------------|---------|
| <b>TraceOnPut</b> | [Class] |
|-------------------|---------|

---

- |           |                                                                                                                                                                                                                             |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:  | Traces replacements of a variable.                                                                                                                                                                                          |
| Behavior: | Has a specialized <b>PutWrappedValue</b> method that causes the values of the arguments <b>containingObj</b> , <b>varName</b> , and <b>propName</b> to print in the trace window when the variable is about to be modified. |

---

|                        |         |
|------------------------|---------|
| <b>TraceOnPutOrGet</b> | [Class] |
|------------------------|---------|

---

- |          |                                                                                                   |
|----------|---------------------------------------------------------------------------------------------------|
| Purpose: | Traces retrievals and replacements of a variable. This is a specialization of <b>TraceOnPut</b> . |
|----------|---------------------------------------------------------------------------------------------------|

Behavior: The **GetWrappedValue** method is also specialized so that the variable is traced before any access of the variable where it is installed.

### 8.2.2.5 AppendSuperValue

This specialization allows the value of a variable to reside only partially in the local instance or class. This is a specialization of the **ReplaceMeAV**, **InheritingAV**, and **LocalStateActiveValue** classes.

#### AppendSuperValue [Class]

Purpose: Allows the value of a variable to be defined by both a local value and an inherited value.

Behavior: When an instance of **AppendSuperValue** is installed in a variable, Get-references return its **localState** appended to the end of the inherited value the variable would have if it had no local value. Any **PutValue** to the variable replaces the active value, not just the **localState**; **InheritingAV** and its specializations are designed for use more in class variables where replacement is infrequent.

Examples: Several examples are included to show the use of **AppendSuperValue**.

**Example 1:** Append the **localState** of the instance variable **idNumber** to the default value specified in the class description.

```
23← (DefineClass 'Datum)
#, ($C Datum)

24← (← ($ Datum) AddIV 'idNumber '(5))
idNumber

25← (← ($ Datum) New 'Datum1)
#, ($ Datum1)

26← (@ ($ Datum1) idNumber)
(5)

27← (←@ ($ Datum1) idNumber '(9))
(9)

28← (@ ($ Datum1) idNumber)
(9)

29← (←New ($ AppendSuperValue) AddActiveValue ($ Datum1) 'idNumber)
#, ($& AppendSuperValue (45 . 54648))

30← (@ ($ Datum1) idNumber)
(5 9)
```

**Example 2:** In this example, there are two classes of cars; the **Two-tone-Car** class is a subclass of the class **Car**. Each **Car** class has the instance variable **color**. The default value for **color** in the class **Car** is (white).

```
89← (DefineClass 'Car)
#, ($C Car)

90← (DefineClass 'Two-tone-Car '(Car))
#, ($C Two-tone-Car)

91← (← ($ Car) AddIV 'color '(white))
color
```

```
92← (← ($ Two-tone-Car) AddIV 'color)
color
```

The default value for color in the class **Two-tone-Car** is an instance of **AppendSuperValue** with its **localState** set to (blue). The technique for adding active values as default values in a class is discussed in Section 8.3.1, "Adding and Deleting Active Values."

```
100← (← ($ AppendSuperValue) New 'asv1)

1← (←@ ($ asv1) localState '(blue))

2← (PutClassIV ($ Two-tone-Car) 'color
 (create annotatedValue
 annotatedValue ← ($ asv1)))
#, ($AV AppendSuperValue (asv1 (|DAW0.1Y:.H53.]99| . 528))
 (localState (blue)))

9← (← ($ Car) New 'car1)
#, ($& Car (|DAW0.1Y:.H53.]99| . 531))

10← (← ($ Two-tone-Car) New 'ttcar1)
#, ($& Two-tone-Car (|DAW0.1Y:.H53.]99| . 532))

11← (@ ($ car1) color)
(white)
```

When an instance of a **Two-tone-Car** is created the default value for its instance variable **color** is the combination of the values in both the classes **Car** and **Two-tone-Car**. The first inspector shows the existence of the active value that provides this behavior. As soon as one puts a value for color in this instance, the **AppendSuperValue** active value is replaced by the new value as shown in the second inspector.

```
12← (@ ($ ttcar1) color)
(white blue)
```

```
13← (INSPECT ($ ttcar1))
{WINDOW}#50,5000
```

```
All Values of Two-tone-Car ($ ttcar1).
color #,($AV AppendSuperValue (asv1 (|
```

```
14← (←@ ($ ttcar1) color '(tan brown))
(tan brown)
```

```
All Values of Two-tone-Car ($ ttcar1).
color (tan brown)
```

For another example, see the **TitleItems** class variable of the class **ClassBrowser**, where **AppendSuperValue** is used to add menu items to an inherited menu.

---

### 8.2.2.6 FirstFetchAV

This specialization has instances that have an expression as the value of the instance variable **localState**. These active values allow a form to be evaluated the first time that they are read.

---

#### FirstFetchAV

[Class]

Purpose: This is a specialization of the **ReplaceMeAV** mixin and **LocalStateActiveValue**. Instances of this class have an expression as the value of the instance variable **localState**.

**Behavior:** On the first get access, the expression in **localState** is evaluated. The resulting value replaces the **FirstFetchAV** so the variable is no longer an active value. On the first put access, the put value replaces the **FirstFetchAV** so the variable is no longer an active value. A **FirstFetchAV** is often used as the default value for a variable. This class also specializes the method **AVPrintSource** so that instances print as follows when wrapped in an **annotatedValue**:

```
#, (Defer contentsOfLocalState)
```

---

### CAUTION

**FirstFetchAVs** cannot be shared. Unlike lists, SEdit does not make copies of active values. Hence, if active values are copied in SEdit, they will share structure, and if one is modified, all will be changed.

**Workaround:** Use **CopyActiveValue** to copy the active value instance and the local state into each instance which uses the **FirstFetchAV**. See Section 8.3.4, "Shared ActiveValues in Variable Inheritance," for information on **CopyActiveValue**.

---

**Example:** An example application of **FirstFetchAV** is an instance variable that stores a font descriptor. A font descriptor in a class definition does not save correctly; only the pointer to the descriptor is saved. A **FirstFetchAV** stores the expression used to create the descriptor. So, for example the expression held in the **localState** of the **FirstFetchAV** is

```
(FONTCREATE 'HELVETICA 12 'BOLD)
```

On the first access of the instance variable, the font descriptor produced by calling **FONTCREATE** replaces the **FirstFetchAV**.

The complete example follows.

```
29← (DefineClass 'TextObject)
#, ($C TextObject)

30← (← ($ TextObject) AddIV 'font)
font

31← (← ($ FirstFetchAV) New 'ffav1)
#, ($& FirstFetchAV (|DAW0.1Y:.H53.]99| . 535))

32← (←@ ($ ffav1) localState '(FONTCREATE 'HELVETICA 12 'BOLD)]
(FONTCREATE (QUOTE HELVETICA) 12 (QUOTE BOLD))

33← (PutClassIV ($ TextObject) 'font
 (create annotatedValue
 annotatedValue ← ($ ffav1)))
#, (Defer (FONTCREATE (QUOTE HELVETICA) 12 (QUOTE BOLD)))

34← (← ($ TextObject) Edit)
TextObject
```

```
SEdit #,($C TextObject) Package: INTERLISP
((MetaClass Class Edited%: ; Edited 4-Dec-87
 ; 14:22 by RBGMartin
)
 (Supers Object) (ClassVariables)
 (InstanceVariables
 (font #,(Defer (FONTCREATE (QUOTE HELVETICA) 12 (QUOTE BOLD)))
 doc (* IV added by MARTIN)))
 (MethodFns))
```

```
35← (← ($ TextObject) New 'to1)
#,($& TextObject (|DAW0.1Y:.H53.]99| . 536))

36← (INSPECT ($ to1))
{WINDOW}#47,125470
```

```
All Values of TextObject ($ to1).
font #,(Defer (FONTCREATE (QUOTE
```

```
37← (@ ($ to1) font)
{FONTDESCRIPTOR}#74,167334
```

```
All Values of TextObject ($ to1).
font {FONTDESCRIPTOR}#74,167334
```

### 8.2.3 InheritingAV

This specialization of **ActiveValue** is used as a mixin to add the **InheritedValue** method.

#### InheritingAV

[Class]

- Purpose: Used as a mixin to add the **InheritedValue** method.
- Behavior: An abstract class, adds a method **InheritedValue** which allows looking at the value a variable would have if it had no local value, as **NotSetValue** would work. Used as a mixin to add this capability to other specializations of **ActiveValue**.
- Example: Used as super class of **AppendSuperValue** to provide incremental menus in various parts of LOOPS.

(← self **InheritedValue** containingObj varName propName type)

[Method of InheritingAV]

- Purpose/Behavior: Allows viewing the value a variable would have inherited if it had no local value yet assigned. Similar to the way **NotSetValue** works, it is removed by an assignment to the variable.
- Arguments: *self* **InheritingAV** instance.
- containingObj* The instance or class that contains the variable to be viewed.
- varName* In the *containingObj* the variable to be viewed.
- propName* Name of an instance variable or class variable property to be looked at. If *propName* is NIL, the variable itself is viewed.

*type* One of IV, CV, or NIL: a *type* of IV or NIL indicates that the variable is an instance variable or an instance variable property of *containingObj*; a *type* of CV indicates a class variable or class variable property of *containingObj*.

Returns: The value which would have been inherited if the variable had no local value.

### 8.2.4 ReplaceMeAV

This specialization of the class **ActiveValue** sets up the functionality to replace itself on the first Put- access.

#### ReplaceMeAV

[Class]

- Purpose: Specializes the method **PutWrappedValue** to simply replace itself on the first Put- request.
- Behavior: No variables are defined in this class. It is an abstract class not intended for instantiation. It is a mixin (see Chapter 3, Classes) to be combined in specialization with another class to add its functionality to the subclass.
- Example: **FirstFetchAV** combines **LocalStateActiveValue** and **ReplaceMeAV** to get an **ActiveValue** that replaces itself with the value of an expression stored in the instance variable **localState**.

### 8.2.5 NotSetValue

This section describes where and when instances of this class appear in user-defined objects.

#### CAUTION

Do not specialize the classes **NotSetValue** and **NestedNotSetValue**. The documentation is provided here only to explain the functionality that these classes provide to the LOOPS system.

#### NotSetValue

[Class]

- Purpose: This specialization of the class **ActiveValue** is unique in that it was created primarily for implementing instance variable inheritance. It has no instance variable to hold a local value and is replaced after the first Put- variable access.
- Behavior: When an instance of any LOOPS object is created, its instance variables are initialized to contain the value of the variable **NotSetValue**. **NotSetValue** is an annotatedValue whose **ActiveValue** is the only instance of the class **NotSetValue**. The value of **NotSetValue** stored in an instance variable may be replaced within other initialization procedures of new instances that are invoked by the methods **NewWithValues** and **NewInstance** and the instance variable property **:initForm**.
- The class **NotSetValue** specializes the default **ActiveValue** protocol to trigger instance variable inheritance. An annotatedValue check is always done by **GetValue** and **PutValue**. LOOPS speeds up instance generation by always initializing instance variables to the value **NotSetValue**. If a retrieval attempt is made on the variable, **NotSetValue** finds the inherited value and returns that value. If no requests are made for the value of the variable, there is no overhead for the instance variable.

The term local value refers to the values LOOPS has actually written into that instance's instance variables. The local value is always equal to **NotSetValue** before the first Put- access, and to a new value after the first Put- access.

The annotatedValue **#,NotSetValue** is bound to the Lisp variable **NotSetValue**. It must always be on the inside of any sequence of nested **ActiveValues**. Its **WrappingPrecedence** method returns NIL, ensuring this functionality. **NotSetValue** has no **localState** instance variable to hold any nested **ActiveValues**.

See Section 8.3.4, "Shared ActiveValues in Variable Inheritance," for information on **ActiveValues** as default values.

Example: Consider the class **Datum** with the instance variable **idNumber**. Create a new instance named **Datum2**. A standard **GetValue** or **@** call returns the default value of **idNumber**, since nothing else has yet been assigned. The call **GetIVHere** shows that the value is not stored in the instance, but is actually returned by **NotSetValue**.

```
91←(← ($ Datum) New 'Datum2)
#, ($ Datum2)

92←(@ ($ Datum2) idNumber)
NIL

93←(GetIVHere ($ Datum2) 'idNumber)
#,NotSetValue
```

8.2.5.1 NestedNotSetValue

---

This subclass of the class **NotSetValue** is used by the internal of LOOPS to solve the problem of using active values as default values.

8.2.6 User Specializations of Active Values

---

If new specializations of the class **ActiveValue** are defined, the methods **GetWrappedValueOnly** and **PutWrappedValueOnly** might need to be specialized (LOOPS-defined specializations of **ActiveValue**, such as **LocalStateActiveValue**, have already done this). You may also want to specialize the following methods:

|                           |                                                                             |
|---------------------------|-----------------------------------------------------------------------------|
| <b>AVPrintSource</b>      | Prints an <b>ActiveValue</b> instance.                                      |
| <b>GetWrappedValue</b>    | Method associated with getting an <b>ActiveValue</b> .                      |
| <b>PutWrappedValue</b>    | Method associated with putting an <b>ActiveValue</b> .                      |
| <b>WrappingPrecedence</b> | Returns T, NIL, or a number to specify order of <b>ActiveValue</b> nesting. |
| <b>CopyActiveValue</b>    | Copies an annotatedValue and its wrapped <b>ActiveValue</b> .               |

8.3 ACTIVE VALUE METHODS

---

8.3 ACTIVE VALUE METHODS

---

8.3 Active Value Methods

---

Methods defined in the class **ActiveValue** describe how active values work.

### 8.3.1 Adding and Deleting Active Values

This section describes the methods to install, delete, and replace active values.

| Name                      | Type   | Description                                                    |
|---------------------------|--------|----------------------------------------------------------------|
| <b>AddActiveValue</b>     | Method | Makes a variable or property an active value.                  |
| <b>WrappingPrecedence</b> | Method | Returns a value which determines how to nest the active value. |
| <b>DeleteActiveValue</b>  | Method | Deletes an active value.                                       |
| <b>ReplaceActiveValue</b> | Method | Replaces an active value.                                      |

(← *self* **AddActiveValue** *containingObj* *varName* *propName* *type* *annotatedValue*) [Method of **ActiveValue**]

**Purpose:** Accomplishes two tasks fundamental in making a variable or property an active value. First, the **ActiveValue** is wrapped inside an *annotatedValue*. Second, the *annotatedValue* is placed as the value of the variable.

**Behavior:** **AddActiveValue** associates the *annotatedValue* with the variable specified by the arguments. If the argument *annotatedValue* is not specified or is NIL, a new *annotatedValue* is created containing the **ActiveValue** *self*. When the current value of the variable is already an *annotatedValue*, the **WrappingPrecedence** message determines if it should be nested in the current one or wrapped around it.

**Arguments:** *self* **ActiveValue** instance.

*containingObj*

The instance or class that contains the variable where the **ActiveValue** is to be added.

*varName* In the *containingObj* the variable to be made into an **ActiveValue**.

*propName* Name of an instance variable or class variable property to be made into an **ActiveValue**. If *propName* is NIL, the **ActiveValue** is associated with the variable itself.

*type* One of IV, CV, CLASS, METHOD, or NIL.

- A *type* of IV or NIL indicates that *varName* is an instance variable or an instance variable property of *containingObj*.
- A *type* of CV indicates a class variable or class variable property of *containingObj*.
- A *type* of CLASS indicates access to a class object's instance variables and properties.
- A *type* of METHOD indicates access to a method object's instance variables and properties.

*annotatedValue*

**AnnotatedValue** object used to contain this **ActiveValue**. If NIL, a new **annotatedValue** is created.

**Returns:** *annotatedValue*

**Example:** Adds the **ActiveValue** instance named (**\$ ActiveValueInstance**) to the object (**\$ ExampleLoopsWindowInstance**) in the instance variable **width**.



```
71←(← ($ ActiveValueInstance) AddActiveValue ($
ExampleLoopsWindowInstance) 'width)

#,$AV IndirectVariable (ActiveValueInstance (NCV0.0X:.SD7.KR . 8))
(object #,$ ExampleLoopsWindowInstance) (varName width))
```

---

**(← self WrappingPrecedence)****[Method of ActiveValue]**

**Purpose:** Returns a value which determines how to nest the **ActiveValue** associated with *self*.

**Behavior:** Varies according to the value returned.

- T

The **ActiveValue** associated with *self* goes on the outside of any other active values.

- NIL

This **ActiveValue** goes on the inside.

If two **ActiveValues** return either T or NIL, a break occurs.

- Number

Specifies precedence: **ActiveValues** with larger **WrappingPrecedence** values go outside ones with smaller **WrappingPrecedence** values.

---

**CAUTION**

It is potentially dangerous to have more than one class with a T or NIL precedence.

---

**ActiveValues** that have the instance variable **localState** nest in the following way. When a new **ActiveValue** is added to an existing one with equal **WrappingPrecedence**, the original **ActiveValue** is held in the **localState** of the new one. **ActiveValues** not having an instance variable **localState** must nest inside of ones that do.

To set the **WrappingPrecedence** for a user specialization of **ActiveValue**, specialize this method to return the proper value.

**Arguments:** *self*            **ActiveValue** instance.

**Returns:** The default method defined in the class **ActiveValue** returns 100. **WrappingPrecedence** for the class **NotSetValue** returns NIL. **WrappingPrecedence** for **IndirectVariable** returns 50.

---

**(← self DeleteActiveValue containingObj varName propName type)****[Method of ActiveValue]**

**Purpose:** Deletes an **ActiveValue** from *containingObj*.

**Behavior:** If the variable defined by the arguments is an **ActiveValue**, it is deleted. If it contains a nested **ActiveValue**, the one matching *self* is deleted; otherwise, nothing happens. No **ActiveValue** messages are triggered. If the deleted **ActiveValue** had a **localState**, it becomes the current value.

**Arguments:** *self*            **ActiveValue** instance.

*containingObj*

The instance or class that contains the variable where the **ActiveValue** is found.

*varName*

In the *containingObj* the variable that contains the **ActiveValue**.

*propName*

Name of an instance variable or class variable property where the **ActiveValue** resides. If *propName* is NIL, the **ActiveValue** is associated with the variable itself.

*type*

One of IV, CV, CLASS, METHOD, or NIL.

- A *type* of IV or NIL indicates that *varName* is an instance variable or an instance variable property of *containingObj*.
- A *type* of CV indicates a class variable or class variable property of *containingObj*.
- A *type* of CLASS indicates access to a class object's instance variables and properties.
- A *type* of METHOD indicates access to a method object's instance variables and properties.

Returns: The deleted annotatedValue if a match was found, NIL otherwise.

(← **self ReplaceActiveValue newVal containingObj varName propName type**) [Method of ActiveValue]

Purpose: Replaces an **ActiveValue**.

Behavior: Replaces the **ActiveValue** *self* with *newVal*. The location of the old **ActiveValue** is described by the arguments. No **ActiveValue** messages are triggered.

Arguments:

*self* **ActiveValue** instance.

*newVal* The new value used to replace *self*.

*containingObj*

The instance or class that contains the variable where the **ActiveValue** is found.

*varName*

In the *containingObj* the variable that holds the **ActiveValue**.

*propName*

Name of an instance variable or class variable property where the **ActiveValue** resides. If *propName* is NIL, the **ActiveValue** is associated with the variable itself.

*type*

*type* is one of IV, CV, or NIL: a *type* of IV or NIL indicates that the variable is an instance variable or an instance variable property of *containingObj*; a *type* of CV indicates a class variable or class variable property of *containingObj*.

Returns: The value of *newVal*.

### 8.3.2 Fetching and Replacing Wrapped Values

The value of a variable is wrapped in an **ActiveValue**, usually by keeping it in the instance variable **localState**. Specify the behavior of new **ActiveValue** specializations by specializing the methods **GetWrappedValue** and **PutWrappedValue**. For example, **IndirectVariable.GetWrappedValue** just does a **GetValue** on the slot specified by its **object**, **varName**, **propName**, and **type** instance variables. These methods may perform arbitrary work before returning a value, usually that of **localState**. The methods **GetWrappedValueOnly** and **PutWrappedValueOnly** are available for accessing **localState** and bypassing the **ActiveValue** mechanism.

The following table shows the items in this section.

| Name                       | Type   | Description                                                                                                       |
|----------------------------|--------|-------------------------------------------------------------------------------------------------------------------|
| <b>GetWrappedValue</b>     | Method | Contains the code to be triggered by a Get- reference to the variable which has been made an <b>ActiveValue</b> . |
| <b>GetWrappedValueOnly</b> | Method | Provides a mechanism to assist in handling nested <b>ActiveValues</b> .                                           |
| <b>PutWrappedValue</b>     | Method | Contains the code to be triggered by a Put- reference to the variable which has been made an <b>ActiveValue</b> . |
| <b>PutWrappedValueOnly</b> | Method | Provides a mechanism to assist in handling nested <b>ActiveValues</b> .                                           |

(← *self* **GetWrappedValue** *containingObj* *varName* *propName* *type*) [Method of **ActiveValue**]

**Purpose:** Contains the code to be triggered by a Get- reference to the variable which has been made an **ActiveValue**.

**Behavior:** Performs arbitrary actions, but when finished, it must return a value which will be returned as the value of the **Get** to the original variable.

This method is fundamental for **ActiveValues**. When **GetValue** or **GetClassValue** finds an annotatedValue in an instance, it does not return that as the value. Instead, it sends the contained **ActiveValue** the **GetWrappedValue** message. This method is not usually called explicitly by users, but is triggered when the **GetValue** function retrieves the value of a variable that contains an **ActiveValue**. It should be specialized when a new subclass of **ActiveValue** is defined.

**Arguments:** *self* **ActiveValue** instance.

*containingObj*

The instance or class that contains the variable where the **ActiveValue** is found.

*varName* In the *containingObj* the variable that holds the **ActiveValue**.

*propName* Name of an instance variable or class variable property where the **ActiveValue** resides. If *propName* is NIL, the **ActiveValue** is associated with the variable itself.

*type* One of IV, CV, or NIL: a *type* of IV or NIL indicates that the variable is an instance variable or an instance variable property of *containingObj*; a *type* of CV indicates a class variable or class variable property of *containingObj*.

**Returns:** The value returned from the actions performed by **GetWrappedValue** message.

(← *self* **GetWrappedValueOnly**) [Method of **ActiveValue**]

**Purpose:** Enables the **ActiveValue** mechanism to deal with different problems of nested **ActiveValues**. You will generally not need to specialize this method, as most uses of **ActiveValues** will specialize a subclass of **ActiveValue**.

**Behavior:** Specializations of the class **ActiveValue** may need to specialize this method. (**LocalStateActiveValue**, **IndirectVariable**, and **NotSetValue** all have specialized versions of this method.)

The class **LocalStateActiveValue** specialization simply returns the value of *self*'s **localState** without triggering the active value mechanism.

The class **IndirectVariable** specialization simply returns the value of tracked variable without triggering the active value mechanism.

The class **NotSetValue** specialization simply returns the value of **NotSetValue**.

Arguments: *self* **ActiveValue** instance.

Returns: See Behavior.

(← *self PutWrappedValue containingObj varName newValue propName type*) [Method of ActiveValue]

Purpose: Contains the code to be triggered by a Put- reference to the variable which has been made an **ActiveValue**.

Behavior: The **PutWrappedValue** message is similar to **GetWrappedValue** except that it is triggered when the local state of the value is to be replaced. When **PutValue** or **PutClassValue** attempts to replace an **ActiveValue**, it instead sends the contained **ActiveValue** the **PutWrappedValue** message.

The default method found in the class **ActiveValue** checks for nested **ActiveValues** by sending the **GetWrappedValueOnly** message to *self*. If the result is an **AnnotatedValue**, **PutWrappedValue** forwards the message on the nested **ActiveValue**; otherwise it sends the message **PutWrappedValueOnly** to *self* and returns the result.

Arguments: *self* **ActiveValue** instance.

*containingObj*

The instance or class that contains the variable where the **ActiveValue** is found.

*varName* In the *containingObj* the variable that holds the **ActiveValue**.

*newValue* The value used to replace the **ActiveValue** containing *self*.

*propName* Name of an instance variable or class variable property where the **ActiveValue** resides. If *propName* is NIL, the **ActiveValue** is associated with the variable itself.

*type* One of IV, CV, or NIL: a *type* of IV or NIL indicates that the variable is an instance variable or an instance variable property of *containingObj*; a *type* of CV indicates a class variable or class variable property of *containingObj*.

Returns: The value of *newValue*.

(← *self PutWrappedValueOnly newValue*) [Method of ActiveValue]

Purpose: Enables the **ActiveValue** mechanism to deal with different problems of nested **ActiveValues**. You will generally not need to specialize this method, as most uses of **ActiveValues** will specialize a subclass of **ActiveValue**.

Behavior: Specializations of the class **ActiveValue** may need to specialize this method. (**LocalStateActiveValue**, **IndirectVariable**, and **NotSetValue** all have specialized versions of this method.)

The class **LocalStateActiveValue** specialization simply stores *newValue* into *self*'s **localState** without triggering the active value mechanism.

The class **IndirectVariable** specialization simply stores *newValue* into tracked variable without triggering the active value mechanism.

The class **NotSetValue** specialization causes a break.

Arguments:    *self*            **ActiveValue** instance.  
                 *newValue*    The new value for **localState**.  
Returns:      See Behavior.

### 8.3.3 Get and Put Functions Bypassing the ActiveValue Mechanism

---

**ActiveValues** normally convert **GetValue**, **GetClassValue**, **PutValue**, and **PutClassValue** accesses into messages which invoke methods to return a value, usually from the **localState** instance variable of the **ActiveValue**. The following functions allow access to class variables and instance variables without triggering any installed **ActiveValue**. See Chapter 2, Instances, for details.

| Name                     | Type     | Description                                                                                                                                                                                                                     |
|--------------------------|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>GetValueOnly</b>      | Function | Finds the value of an instance variable without triggering active values.                                                                                                                                                       |
| <b>PutValueOnly</b>      | Function | Writes the value of an instance variable without triggering active values.                                                                                                                                                      |
| <b>GetClassValueOnly</b> | Function | Returns the value of a class variable; does not trigger active values.                                                                                                                                                          |
| <b>PutClassValueOnly</b> | Function | Changes the value of a class variable and changes the value of a class variable. The change occurs within the class and therefore causes all instances to access the new value of the variable. Does not trigger active values. |

### 8.3.4 Shared ActiveValues in Variable Inheritance

---

When a **LocalStateActiveValue** is used as the default value for an instance variable in a class, it must be copied into each instance or else all of the instances try to share a single **localState**. This copying is done automatically by LOOPS when the instance variable is first accessed, which means that all instances will share the same **ActiveValue** until that first access. Copying an **ActiveValue** implies creating a new **annotatedValue**, so it must be done with the specialized method **CopyActiveValue**.

**ActiveValues** with no local state may be shared by several variables. In the most extreme case, one instance of **NotSetValue** is the default for the instance variables of all new instances of all classes.

---

(← *self* **CopyActiveValue** *annotatedValue*) [Method of ActiveValue]

---

Purpose:        Makes a copy of an **ActiveValue** instance.

Behavior:      Copies the *AnnotatedValue* and the wrapped **ActiveValue** handling instance variables as follows:

- Instance variables that contain **AnnotatedValues** are copied using the **CopyActiveValue** method.
- The instance variable **localState** is copied so that each copy has its own unique local state.
- All other instance variables are considered shared, and are not copied.

Arguments:    *self*            **ActiveValue** instance.

*annotatedValue*

The annotatedValue that surrounds *self*.

Returns: A new annotatedValue wrapped around a copy of the **ActiveValue** *self*.

### 8.3.5 Creating Your Own Active Value

This example defines a new kind of active value, a **BlippingActiveValue**, which prints out a "blip" of some kind whenever the variable it wraps is read or written.

First, define the new class as a specialization of **LocalStateActiveValue**, then specialize the **PutWrappedValue** and **GetWrappedValue** methods. This is done with the display editor, so in the example they are printed out via the **PPMethod** method. In each case, a **PRINTOUT** function was added before the call to **←Super**.

Create an instance of **Window** for a location to install a **BlippingActiveValue** for the example. Line 38 is required to set the value of **height** locally to instance **Window1**; if this is not done, its initial value is the active value **#,NotSetValue**, which would remove any active value as soon as it was accessed.

The last few statements in the example show how read and write accesses to **height** cause a blip character to be printed before **height** is either read or written, with a "!" character representing a write access triggering **PutWrappedValue**, and a "." character representing a read access triggering **GetWrappedValue**.

```

32← (DefineClass 'BlippingActiveValue ' (LocalStateActiveValue]
#, ($ BlippingActiveValue)

33← (← ($ BlippingActiveValue) SpecializeMethod 'PutWrappedValue]
BlippingActiveValue.PutWrappedValue

34← (← ($ BlippingActiveValue) SpecializeMethod 'GetWrappedValue]
BlippingActiveValue.GetWrappedValue

35← (← ($ BlippingActiveValue) PPMethod 'PutWrappedValue]

(BlippingActiveValue.PutWrappedValue
 (Method ((BlippingActiveValue PutWrappedValue)
 self containingObj varName newValue propName type)
 COMMENT **COMMENT**
 (PRINTOUT PPDefault "!")
 (←Super self PutWrappedValue containingObj varName newValue
 propName type)))
 (BlippingActiveValue.PutWrappedValue)

36← (← ($ BlippingActiveValue) PPMethod 'GetWrappedValue]

(BlippingActiveValue.GetWrappedValue
 (Method ((BlippingActiveValue GetWrappedValue)
 self containingObj varName propName type)
 COMMENT **COMMENT**
 (PRINTOUT PPDefault ".")
 (←Super self GetWrappedValue containingObj varName propName
 type)))
 (BlippingActiveValue.GetWrappedValue)

37← (← ($ Window) New 'Window1]
#, ($ Window1)

38← (←@ ($ Window1) height 9876]

```

```
9876

39←(←New ($ BlippingActiveValue) AddActiveValue ($ Window1) 'height]
#,$(& BlippingActiveValue (46 . 45056))

40←(←@ ($ Window1) height 300)
!300

41←(@ ($ Window1) height]
.300

42←(FOR I TO 20 SUM (@ ($ Window1) height]
.....6000

43←(←@ ($ Window1) height 123]
!123

44←(FOR I TO 20 DO SUM (←@ ($ Window1) height I]
!!!!!!!!!!!!!!!!!!!!!!!!!!!!210

45←
```

---

## 8.4 Annotated Values

---

AnnotatedValue is a LOOPS pseudoclass, and instances of it, called pseudoinstances, are Interlisp data type instances.

The structure of the data type is simple. Each annotatedValue contains one field named annotatedValue. This field contains an **ActiveValue** object. The Interlisp record package macros discussed below let you create and work with instances of the data type annotateValue.

There is also a LOOPS class named **AnnotatedValue**. It is an abstract class so it cannot be instantiated, but paradoxically there are objects which consider it their class. (Actually, it is not paradoxical, but this behavior is implemented at a low level within the LOOPS system.) These are the Lisp data type annotatedValue. In normal use this class can be ignored.

---

|                       |         |
|-----------------------|---------|
| <b>AnnotatedValue</b> | [Class] |
|-----------------------|---------|

---

- |           |                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:  | LOOPS class equivalent of Lisp data type annotatedValue.                                                                                                                                                                                                                                                                                                                                              |
| Behavior: | This is a LOOPS class, but not a subclass of <b>Object</b> . Its super is the LOOPS class <b>Tofu</b> . (See Chapter 4, Metaclasses, for a description of <b>Tofu</b> .) <b>AnnotatedValue</b> is a LOOPS abstract class, and instances are Interlisp data type instances. LOOPS fields messages sent to the annotatedValue data type instances by using the class definition <b>AnnotatedValue</b> . |

---

### 8.4.1 Explicit Control over Annotated Values

---

This section describes the macros and methods that allow explicit control over annotated values.

| Name         | Type  | Description                                                                 |
|--------------|-------|-----------------------------------------------------------------------------|
| <b>type?</b> | Macro | Performs a type check for an instance of the Lisp data type annotatedValue. |

|                             |        |                                                                                                             |
|-----------------------------|--------|-------------------------------------------------------------------------------------------------------------|
| <b>create</b>               | Macro  | Creates a new instance of the data type <code>annotatedValue</code> .                                       |
| <b>fetch</b>                | Macro  | Retrieves the contents of the <code>annotatedValue</code> field of an <code>annotatedValue</code> instance. |
| <b>replace</b>              | Macro  | Replaces contents of the <code>annotatedValue</code> field of the <code>annotatedValue</code> instance.     |
| <b>_AV</b>                  | Macro  | Sends a message to the <b>ActiveValue</b> object wrapped in an <code>annotatedValue</code> .                |
| <b>MessageNotUnderstood</b> | Method | Forwards messages intended for the wrapped <b>ActiveValue</b> to that object.                               |

**(type? annotatedValue value)** [Macro]

Purpose: Performs a type check for an instance of the Lisp data type `annotatedValue`.

Arguments: *value* The value to type check.

Returns: T if *value* is an instance of the data type `annotatedValue`, NIL otherwise.

**(create annotatedValue annotatedValue ← object)** [Macro]

Purpose: Creates a new instance of the data type `annotatedValue`.

Arguments: *object* An **ActiveValue** object to initialize the field `annotatedValue` of the new `annotatedValue` instance. This must be an object that has a method **AVPrintSource** (a method of **ActiveValue**) or this form breaks on evaluation. No type checking of *object* will be performed by the macro.

Returns: An instance of `annotatedValue`.

**(fetch annotatedValue of value)** [Macro]

Purpose: Retrieves the contents of the `annotatedValue` field of an `annotatedValue` instance.

Arguments: *value* An `annotatedValue` instance.

Returns: Contents of field `annotatedValue`.

**(replace annotatedValue of value with object)** [Macro]

Purpose: Replaces contents of the `annotatedValue` field of `annotatedValue` instance with *object*.

Arguments: *value* An `annotatedValue` instance.

*object* **ActiveValue** object to be stored in the field. No type checking is done on *object*.

Returns: If *value* is not an `annotatedValue`, generates an error; otherwise the previous contents of the field is returned.

**(\_AV av selector . args)** [Macro]

Purpose: Sends a message to the **ActiveValue** object wrapped in an `annotatedValue`.



|            |                                                                                   |                                                                |
|------------|-----------------------------------------------------------------------------------|----------------------------------------------------------------|
| Behavior:  | Equivalent to<br><br><code>(_ (fetch annotatedValue of av) selector .args)</code> |                                                                |
| Arguments: | <i>av</i>                                                                         | Instance of an <code>annotatedValue</code> .                   |
|            | <i>selector</i>                                                                   | Selector for message to send to the <b>ActiveValue</b> object. |
|            | <i>args</i>                                                                       | Arguments to be passed when the message is sent.               |
| Returns:   | Result of message.                                                                |                                                                |

---

`(← self MessageNotUnderstood)` [Method of AnnotatedValue]

---

|           |                                                                                                                                                  |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:  | Forwards messages intended for the wrapped <b>ActiveValue</b> to that object.                                                                    |
| Behavior: | Messages sent to an <code>annotatedValue</code> are forwarded to its wrapped <b>ActiveValue</b> . Users should not explicitly send this message. |

## 8.4.2 Saving and Restoring Annotated Values

---

The following are methods of the class **ActiveValue** that handle annotated values.

---

`(← self AVPrintSource)` [Method of ActiveValue]

---

|           |                                                                                                                                             |
|-----------|---------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:  | Prints <b>ActiveValues</b> .                                                                                                                |
| Behavior: | An <code>annotatedValue</code> determines how it prints out by sending the <b>AVPrintSource</b> message to its wrapped <b>ActiveValue</b> . |

The default method in **ActiveValue** returns a list of the form:

```
("#," $AV className avNames(ivName value propName value ...) (ivName ...) ...)
```

which causes the `annotatedValue` to print out as

```
#, ($AV className avNames(ivName value propName value ...) (ivName ...) ...)
```

|            |                  |                                                                                                  |
|------------|------------------|--------------------------------------------------------------------------------------------------|
| Arguments: | <i>self</i>      | <b>ActiveValue</b> instance.                                                                     |
|            | <i>className</i> | Name of the class of the <b>ActiveValue</b> .                                                    |
|            | <i>avNames</i>   | List of names of <i>self</i> , the last element being the unique identifier (UID) of <i>self</i> |

The list *(ivName value propName value ...)* describes the state of the instance variables of the **ActiveValue**. Including the UID of the **ActiveValue** in the print form enables recovery of the identity of the **ActiveValue**. This enables different `annotatedValues` to share the same **ActiveValue**, and maintain this sharing across saving to a file and reloading into Lisp.

|          |                                                                                                                                                                                                                                                                                                                               |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Returns: | A form suitable for use by the Interlisp function <b>DEFPRINT</b> . Result should be a pair of the form (item1 . item2); item1 will be printed using <b>PRIN1</b> , and item2 will be printed using <b>PRIN2</b> (see <i>Lisp Release Notes</i> and the <i>Interlisp-D Reference Manual</i> description of <b>DEFPRINT</b> ). |
|----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|          |                                                                                                                                                     |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Example: | <pre>#, (\$AV IndirectVariable (HeightFromWidth (NCV0.0X:.SD7.KR . 8)) (object #.(\$ SquareWindow)) (varName width) (propName NIL) (type IV))</pre> |
|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------|

(\$AV className avNames . ivForms)

[NLambda, NoSpread Function]

Purpose: Reconstructs an annotatedValue that has been saved to a file.

Arguments: *className* Name of the class of **ActiveValue**.

*avNames* A list of the LOOPS names of **ActiveValue** instances.

*ivForms* A list describing the state of the instance variables of the **ActiveValue**.

Returns: A new annotatedValue whose **ActiveValue** is reconstructed from the *avNames* and *ivForms*.

## 8.5 ACTIVE VALUES IN CLASS STRUCTURES

### 8.5 ACTIVE VALUES IN CLASS STRUCTURES

## 8.5 Active Values in Class Structures

It is possible to have an active value as the default value of an instance variable or the value of a class variable in a class. For example, the following class has an active value installed in the class variable **dontChange** and one installed in the instance variable **firstRead**.

```
SEdit #,($C test) Package: INTERLISP
((MetaClass Class Edited%: ; Edited 2-Dec-87 12:59
)
(Supers Object)
(ClassVariables
(dontChange
#,$AV NoUpdatePermittedAV ((IDAW0.1Y:H53.J99) . 540)) (localState 100)))
(InstanceVariables (firstRead #,(Defer (DATE)))) (MethodFns))
```

**LocalStateActiveValue** active values as default IV values are copied down into the instance when their **localState** is smashed, instead of being shared by all instances; this is different from normal default behavior. It is also possible to create a **LocalStateActiveValue** which inherits its **localState** value by giving it a **localState** value of the value of **NotSetValue**. These copy the inherited value down from the superclass when the **LocalStateActiveValue** is created; if the value in the superclass is changed after the **LocalStateActiveValue** is created, that change will not be reflected in the **LocalStateActiveValue**. Normally inherited values are always tracked by instances that inherit them.

There are two ways to enter active values into the structure of a class: with the editor or programmatically.

It is possible to create active values by typing in a form such as:

(\$AV activeValueClassName NIL (ivname value propName value ...) (ivname value propName value ...) ...). None of the arguments are evaluated.

To add an active value through the editor, you can type in the above form, select it, and mutate it with the function **EVAL**.

Programmatically, you can use the functions **PutClassIV**, **PutClassValue**, **PutClassValueOnly**, **AddCIV**, **AddCV**, etc. or different methods, such as **Add**, to modify or add class variables and instance variables.

For example, given the above class, **test**:

```
(← ($ test) Add 'CV 'randomJustOnce ($AV FirstFetchAV NIL
(localState (RAND 0 1000))))
```

and

```
(AddCIV ($ test) 'newIV ($AV LocalStateActiveValue NIL
(localState (1 2 3))))
```

will result in the following:

```
SEdit #,($C test) Package: INTERLISP
((MetaClass Class Edited%: ; Edited 2-Dec-87 13:20
)
(Supers Object)
(ClassVariables
(dontChange
#,$AV NoUpdatePermittedAV (([DAW0.1Y:.H53.]99] . 540)) (localState 100)))
(randomJustOnce #,(Defer (RAND 0 1000))))
(InstanceVariables (firstRead #,(Defer (DATE)))
(newIV
#,$AV LocalStateActiveValue (([DAW0.1Y:.H53.]99] . 541)) (localState (1 2 3)))
(MethodFns))
```

An even more general programmatic method that more easily allows customization of an active value uses the `annotatedValue` data type explicitly. First, you must create an instance of an **ActiveValue** class.

```
(← ($ MyActiveValue) New 'MyAV1)
```

Then the contents of the instance **MyAV1** are initialized. Finally, it is added as the value of a variable in a class structure.

```
(AddCIV ($ test) 'myNewIV (create annotatedValue
annotatedValue ← ($ MyAV1)))
```

[This page intentionally left blank]

## 9. DATA TYPE PREDICATES AND ITERATIVE OPERATORS

This chapter describes data type predicates and an operator for iterative statements.

### 9.1 Data Type Predicates

Data type predicates test the Lisp data type of some datum. For example, some predicates test whether a datum is an object, instance, or class.

LOOPS defines three Lisp data types: `annotatedValue`, `class`, and `instance`. LOOPS provides predicates that enable testing aspects of these types.

| Name                   | Type   | Description                                                                                        |
|------------------------|--------|----------------------------------------------------------------------------------------------------|
| <b>Object?</b>         | Macro  | Determines if a particular datum is a LOOPS object.                                                |
| <b>Class?</b>          | Macro  | Determines if a particular datum is a class.                                                       |
| <b>Instance?</b>       | Macro  | Determines if a particular datum is an instance of a class.                                        |
| <b>AnnotatedValue?</b> | Macro  | Determines if a particular datum is an instance of the <code>annotatedValue</code> Lisp data type. |
| <b>Understands</b>     | Method | Determines if an object will respond to a message.                                                 |

To determine if a particular datum has an instance variable, class variable, or a property, use **HasIV**, **HasIV!**, or **HasCV** (see Chapter 2, Instances, and Chapter 3, Classes). To determine if a particular datum is an instance of a class or its superclasses, use **InstOf** or **InstOf!** (see Chapter 2, Instances).

**(Object? X)**

[Macro]

Purpose/Behavior: Determines if `X` is a LOOPS object. **Object?** returns `T` for both instances and classes.

Arguments: `X` Possible object.

Returns: Returns `T` if a name is a pointer to a LOOPS object, and returns `NIL` otherwise.

Example: This example demonstrates the use of **Object?**.

```
3←(← ($ Window) New 'Window1)
#,($& Window (|OZW0.1Y:.;h.Qm:| . 495))

4←(Object? ($ Window1))
T
```

```
5←(Object? ($ Window))
T

6←(Object? ($ NotAnObject))
NIL

7←(Object? 'NotAnObject)
NIL
```

---

**(Class? X)****[Macro]**

**Purpose/Behavior:** Determines if *X* is a class.

**Arguments:** *X* Possible class.

**Returns:** Returns T if *X* is a class; returns NIL otherwise.

**Example:** This example demonstrates the use of the predicate **Class?** Since **(\$ Window)** is a class, the function returns T. Since **Window1** and **NotClass** are not class names, NIL is returned. (Class? X) is equivalent to (type? class X).

```
8←(Class? ($ Window))
T

9←(Class? ($ NotClass))
NIL

10←(Class? ($ Window1))
NIL
```

---

**(Instance? X)****[Macro]**

**Purpose/Behavior:** Determines if *X* is an instance of some class.

**Arguments:** *X* Possible instance.

**Returns:** Returns T if *X* is an instance; returns NIL otherwise.

**Example:** This example shows the use of **Instance?** (Instance? X) is equivalent to (type? instance X).

```
11←(Instance? ($ Window1))
T

12←(Instance? 'Unbound)
NIL

13←(Instance? ($ Window))
NIL
```

---

**(AnnotatedValue? X)****[Macro]**

**Purpose/Behavior:** Determines if *X* is an instance of the annotatedValue data type. For a complete explanation of annotated values, see Chapter 8, Active Values.

**Arguments:** *X* Possible annotatedValue.

**Returns:** Returns T if *X* is an annotated value; returns NIL otherwise.

**Example:** Instances of class Window are created with an active value in the window instance variable. **AnnotatedValue** returns T for the annotatedValue which "wraps" an active value, not for the active value itself.

```

100←(← ($ Window) New 'Window3]
#, ($& Window (|OZW0.1Y:.;h.Qm:| . 495))

1←(GetValue ($ Window3) 'window)
{WINDOW}#51,140000

2←(GetValueOnly ($ Window3) 'window)
#,$AV LispWindowAV ((|OZW0.1Y:.;h.Qm:| . 495))
(localState {WINDOW}#51,140000))

3←(AnnotatedValue? (GetValueOnly ($ Window3) 'window))
T

4←(AnnotatedValue? (GetValue ($ Window3) 'window))
NIL

5←(AnnotatedValue? (_ ($ LispWindowAV) New 'LWAV4]
NIL

```

**(← self Understands selector)**

[Method of Object]

**Purpose/Behavior:** Determines if the object *self* will respond to a message with *selector*.

**Arguments:** *self* Instance or class in question.  
*selector* Selector in question.

**Returns:** T if *self* is a class or an instance of a class that understands message selector; NIL otherwise.

**Note:** If *self* is not a LOOPS object, you get NIL and not an error.

**Categories:** Object

**Example:** Given that **Window** is a class, **MyWindow** is an instance, and **SpinAround** is a method of **MyWindow**, **Window** returns NIL, and **MyWindow** returns T. Since **Shape** is a method of **Window**, this also returns T.

```

90←(← ($ Window) Understands 'SpinAround)
NIL

91←(← ($ MyWindow) Understands 'SpinAround)
T

91←(← ($ MyWindow) Understands 'Shape)
T

```

## 9.2 ITERATIVE OPERATORS

## 9.2 ITERATIVE OPERATORS

---

## 9.2 Iterative Operators

---

LOOPS provides an iterative operator to be used with Interlisp-D iterative statements.

**in-supers-of X**

[Iterative Statement Operator]

**Purpose / Behavior:** Allows iteration up the supers chain of the object X. Used in an Interlisp-D iterative statement. (See the *Interlisp-D Reference Manual* for more information on iterative statements.)

Arguments:    X                   A LOOPS class or an instance.

Example:      This example shows one way to use this operator.

```
55←(FOR I in-supers-of ($ ClassBrowser) DO (PRINT (← I ClassName]
ClassBrowser
IndexedObject
LatticeBrowser
Window
Object
Tofu
NIL
```



[This page intentionally left blank]

As described in Chapter 1, Introduction, one of the key components in the LOOPS system is inheritance, in which structures have well-defined relationships to other structures. Class inheritance is a typical example of this relationship.

Since inheritance can be described by a two-dimensional graph, it is natural to create a user interface for LOOPS built on the Lisp Library Module, Grapher. This user interface is called a browser. Browsers are tools to assist in the development cycle of a product or vehicles for building user interfaces within a final product.

Much development time is spent building, examining, and modifying the relationships between classes. These tasks include specifying the contents of various classes: class variables, instance variables, properties, and methods. The location of the class within the inheritance structure must also be determined. After a number of classes have been built, the relationships between the classes may need to be reviewed. Often, the initial design is flawed and requires the following changes:

- Moving parts of one class to another class.
- Adding, subtracting, or changing data or functionality within classes.
- Adding new classes, or merging different classes.

Browsers are the facility within LOOPS which support these types of operations. This chapter describes how to use browsers both interactively with the mouse, and programmatically.

Browsers are most commonly used on the classes defined for an application. Many of the examples here browse objects which LOOPS uses internally; the functionality is exactly the same.

---

## 0.1 Types of Built-in Browsers 1

---

A number of different types of browsers are already built into LOOPS:

- Lattice browsers
- Class browsers
- File browsers
- Supers browsers
- Metaclass browsers

This section describes these browsers in detail.

### 10.1.1 Lattice Browsers

---

The most general class is called **LatticeBrowser**. Figure 10-1 shows a class inheritance lattice with the subclasses of **LatticeBrowser**.

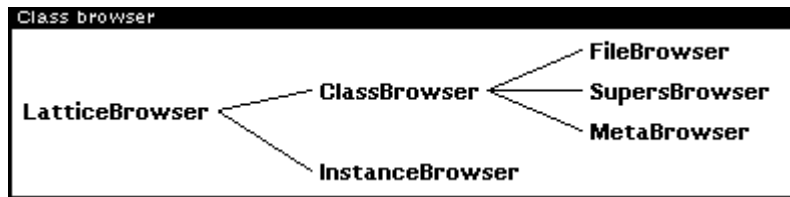


Figure 10-1. Sample Lattice Browser

### 10.1.2 Class Browsers

---

A class browser shows the linkages between a class or classes and their subclasses. Super classes are shown on the left (or top) side of the browser window. Subclasses of these are connected by links moving to the right (or down). An example of a class browser is shown in the previous section. The class **LatticeBrowser** is the root object of this example. Subclasses of **LatticeBrowser** are **ClassBrowser** and **InstanceBrowser**. Subclasses of **ClassBrowser** are **FileBrowser**, **SupersBrowser**, and **MetaBrowser**.

### 10.1.3 File Browsers

---

A file browser is a class browser containing all classes defined within a file. Additionally, file browsers contain a menu interface to common operations on files.

### 10.1.4 Supers Browsers

---

A supers browser is an inverted class browser. A class browser is built by following subclass links from a class. A supers browser is built by following superclass links from a class. An example of a supers browser is shown in Figure 10-2.

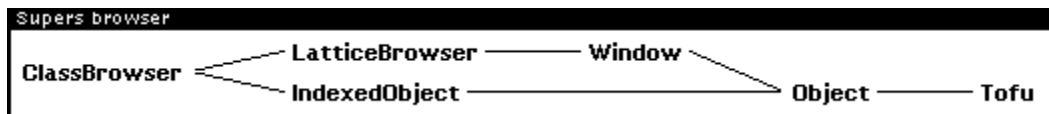


Figure 10-2. Sample Supers Browser

### 10.1.5 Metaclass Browsers

---

A metaclass browser is like a supers browsers, but is built by following metaclass links. Figure 10-3 shows two root classes: **ActiveValue** and **ClassBrowser**.

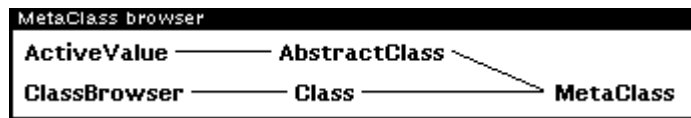


Figure 10-3. Sample Metaclass Browser

### 10.1.6 Instance Browsers

An instance browser shows the relationships between instances. These relationships may be more dynamic than the inheritance relationships shown by the class browsers. Typically, the relationships are not defined until runtime, and may be changed often. By specializing the instance browser, you can show several relationships between a fixed set of objects.

10.2 OPENING BROWSERS

## 10.2 OPENING BROWSERS

## 10.2 Opening Browsers

A browser can be opened in several ways:

- Selecting a menu option from the Background Menu.
- Selecting a menu option from the LOOPS icon.
- Sending a **Browse** message to an instance of a browser.
- Calling either of the functions **Browse** or **FileBrowse**.

### 10.2.1 Using Menu Options to Open Browsers

Since browsers are an important part of LOOPS, you can use menus in several ways to create standard browsers. Once opened, via menu or program, any browser can be operated from both the appropriate menus and programmatic commands.

#### 10.2.1.1 Overview of Background Menu and LOOPS Icon

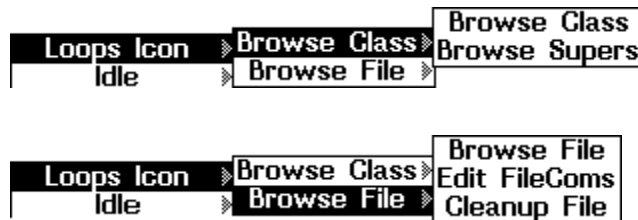
When LOOPS is loaded, the option **Loops Icon** is added to the background menu, as shown in this window:



The **Loops Icon** option has two suboptions:

- **Browse Class**
- **Browse File**

These suboptions are shown in the following windows:



Selecting **Loops Icon** puts a LOOPS icon on the screen; you are prompted to place the icon after it is created. The other commands are discussed in Section 10.2.1.2, "Command Summary."

The LOOPS icon, which appears in Figure 10-4, is a prototype instance of the class **LoopsIcon**. It is provided to give you another convenient menu interface to typical programming operations.



Figure 10-4. LOOPS Icon

Pressing the left button while the mouse is on the icon causes the following menu to appear with options appropriate for class browsers:



Pressing the middle button while the mouse is on the icon causes the following menu to appear with choices appropriate for file browsers:



Pressing the right button while the mouse is on the icon causes the following menu to appear with two options for operations on the icon itself:



**Close** removes the icon from the screen. It can be restored at any time from the background menu. **Move** lets the icon be moved to another location on the screen, just as any icon is moved in Lisp.

---

### 10.2.1.2 Command Summary

---

The background menu and the LOOPS icon provide the same functionalities. This section describes the commands available.

---

#### Browse Class, Browse Supers

---

Selecting either of these options causes the following prompt to appear in the prompt window.

Please tell me the name of the root object >

Enter the name of a class without using the "\$" notation. The system builds the appropriate type of browser and prompts you to move the window containing the browser.

### Browse File

Selecting this option causes the following menu to appear:



The top option on this menu is **\*newFile\*** which has three suboptions; the remaining options are the names of the files that are on **FILELST** (Lisp remembers what files are loaded and how they were loaded; **FILELST** files were loaded normally. See the *Interlisp-D Reference Manual* for a full explanation). Selecting one of the filenames will open a file browser on that file.

**\*newFile\*** Prompts you in the prompt window with the following prompt :

Please type in file name: >.

Enter the name of a file to create. The system checks to determine if a filecoms exists for that file name. If one exists, the system asks for confirmation before destroying the value of that filecoms and opening up an empty browser window. If no filecoms exists for that filename, an empty file browser window is opened.

**\*loadFile\*** Prompts you with:

Please type in file name to load: >

The system loads that file and opens a browser on it.

**\*hiddenFile\*** Causes a menu to appear with files that have been loaded but not **SYSLOADED** and are not on **FILELST**; that is, the files are on **LOADEDFILELST**, but not on **FILELST**. The **LOOPS** files, for example, the **.LCOMs** that add **LOOPS** to Lisp, are on this list.

### Edit Filecoms

Selecting this option brings up the same menu as the option **Browse File**. Instead of opening a browser on the file, a display editor window is opened on the filecoms of that file. If **\*newFile\*** is selected, you are prompted to enter a file name and an **SEdit** window is opened with a template containing the File Manager commands **CLASSES**, **METHODS**, **FNS**, **VARS**, and **INSTANCES**.

### CleanUp File

Selecting this option first calls **FILES?** and then builds a menu of files in **FILELST** that have changed. From this menu, select a file to be cleaned up; this calls **CLEANUP**.

## 10.2.2 Using Commands to Open Browsers

You can use the following methods and functions for opening browsers programmatically and from the Lisp Executive window.

| Name          | Type   | Description                                                |
|---------------|--------|------------------------------------------------------------|
| <b>Browse</b> | Method | Opens a browser showing the relationships between classes. |

|                   |          |                                                                      |
|-------------------|----------|----------------------------------------------------------------------|
| <b>BrowseFile</b> | Method   | Opens a browser showing the relationships between classes on a file. |
| <b>Browse</b>     | Function | Provides a short way to create a class browser.                      |
| <b>FileBrowse</b> | Function | Provides a short way to create a file browser.                       |

---

(← *self* **Browse** *browseList windowOrTitle goodList position*) [Method of LatticeBrowser]

---

Purpose/Behavior: Opens a browser showing inheritance relationships between classes.

Arguments: *self* An instance of **ClassBrowser**, **MetaBrowser**, or **SupersBrowser**.

*browseList* A LOOPS class name, a LOOPS pointer to a class name, or a list of those.

*windowOrTitle* A title to appear on the browser or a window to use (but which will be reshaped to fit the browser.) Title defaults to "Class browser."

*goodList* A **goodList** other than the *browseList*. (See Section 10.5.1, "Instance Variables of Class LatticeBrowser," for more information on **goodList**.)

*position* Lower left corner of browser. If NIL, position the window with the mouse.

Returns: Pointer to the browser object.

Examples: The following command opens a class browser on **Window**.

```
(←New ($ ClassBrowser) Browse 'Window)
```

The following command opens a supers browser on **InstanceBrowser** and **ClassBrowser**.

```
(←New ($ SupersBrowser) Browse (LIST 'InstanceBrowser ($ ClassBrowser)))
```

---

(← *self* **BrowseFile** *fileName*) [Method of FileBrowser]

---

Purpose: Opens a browser showing relationships between classes on a file.

Behavior: Classes defined within *fileName* are displayed within the browser. If *fileName* is NIL, a menu of files on **FILELST** opens. The selected file has a file browser opened on it.

Arguments: *self* An instance of the class **FileBrowser**

*fileName* File to browse; should not be a list.

Returns: *self*

Categories: FileBrowser

Example: The following command opens a file browser on the file **LoopsWindow**.

```
(←New ($ FileBrowser) BrowseFile 'LoopsWindow)
```

---

(**Browse** *classes title goodClasses position*) [Function]

---

|            |                                                                                                                                                                                                                                                                              |                                                                                                                                                             |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:   | Provides a short way to create a class browser.                                                                                                                                                                                                                              |                                                                                                                                                             |
| Behavior:  | Sends a <b>Browse</b> message to a new instance of a <b>ClassBrowser</b> passing <i>classes</i> , <i>title</i> , <i>goodClasses</i> , and <i>position</i> as arguments. If <i>goodClasses</i> is T, it is rebound to the value of <i>classes</i> before the message is sent. |                                                                                                                                                             |
| Arguments: | <i>classes</i>                                                                                                                                                                                                                                                               | A LOOPS class name, a LOOPS pointer to a class name, or a list of those.                                                                                    |
|            | <i>title</i>                                                                                                                                                                                                                                                                 | A title to appear on the browser. Title defaults to "Class browser."                                                                                        |
|            | <i>goodClasses</i>                                                                                                                                                                                                                                                           | A <b>goodList</b> other than <i>classes</i> . (See Section 10.5.1, "Instance Variables of Class LatticeBrowser," for more information on <b>goodList</b> .) |
|            | <i>position</i>                                                                                                                                                                                                                                                              | Lower left corner of browser. If NIL, position the window with the mouse.                                                                                   |
| Returns:   | A new instance of <b>ClassBrowser</b> .                                                                                                                                                                                                                                      |                                                                                                                                                             |
| Example:   | The following command creates a class browser on the class <b>ActiveValue</b> and all its subclasses.                                                                                                                                                                        |                                                                                                                                                             |
|            | <pre>11← (Browse 'ActiveValue)</pre>                                                                                                                                                                                                                                         |                                                                                                                                                             |

**(FileBrowse filename)**

[Function]

|            |                                                                                                                      |                 |
|------------|----------------------------------------------------------------------------------------------------------------------|-----------------|
| Purpose:   | Provides a short way to create a file browser.                                                                       |                 |
| Behavior:  | Sends a <b>BrowseFile</b> message to a new instance of a <b>FileBrowser</b> passing <i>filename</i> as the argument. |                 |
| Arguments: | <i>filename</i>                                                                                                      | File to browse. |
| Returns:   | New instance of <b>FileBrowser</b> .                                                                                 |                 |
| Example:   | The following command creates a file browser on the file <b>LoopsWindow</b> .                                        |                 |
|            | <pre>12← (FileBrowse 'LoopsWindow)</pre>                                                                             |                 |

## 10.3 USING CLASS BROWSERS, META BROWSERS, AND SUPERS BROWSERS

## 10.3 USING CLASS BROWSERS, META BROWSERS, AND SUPERS BROWSERS

## 10.3 Using Class Browsers, Meta Browsers, and Supers Browsers

Instances of **ClassBrowser**, **SupersBrowser**, **MetaBrowser** all have the same menu interface. This section shows examples of the various menus followed by descriptions of the actions performed after selecting particular options.

Three pop-up menus are associated with browsers:

- One menu appears by positioning the mouse on the title bar of the browser window and pressing either the left or the middle mouse button. This menu contains options that control the appearance of the browser.
- A second menu appears by positioning the mouse on one of the nodes in a browser and pressing the left mouse button. This menu contains informational options.



- A third menu appears by positioning the mouse on one of the nodes in a browser and pressing the middle mouse button. This menu contains editing options.

These menus differ depending on the browser type. The following sections describe the menus associated with class browsers, supers browsers, and metaclass (or, more simple, meta) browsers. Sections then describe the additional functionality associated with file browser menus.

### 10.3.1 Selecting Options in the Title Bar Menu

---

The following menu appears when you position the mouse on the title bar of the browser menu and press either the left or the middle mouse button:

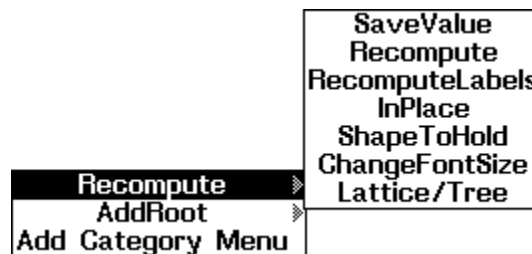


This section describes each menu option.

#### 10.3.1.1 Recompute and its Suboptions

---

Selecting the **Recompute** option and dragging the mouse to the right causes the following submenu to appear:



Most of the **Recompute** suboptions change the appearance of a browser but not its contents. For example, **SaveValue** provides a pointer to the browser without changing anything in it.

|                        |                                                                                                                                                                                                                                                                         |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SaveValue</b>       | The browser instance is stored in the instance variable <b>savedValue</b> of the prototype instance of <b>LoopsIcon</b> and in the value of <b>IT</b> (see the <i>Interlisp-D Reference Manual</i> ). This value is returned from the function call <b>SavedValue</b> . |
| <b>Recompute</b>       | Recomputes the entire browser structure from the starting objects. It does not recompute the labels for each item if those labels have been cached in the property <b>objectLabels</b> of the instance variable <b>menus</b> .                                          |
| <b>RecomputeLabels</b> | Recomputes the entire browser structure from the starting objects and recomputes the labels for each item.                                                                                                                                                              |
| <b>InPlace</b>         | Recomputes the browser without affecting the scrolled location of the lattice within the window. This may be necessary for a browser containing a large lattice structure.                                                                                              |
| <b>ShapeToHold</b>     | Makes the window for the browser just large enough to hold all of the nodes in the browser, up to a maximum size. Browser windows may also be changed interactively or programmatically with <b>SHAPEW</b> .                                                            |
| <b>ChangeFontSize</b>  | Causes a menu to appear containing 8, 10, 12, and 16. Selecting one changes the font size used to display the nodes to that value. The font family is                                                                                                                   |

```
(@ self browseFont:,FontFamily)
```

The font face is

```
(@ self browseFont:,FontFace)
```

Note: An alternative way to change the font of a browser is to enter:

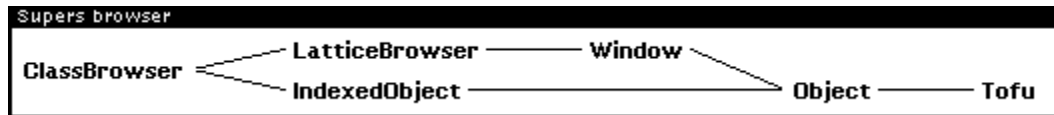
```
[PROGN (←@ ($ InstanceOfBrowser) browseFont
(FONTCREATE)) (← ($ InstanceOfBrowser)
RecomputeLabels)]
```

**Lattice/Tree** Causes the following menu to appear:

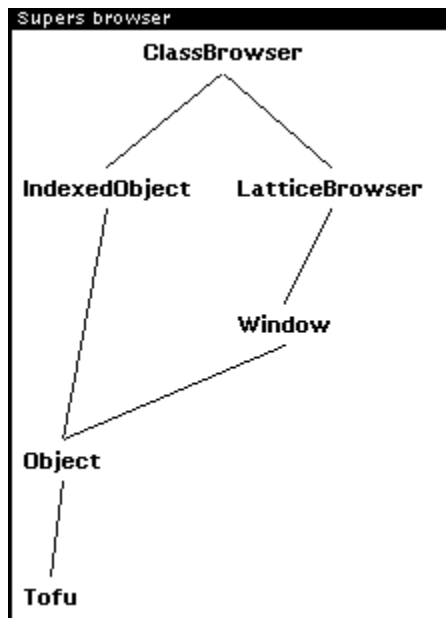
```
HORIZONTAL/LATTICE
VERTICAL/LATTICE
HORIZONTAL/TREE
VERTICAL/TREE
```

Using the example of a supers browser for the class **ClassBrowser**, this browser is drawn for each of the formatting options. A tree does not show branches recombining; a lattice does. A boxed node in a tree indicates the node shows up in more than one location in a tree. When a browser is constructed by the system the default formatting style is HORIZONTAL/LATTICE.

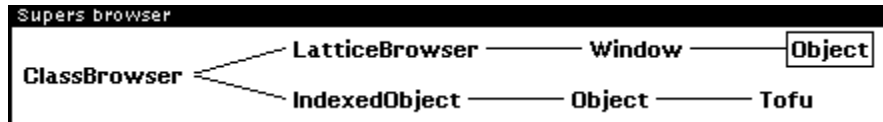
- HORIZONTAL/LATTICE



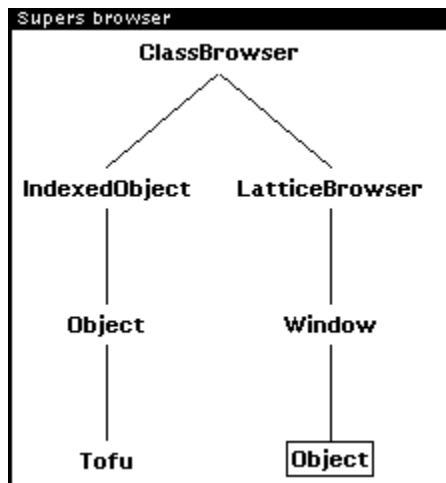
- VERTICAL/LATTICE



- HORIZONTAL/TREE



- VERTICAL/TREE



### 10.3.1.2 AddRoot and its Suboptions

The **AddRoot** options add items or subtrees to the browser.

Selecting the **AddRoot** option and dragging the mouse to the right causes the following submenu to appear:



#### AddRoot

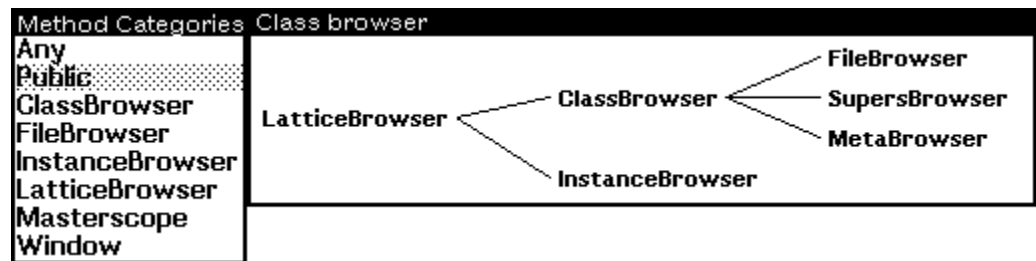
A prompt appears in an attached window to enter the name of a class to be added to the browser. If the entered item is not an object, a message that nothing was added to the browser is printed. If the entered item is already in the browser, nothing occurs. If the entered name does not correspond to a class, nothing occurs.

#### RemoveFromBadList

Objects within a browser can be put on the instance variable **badList**. This can be done by positioning the mouse on the node in a browser, pressing the left mouse button, and selecting an option from the menu that appears. Items on the **badList** are not displayed in the browser. If you select the option **RemoveFromBadList**, a menu appears showing any objects on the **badList**. Selecting one of those objects removes it from the **badList** and causes it to be redisplayed in the browser.

### 10.3.1.3 Add Category Menu

The system searches all methods in all classes shown in the browser and computes the categories for these. These categories are made into a sorted menu with the categories **Any** and **Public** included at the top. This menu is attached to the left side of the browser. Selecting options in this menu acts as a toggle, either highlighting them or returning them to their normal display.

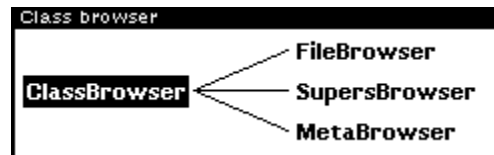


Selected options are stored on the browser instance variable **viewingCategories**. Options on this menu interact with the browser interface for editing methods as described in Section 10.3.2, "Selecting Options in the Left Menu."

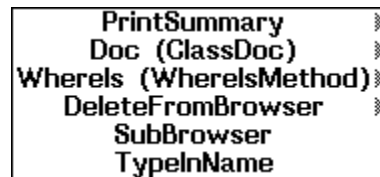
Note: Very often when using a browser, you ask to see what items a class inherits from classes above it in the inheritance lattice. To keep this inherited information more manageable, information inherited from the classes **Tofu**, **Object**, and **Class** are filtered out from the information presented to you. As an example, see the description of **PP** in the following section.

### 10.3.2 Selecting Options in the Left Menu

When the mouse is inside a browser and you hold down the left mouse button, nodes within the browser become inverted when the cursor moves over them, as shown in the following window:



If you release the left mouse button while the cursor is over a node, the following menu appears:



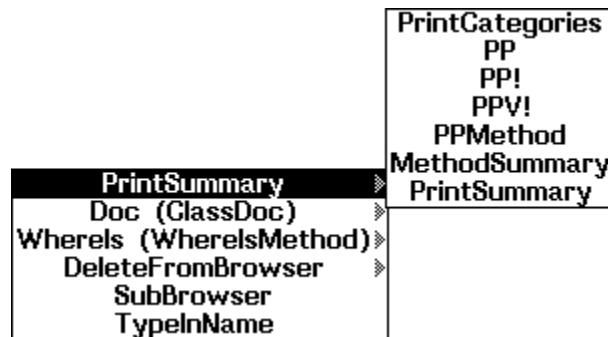
The options shown on the menu operate on the node (class) selected. Several of these options have associated submenus. Common options are in the main menu, and less common ones are menu suboptions. The actions that occur as a result of selecting one of these options are described in the following subsections. An additional subsection describes extended functionality available with the left mouse button.

### 10.3.2.1 PrintSummary and its Suboptions

---

**PrintSummary** provides a quick way to see object and method definitions. For all printing that occurs as a result of selecting this option or one of its suboptions, the output is sent to the value of the variable **PPDefault**, which is by default the Common Lisp Executive Window.

Selecting the **PrintSummary** option and dragging the mouse to the right causes the following submenu to appear:



#### **PrintCategories**

Prints the categories and associated methods for the selected class, as shown in the following window:

```
#,($ DestroyedObject)
Object
 Destroy!
```

#### **PP**

Produces a standard **PrettyPrint** of class, as shown in the following window:

```
(DEFCLASS DestroyedObject
 (MetaClass Class Edited%:
 (* --))
 (Supers Object))
```

#### **PP!**

Produces a formatted **Print** of class, as shown in the following window:

```
#,($ DestroyedObject)
MetaClass and its Properties
 Class Edited: (*
TheCollaborators:
15-Oct-84 16:23)
Supers
 (Object Tofu)
Instance Variable Descriptions
Class Variables
Methods
 Destroy! DestroyedObject.Destroy!
 doc NIL args NIL
```

Information that is defined locally within the class is printed in the bold font. Inherited information is printed in the regular font. Inherited information from the classes **Object** and **Tofu** is not printed.

#### **PPV!**

Same as **PP!**, but does not include **Methods**, as shown in the following window:

```

#,$ DestroyedObject)
MetaClass and its Properties
 Class Edited: (*
TheCollaborators:
15-Oct-84 16:23)
Supers
 (Object Tofu)
Instance Variable Descriptions
Class Variables

```

**PPMethod**

Brings up a menu of the methods for this class followed by a list of the known categories. The menu is influenced by the shaded options on the **Method Categories** menu (see Section 10.3.1.3, "Add Category Menu"), whether or not it is opened. Selecting a category will include any methods under that category in the menu. After selecting one of the methods, the Lisp function for that method is prettyprinted.

For example, selecting **PPMethod** from the node **ClassBrowser** with the **Method Categories** menu as shown results in the following menu of methods:

```

PPMethod: ClassBrowser
AddRoot
BoxNode
CareAbout?
GetSubs
LeftShiftSelect
MessageFormForProcess
NewItem
** Any category **
** Public category **
** ClassBrowser category **

```

**MethodSummary**

Prints a summary of the methods defined for the class, as shown in the following window:

```

((Destroy! DestroyedObject.Destroy! args
 NIL doc NIL))

```

**PrintSummary**

Similar to **PP**, but default values or properties associated with variables and methods are not printed, as shown in the following window:

```

#,$ DestroyedObject)
Supers
 Object
IVs

CVs

Methods
 Destroy!

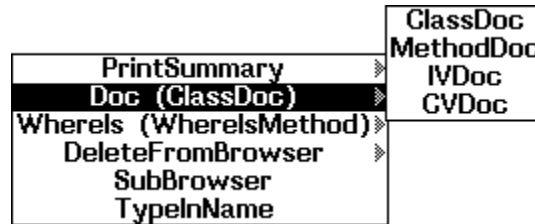
```

**10.3.2.2 Doc (ClassDoc) and its Suboptions**

Each part of an object's definition has a **doc** property containing strings. This menu option is a quick way to see the string for a specific part of a definition. For all printing that occurs as a result of selecting this option or one of its

suboptions, the output is sent to the value of the variable **PPDefault**, which is by default the Common Lisp Executive Window.

Selecting the **Doc (ClassDoc)** option and dragging the mouse to the right causes the following submenu to appear:



**ClassDoc** Prints documentation for class, that is, the **doc** property.

**MethodDoc** Causes a menu to appear in the same manner as **PPMethod** described in Section 10.3.2.1, "PrintSummary and its Suboptions." After selecting a method, information about that method is printed, as shown in the following window:

```
class: ClassBrowser selector: BoxNode
args: NIL
doc: NIL
```

The menu of methods reappears until a selection is made from outside the menu.

**IVDoc** Causes a menu to appear showing instance variables associated with the class. After selecting one, its documentation is printed, as shown in the following window:

```
ClassBrowser:menus:
Cache For Saved Menus. Will Cache Menus only
if value is T
```

The menu of instance variables reappears until a selection is made from outside the menu.

**CVDoc** Causes a menu to appear showing class variables associated with the class. After selecting one, its documentation is printed, as shown in the following window:

```
ClassBrowser::RightButtonItem:
Items to be done if Right button is selected
```

The menu of class variables reappears until no selection is made from the menu.

---

### 10.3.2.3 WhereIs and its Suboptions

**WhereIs** describes where each part of the object comes from. For all printing that occurs as a result of selecting this option or one of its suboptions, the output is sent to the value of the variable **PPDefault**, which is by default the Common Lisp Executive Window.

Selecting the **WhereIs** option and dragging the mouse to the right causes the following submenu to appear:

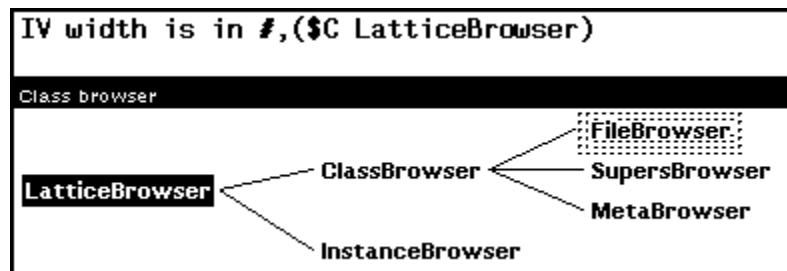
**WhereIsIV**

Causes a menu to appear at the current cursor position, showing the local and inherited instance variables. A sample menu appears here:

```
Finding IVS: ClassBrowser
LabelMaxCharsWidth
LabelMaxLines
badList
bottom
boxedNode
browseFont
goodList
graphFormat
height
lastSelectedObject
left
menus
showGraphFn
startingList
title
topAlign
viewingCategories
width
window
```

When you select an instance variable from this menu, these actions occur, as shown in the following window:

- A search starts with the selected class and then proceeds upwards through its supers. The first class that contains the selected instance variable flashes three times.
- All other classes in the browser that contain, that is, specialize, the instance variable are shaded.
- The name of the topmost class name is printed across the top of the window.

**WhereIsCV**

Same as above, but for class variables.

**WhereIsMethod**

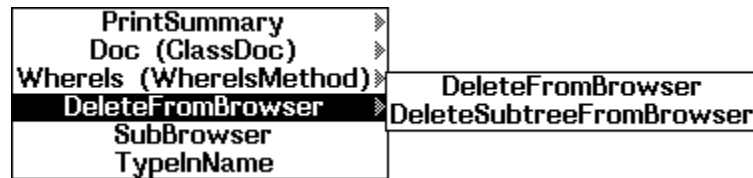
Same as above, but for **Methods**. The menu of methods is not filtered by any category information.



#### 10.3.2.4 DeleteFromBrowser and its Suboptions

---

Browsers show all of the lattice or tree from the root to the leaves. The **DeleteFromBrowser** option and its suboptions, shown here, allow you to prune the tree.



##### DeleteFromBrowser

Deleting a class from a browser adds it to that browser's instance variable **badList**. Items listed in a browser's **badList** are not displayed by the browser. If a class on the **badList** has subclasses, these are made into roots. To redisplay a class once it has been deleted, refer to the command **RemoveFromBadList** in Section 10.3.1.2, "AddRoot and its Suboptions."

##### DeleteSubtreeFromBrowser

Deleting a class with this command places this class on the browser's **badList**. Additionally, any subclasses of the deleted class are also placed on the **badList**. If one uses the command **RemoveFromBadList** to redisplay the deleted class, only that class is redisplayed. Its subclasses remain on the **badList** until they are explicitly removed from it.

#### 10.3.2.5 SubBrowser

---

The following window shows the selection of this option:



For class and supers browsers, the SubBrowser option opens a new browser of the same type (for example, a **SubBrowser** of a supers browser is a supers browser) with the class selected becoming the root object of the browser. For file browsers, the **SubBrowser** becomes a class browser.

#### 10.3.2.6 TypeInName

---

The following window shows the selection of this option:



This option puts the class name in the type-in buffer.

#### 10.3.2.7 Extending Functionality with the Left Mouse Button

---

Using various keys in conjunction with the left mouse button extends the available options.

- **SHIFT** key

Pressing the **SHIFT** key while selecting a node with the left button causes the name of the class to be typed into the current type-in point. If a node is not selected, but the cursor is in the background of the browser, the entire graph is copied. This can be used to insert browser images into TEdit documents, for example. See the Lisp Library documentation on Grapher for more details.

- **Control (CTRL)** key

Pressing the **CTR** key while selecting a node with the left button causes the node to track movement by the cursor. This allows you to temporarily change the layout of the nodes in the graph. The next update of the browser recomputes the node positions.

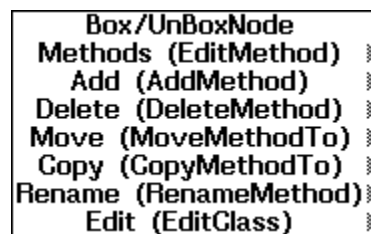
- **META** key

Pressing the **META** key while selecting a node with the left button is the same as a **PrintSummary** selection.

---

### 10.3.3 Selecting Options in the Middle Menu

When you position the cursor on a browser node and press the middle mouse button, the following menu appears:



This section describes the actions that occur when you select an option from this menu.

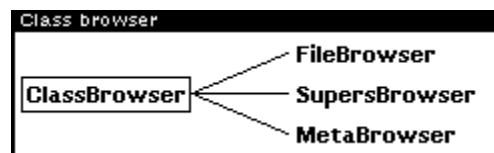
Except for **BoxNode**, all options are followed by a parenthetical suboption. This suboption appears in the option's submenu and performs the same operation as the option itself. For example, selecting **Methods (EditMethod)** performs the same operation as selecting **EditMethod** from its submenu.

The prompt for these options usually appears in a small window at the top of the browser, unless otherwise stated.

---

#### 10.3.3.1 Box/UnBoxNode

Draws a box around the node, as shown here:

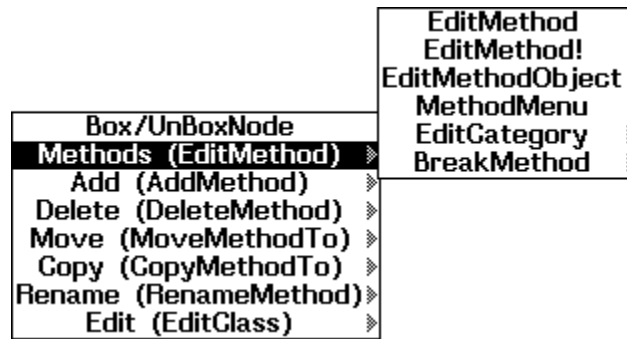


This also selects the node as a target for **Move** and **Copy** options. If a node is already boxed, **Box/UnBoxNode** unboxes it. Only one node in a class browser can be boxed at a time with this menu option.

---

#### 10.3.3.2 Methods (EditMethod) and its Suboptions

Selecting the **Methods (EditMethod)** option and dragging the mouse to the right causes the following submenu to appear:



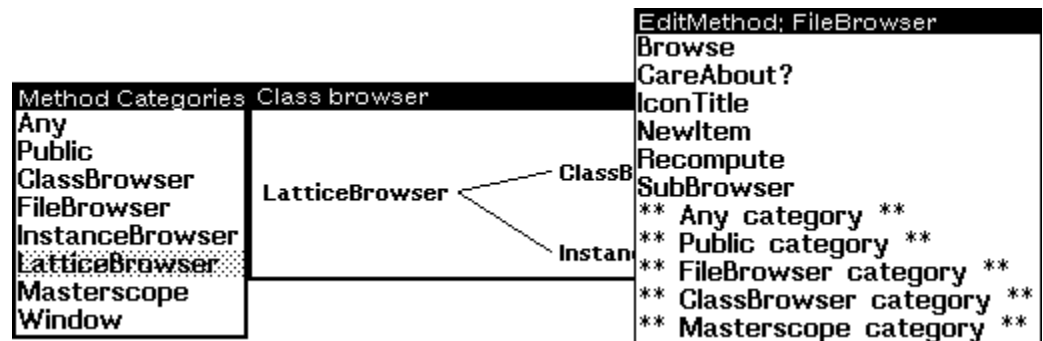
#### EditMethod

Edits a method of class, selected from a menu of local methods. The menu that appears contains some or all of the methods of the class and some category choices. The menu options that appear depend on the shaded options on the **Method Categories** menu (see Section 10.3.1.3, "Add Category Menu").

For example, assume the following actions occur:

- A class browser is opened on the class **LatticeBrowser**.
- A category menu is added and the category **LatticeBrowser** is selected.
- **FileBrowser** under the class **ClassBrowser** is selected.
- The **EditMethod** suboption is selected.

The following menu appears:



If you select one of the method names, an edit window appears containing the source code for the method (assuming the source code was loaded). If you select one of the categories, a similar menu appears showing the changed methods and categories. The method/category menu continues to appear until you select a method or press a mouse button outside of the menu. This operation is provided by the method **PickSelector**.

#### EditMethod!

Edits a method selected from a menu of all the inherited methods, making the method local if necessary.

#### EditMethodObject

Edits the object representing the method, as shown in this display editor window:

```

SEdit #,($& Method (NEW0.1Y;.;h.eN6 , 453
((className FileBrowser)
 (selector Recompute)
 (method FileBrowser.Recompute)
 (args NIL)
 (doc NIL)
 (category (LatticeBrowser)))

```

This suboption uses the method **PickSelector** to provide the same menu interface for choosing a selector that **EditMethod** uses. Within this edit window, the only items you should change are the **doc** and **category** properties.

**MethodMenu** Creates a permanent menu of methods of this class, for example,

```

Edit methods for DestroyedObject
Destroy!

```

After you have placed the menu, pressing the left mouse when the cursor is over a menu option will cause the method to be printed; pressing the middle button will cause the method to be edited.

**EditCategory** This option has three suboptions:

- **EditCategory**

Opens a menu of available categories, similar to the following menu example:

```

Method category
Any
Public
Internal
LatticeBrowser
Window

```

After you select a category, another menu appears containing the methods of that category and a list of categories not chosen (using the method **PickSelector**). When a method is selected, it appears in an edit window.

- **ChangeMethodCategory**

Using the method **PickSelector**, this prompts you for a method. When one is selected, another menu appears containing the option **\*other\*** with all of the categories of the selected class.

If one of the categories is chosen, the category for the method is changed to this value.

If **\*other\*** is chosen, this prompts you in the **PROMPTWINDOW** for a symbol or a list of symbols that will become the new category or categories for the method.

See the method **Class.ChangeMethodCategory**.

- **CategorizeMethods**

Edits an association list of categories and the local methods they contain via the editor.

This is an example of the edit window that appears:

```
SEdit Package: INTERLISP
((Any (CreateClass DestroyInstance New NewWithValues))
 (Public (CreateClass DestroyInstance New NewWithValues))
 (Internal NIL)
 (MetaClass (CreateClass))
 (Class (DestroyInstance New NewWithValues)))
```

**BreakMethod** This option has three suboptions:

- **BreakMethod**

Places a break on a method of class, selected from a menu of local methods, by sending the message **BreakMethod**.

- **TraceMethod**

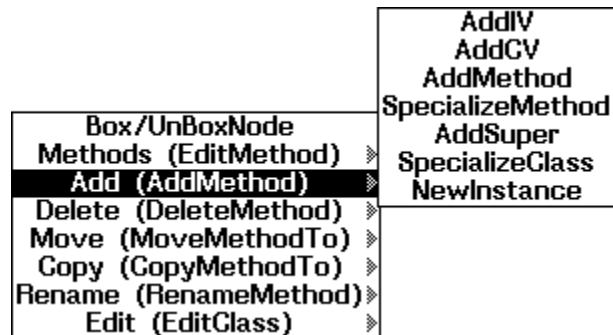
Places a trace on a method of class, selected from a menu of local methods, by sending the message **TraceMethod**.

- **UnbreakMethod**

Brings up a menu of methods local to class that have been broken, and removes any breaks or traces on the one selected by sending the message **UnbreakMethod**.

### 10.3.3.3 Add (AddMethod) and its Suboptions

Selecting the **Add (AddMethod)** option and dragging the mouse to the right causes the following submenu to appear:



**AddIV** Prompts you for the name of a new instance variable to be added to a class, and opens an editor as shown here:

```
* SEdit DestroyedObject Package: INTERLISP
(newIV **DefaultValue doc (* IV added by MCGILL))
```

From here, change the default value of the instance variable to the desired value, and change the documentation to add any other properties and values if necessary. When you exit from the editor, the instance variable is added to the class.

**AddCV** Same as **AddIV**, except that you add a class variable.

**AddMethod** Allows you to create and edit a new method for this class. You are prompted to type a selector for the new method. Next, an edit window appears with the following template:

```
SEdit DestroyedObject.newMethod Package: INTERLISP
(Method ((DestroyedObject newMethod) self)
 "Method documentation"
 (SubclassResponsibility))
```

Replace the form (SubclassResponsibility) with the functionality you want.

**SubclassResponsibility** is a macro that causes a call to **HELPCHECK**, so if you forget to remove it, or want to leave it in for debugging, it will break when the new method is invoked.

**SpecializeMethod** Causes a menu to appear, containing the selectors of inherited methods, the option **\*\* Generic Methods \*\***, and available categories.

If one of the selectors is chosen, a display editor window appears with a template that contains a **\_Super** and the same comment as the specialized method. An example of this window appears here:

```
SEdit DestroyedObject.Destroy! Package: INTERLISP
(Method ((DestroyedObject Destroy!) self)
 "Specialization"
 (<Super self Destroy!))
```

If **\*\* Generic methods \*\*** is chosen this causes a menu to appear, listing the methods inherited from **Object**, **Class**, or **Tofu**.

**AddSuper** Prompts you for a class name to be added to the beginning of the **Supers** list.

**SpecializeClass** Defines a subclass of this class. This subclass is initialized with no locally defined instance variables, class variables, or methods. You are prompted to give a name for the new class. The new class is added to the browser.

**NewInstance** Creates a new instance of this class, calls **PutSavedValue** with the new instance as an argument, and prints the instance.

#### 10.3.3.4 Delete (DeleteMethod) and its Suboptions

Selecting the **Delete (DeleteMethod)** option and dragging the mouse to the right causes the following submenu to appear:

|                              |              |
|------------------------------|--------------|
| Box/UnBoxNode                |              |
| Methods (EditMethod)         |              |
| Add (AddMethod)              |              |
| <b>Delete (DeleteMethod)</b> | DeleteIV     |
| Move (MoveMethodTo)          | DeleteCV     |
| Copy (CopyMethodTo)          | DeleteMethod |
| Rename (RenameMethod)        | DeleteClass  |
| Edit (EditClass)             |              |

**DeleteIV** Opens a menu containing the local instance variables, that is, those defined in the class. Selecting one removes it from the class.

**DeleteCV** Same as **DeleteIV**, but for class variables.

**DeleteMethod** Opens a menu containing the local selectors. Choosing one opens the following menu for confirmation:

```
Destroy!Confirm method deletion
Delete Method and Function
Abort
```

**DeleteClass** Deletes this class. Opens a menu similar to the following for confirmation:

```
Confirm
Destroy DestroyedObject
```

If the class has no subclasses it will be deleted. If it does have subclasses a **HELPCHECK** break occurs; you can then abort or type OK to destroy the class and all of its subclasses.

---

### 10.3.3.5 Move (MoveMethodTo) and its Suboptions

---

Selecting the **Move (MoveMethodTo)** option and dragging the mouse to the right causes the following submenu to appear:

|                            |          |
|----------------------------|----------|
| Box/UnBoxNode              |          |
| Methods (EditMethod)       | »        |
| Add (AddMethod)            | »        |
| Delete (DeleteMethod)      | »        |
| <b>Move (MoveMethodTo)</b> | <b>»</b> |
| Copy (CopyMethodTo)        | »        |
| Rename (RenameMethod)      | »        |
| Edit (EditClass)           | »        |

|                     |
|---------------------|
| <b>MoveIVTo</b>     |
| <b>MoveCVTo</b>     |
| <b>MoveMethodTo</b> |
| <b>MoveSuperTo</b>  |
| <b>MoveToFile</b>   |
| <b>MoveToFile!</b>  |

Moving methods and variables requires that you first have used **BoxNode** (see Section 10.3.3.1, "BoxNode") on the class which is to receive whatever is moved.

---

#### CAUTION

Moving methods and variables can have profound effects on the classes that inherit them.

---

**MoveIVTo** Opens a menu with the instance variables for the class. Choosing one causes it to be moved to the class that is boxed in the browser. When this operation is completed, the menu opens again prompting you for another instance variable to move, if desired.

**MoveCVTo** Similar to **MoveIVTo**, but for class variables.

**MoveMethodTo** Similar to **MoveIVTo**, but for methods.

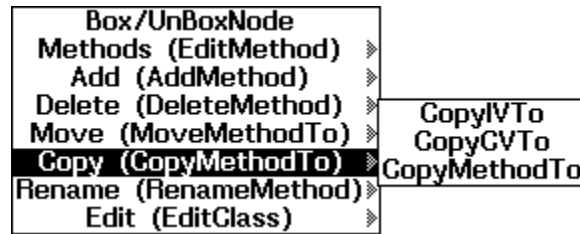
**MoveSuperTo** Opens a menu of the Supers for the class. Choosing one of these will cause it to be removed as a super class, and the boxed class to be added to the Supers list.

**MoveToFile** Opens a menu of files on **FILELST** along with **\*newFile\***. (See Section 10.2.1.2, "Command Summary," for more information on **\*newFile\***.) The class and its methods are moved to the chosen file.

**MoveToFile!** Same as **MoveToFile**, but includes subclasses and their methods of the classes.

### 10.3.3.6 Copy (CopyMethodTo) and its Suboptions

Selecting the **Copy (CopyMethodTo)** option and dragging the mouse to the right causes the following submenu to appear:

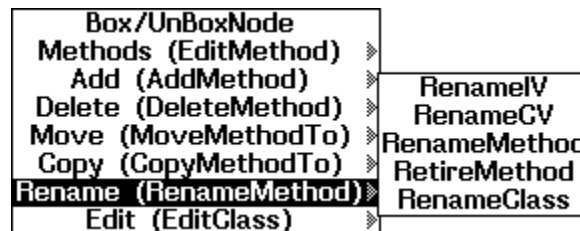


Copy options operates similarly to Move options, but leave the original method or variable in its place while adding it to the destination. You can then specialize the original or copy.

- CopyIVTo** Similar to **MoveIVTo**, but copies the instance variables instead of moving them.
- CopyCVTo** Similar to **MoveCVTo**, but copies the class variables instead of moving them.
- CopyMethodTo** Similar to **MoveMethodTo**, but copies the methods instead of moving them.

### 10.3.3.7 Rename (RenameMethod) and its Suboptions

Selecting the **Rename (RenameMethod)** option and dragging the mouse to the right causes the following submenu to appear:

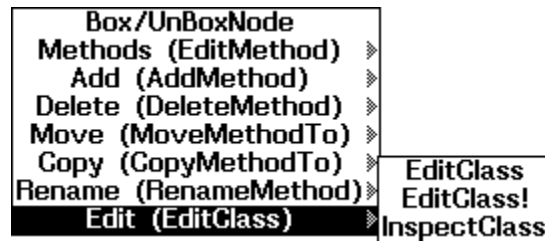


- RenameIV** A menu appears, showing the instance variables of the class. After selecting one of these, you are prompted to give a new name for that instance variable.
- RenameCV** Similar to **RenameIV**, but for class variables.
- RenameMethod** Similar to **RenameIV**, but for methods. You are prompted to give a new name for the selector of the method. The method function name is changed to reflect the change in the name of the selector.
- RetireMethod** A menu appears, showing the selectors of the class. After selecting one of these, the selector is changed by adding the prefix "Old" to it. The method function is also renamed appropriately.
- RenameClass** You are prompted for a new name for the class. After the new name is entered, the browser is updated to reflect the change. In addition, the method functions associated with the class are also renamed. For example, given a class **Foo** with selector **Fie**, when the class is renamed to **Fum**, the method function is automatically renamed from **Foo.Fie** to **Fum.Fie**.

### 10.3.3.8 Edit (EditClass) and its Suboptions



Selecting the **Edit (EditClass)** option and dragging the mouse to the right causes the following submenu to appear:



Editing options allow you to make quick, massive changes to object descriptions, and are sometimes the only menu-driven way to change certain items. See Chapter 13, Editing, for more details.

- |                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>EditClass</b>    | Edits the class definition of the class showing only the locally defined class variables, instance variables, and methods.                                                                                                                                                                                                                                                                                                                                         |
| <b>EditClass!</b>   | Edits the class definition of the class showing both the locally defined and inherited class variables, instance variables, and methods. Changes to the inherited information that are done during the edit have no effect. The inherited information is included for informational purposes only. For example, you may want to copy the definition of an instance variable from the list of inherited instance variables to the list of local instance variables. |
| <b>InspectClass</b> | Opens an inspector window on the class. See Chapter 18, User Input/Output Modules, for more details.                                                                                                                                                                                                                                                                                                                                                               |

---

## 10.4 Using File Browsers

---

File browsers are a specialization of class browsers. In addition to the capabilities of class browsers, file browsers allow you to manipulate files. This section explains the file browser menu options not available from class browser menus, or those that have been modified from the class browser menu options. This section lists all file browser menu options; references are made to class browser menus where appropriate.

Multiple files can be associated with a file browser. Thus, one of those files can be designated as the "selected" file. There are various options as to which classes should be displayed in a file browser. See Section 10.4.1.4, "Change display mode and its Suboptions," for more information.

When a file browser is opened, the window title displays the selected file:

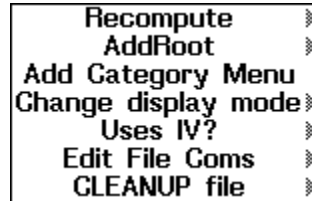


The icon for a shrunk file browser contains the name of the selected file, as shown here:



### 10.4.1 Selecting Options in the Title Bar Menu

The title bar menu in a **FileBrowser**, shown here, is like the title bar menu in a **ClassBrowser**, but has additional entries for file system and Masterscope functions.



#### 10.4.1.1 Recompute and its Suboptions

Operates identically to the class browser. See Section 10.3.1.1, "Recompute and its Suboptions," for more information.

#### 10.4.1.2 AddRoot and its Suboptions

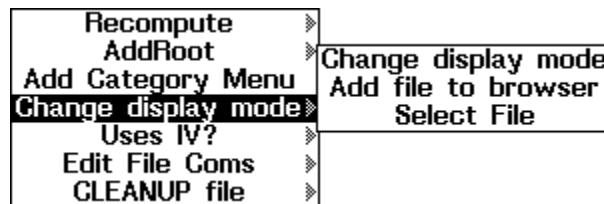
Prompts you for the name of a class to add. If the class already exists, it is added to the browser and shaded. If the class is not contained on the file, it will appear shaded in the browser. If the class does not exist, it is created and added to the selected file.

#### 10.4.1.3 Add Category Menu

Operates identically to the class browser. See Section 10.3.1.3, "Add Category Menu and its Suboptions," for more information.

#### 10.4.1.4 Change display mode and its Suboptions

Selecting the **Change display mode** option and dragging the mouse to the right causes the following submenu to appear:



A **FileBrowser** logically includes more display options than a **ClassBrowser**. A **FileBrowser** can display a class hierarchy as it is stored in the file, or as it exists in combination with other files and the system as a whole.

#### Change display mode

Selecting this option causes a sub-submenu to appear showing three options:

- **selectedFile**

Displays only the classes contained within the selected file or classes that have been added to the browser by **AddRoot** or **AddSubs** (by setting the browser's instance variable **goodList** to the appropriate value). See

Section 10.5.3, "Methods for the Class LatticeBrowser," for an explanation of **AddSubs**.

- **associatedFiles**

Same as **selectedFile**, but the browser also includes any classes defined in files associated with the browser. The instance variable **goodList** is bound to this list, slightly differently than the use of **goodList** in a class browser.

- **all**

Same as **associatedFiles**, but any subclasses, even if not defined in the files, are also displayed because the instance variable **goodList** is bound to NIL.

**Add file to browser**

Prompts you with a menu that is similar to the menu that is displayed when you select **Browse File** from the LOOPS Icon or the background menu, except that files already associated with the browser are not displayed on the menu.

**Select file**

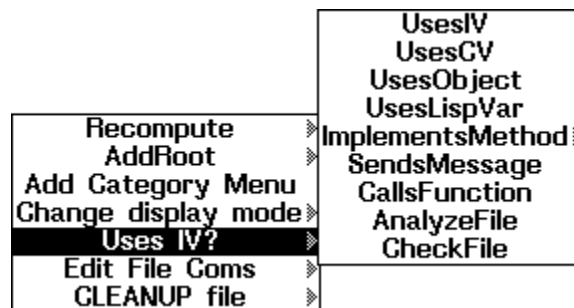
Causes a menu to appear, showing the files associated with the browser. Selecting one causes that file to become the "selected file" of the browser. It clears the browser of any classes added to the browser by the **AddRoot** and **AddSubs** menu commands. That is, it resets the starting list of the browser to be only those classes contained within the selected file. The instance variable **badList** of the browser is set to NIL.

---

#### 10.4.1.5 Uses IV? and its Suboptions

---

Selecting the **Uses IV?** option and dragging the mouse to the right causes the following submenu to appear:



These menu options trigger various Masterscope operations. Most of these operations prompt you for information that is used in a Masterscope query. The results of this query are used to build a second menu. If the situation occurs that the second menu is empty, a message is printed in the prompt window of the browser similar to "someCV not used as a CV."

---

#### CAUTION

Source files being displayed in the file browser must be available or these functions cannot work. In addition, the LOOPS Library Module LOOPSMS must also be loaded (see the *LOOPS Library Modules Manual* for details).

---

Additionally, the first time one of these options is selected there may be a pause while Masterscope analyzes the file. A window will open, and fill with

"blips" as the analysis proceeds, until the file is analyzed and the original question is answered.

#### UsesIV

This first opens a menu of instance variables defined in classes contained in the selected file of the browser. Two additional options are placed at the top of the menu:

- **\*other\***

Selecting **\*other\*** causes a prompt to enter the name of an instance variable.

- **\*any\***

Selecting **\*any\*** creates a menu with all methods that reference any instance variable.

After an instance variable has been chosen, you are prompted to place a menu that contains the following options:

- A list of methods on the selected file that use that instance variable
- A list of classes on the selected file that contain that instance variable.
- **\*EditAll\***

If one of the methods or classes is selected, it is edited. Suboptions from the methods or classes include:

- **Edit**

Edits the method.

- **Substitute**

Prompts for a new name for the instance variable. Changes the instance variable name to the new name in the method and then brings up the display editor for you to edit the method.

- **Check**

Executes the Masterscope command CHECK <file> on the file associated with the LOOPS FileBrowser. See the *Lisp Library Modules Manual* for details.

**\*EditAll\*** has the following two suboptions:

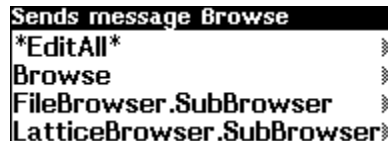


- **\*EditAll\***

Edits each method and class in succession.

- **\*SubstituteAll\***

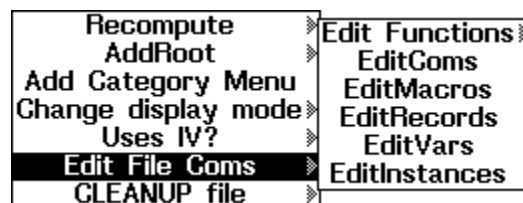
Prompts you for a new name for the instance variable. Substitutes this new name for the old name in all methods and classes listed in the menu.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>UsesCV</b>           | Same as <b>UsesIV</b> , but for class variables.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| <b>UsesObject</b>       | Opens a menu of classes or instances defined on the selected file that are used by any of the methods or functions on the selected file. After you choose one of the objects, a menu similar to the one created for <b>UsesIV</b> is created, but contains the methods and functions that use the chosen objects.                                                                                                                                                                                                                                                                                                                       |
| <b>UsesLispVar</b>      | Same as <b>UsesIV</b> , but the initial menu displays Lisp variables instead of objects.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <b>ImplementsMethod</b> | <p>This option has three suboptions:</p> <ul style="list-style-type: none"><li>• <b>ImplementsMethod</b><br/>Opens a menu of all of the selectors in the selected file. When one is chosen, a menu is created showing the methods that use that selector and the classes that are associated with those methods.</li><li>• <b>OverridesMethod</b><br/>Generates a menu of methods and classes that override (that is, does not invoke <b>_Super</b>) the selected method.</li><li>• <b>SpecializesMethod</b><br/>Generates a menu of methods and classes that specialize (that is, invoke <b>_Super</b>) the selected method.</li></ul> |
| <b>SendsMessage</b>     | Opens a menu of all of the selectors in the selected file. When one is chosen, a menu is created that lists the methods and functions that send messages using that selector. The following window shows a sample of this menu.                                                                                                                                                                                                                                                                                                                                                                                                         |
|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>CallsFunction</b>    | Opens a menu of all of the functions that are called by functions or methods in the selected file. After one is chosen, a menu is opened that contains the methods and/or functions that call the chosen function; the last option on the menu is the chosen function.                                                                                                                                                                                                                                                                                                                                                                  |
| <b>AnalyzeFile</b>      | Begins a separate process analyzing the selected file. When the analysis is completed, "Done analyzing" is printed in the browser's prompt window.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| <b>CheckFile</b>        | Begins a separate process checking the selected file. When the checking is completed, "Done checking" is printed in the browser's prompt window.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

---

#### 10.4.1.6 Edit FileComs and its Suboptions

Selecting the **Edit FileComs** option and dragging the mouse to the right causes the following submenu to appear:



The filecoms are variables that describe the contents of a file, for example, methods, classes, and Lisp functions and variables. LOOPS extends the File Manager to handle object oriented code and data, and the **FileBrowser** gives users a menu driven interface to deal with this extended file functionality.

### Edit Functions

Opens a sub-submenu giving options dealing with filecoms and with some of the items listed (functions in particular). This sub-submenu contains five suboptions:

- **EditFns**

Opens a menu of the functions contained within the selected file (those listed under FNS in the filecoms) and the option **\*NewFunction\***. Selecting one of the functions calls the editor on that function.

Selecting **\*NewFunction\*** causes a prompt for a name for the new function. An edit window then opens containing a template for a lambda expression. This newly defined function is added to the FNS list of the selected file's filecoms.

- **MakeFunctionMenu**

Does an **ADDMENU** (the Interlisp function which adds a permanent menu to the screen) of a menu containing functions on the selected file. Selecting one of the functions opens an editor on it.

- **BreakFunction**

Opens a menu containing functions on the selected file that are not on **BROKENFNS** (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*). Selecting one of the functions causes it to break next time it is invoked.

- **TraceFunction**

Same as **BreakFunction**, except that the selected function is traced.

- **UnbreakFunction**

Creates a menu of functions that are members of **BROKENFNS** and are contained in the selected file. The selected file is unbroken.

**EditComs** Edits the filecoms of the selected file.

**EditMacros** Creates a menu of macros contained in the selected file. Selecting one of them opens an editor on it.

**EditRecords** Same as **EditMacros**, but for records.

**EditVars** Same as **EditMacros**, but for variables.

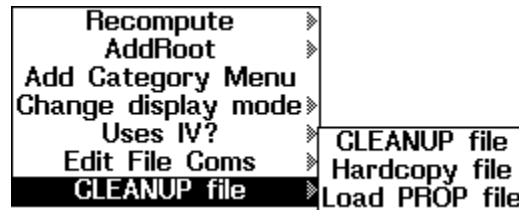
**EditInstances** Same as **EditMacros**, but for instances.

### 10.4.1.7 CLEANUP file and its Suboptions

---

This option invokes some or all of **CLEANUP**, which is the automatic file maintenance utility of Medley.

Selecting the **CLEANUP file** option and dragging the mouse to the right causes the following submenu to appear:



- |                       |                                                                                                                                                            |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CLEANUP file</b>   | Calls <b>FILES?</b> and then calls <b>CLEANUP</b> on the selected file.                                                                                    |
| <b>Hardcopy file</b>  | Sends the selected file to the <b>DEFAULTPRINTINGHOST</b> .                                                                                                |
| <b>Load PROP file</b> | Loads the selected file with <b>LDPLG</b> set to <b>PROP</b> , making sources available to Masterscope, but leaving any compiled code in place to execute. |

### 10.4.2 Selecting Options in the Left Menu

---

When the cursor is inside of a file browser and you press the left mouse button, nodes within the browser are inverted when the cursor moves over them. If you release the left mouse button while the cursor is over a node, the following menu appears:



These options include those in a class browser and add **AddSubs**, an option that expands the lattice of a file browser to look more like that of a class browser by showing related classes not stored in the browsed file.

#### 10.4.2.1 PrintSummary and its Suboptions

---

Operates identically to the class browser. See Section 10.3.2.1, "PrintSummary and its Suboptions," for more information.

#### 10.4.2.2 Doc (ClassDoc) and Its Suboptions

---

Operates identically to the class browser. See Section 10.3.2.2, "Doc (ClassDoc) and its Suboptions," for more information.

#### 10.4.2.3 WhereIs (WhereIsMethod) and Its Suboptions

---

Operates identically to the class browser. See Section 10.3.2.3, "WhereIs (WhereIsMethod) and its Suboptions", for more information.

#### 10.4.2.4 DeleteFromBrowser and Its Suboptions

Operates identically to the class browser. See Section 10.3.2.4, "DeleteFromBrowser and its Suboptions," for more information.

#### 10.4.2.5 SubBrowser

Creates an instance of a class browser with the selected class as the root node, not a file browser. See Section 10.3.2.5, "SubBrowser," for more information.

#### 10.4.2.6 TypeInName

Operates identically to the class browser. See Section 10.3.2.6, "TypeInName," for more information.

#### 10.4.2.7 AddSubs and its Suboptions

**AddSubs** fills out the class lattice in a file browser window. This shows classes, the file they are from (if any) and the inherited methods and variables from classes which are in the file.

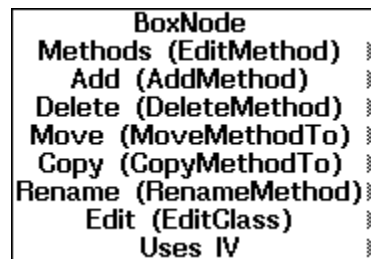
**AddSubs** Adds the immediate subclasses of the class to the browser and shades the new subclasses, as shown here:



**AddSubs!** Adds all subclasses of the class to the browser and shades the new subclasses.

### 10.4.3 Selecting Options in the Middle Menu

When the cursor is over a node and you press the middle mouse button, the following menu appears:



The middle button commands are the same as those on a **ClassBrowser**, with some new functionality for **Add (AddMethod)**, and with Masterscope options added under the option **UsesIV**.



### 10.4.3.1 BoxNode

---

Operates identically to the class browser. See Section 10.3.3.1, "BoxNode/UnBoxNode," for more information.

### 10.4.3.2 Methods (EditMethod) and its Suboptions

---

Operates identically to the class browser. See Section 10.3.3.2, "Methods (EditMethod) and its Suboptions," for more information.

### 10.4.3.3 Add (AddMethod) and its Suboptions

---

Operates identically to the class browser, but with additional functionality to keep the added items associated with the file being browsed in the following suboptions.

- |                         |                                                                                                                                                                                                                                                                            |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>SpecializedClass</b> | Prompts you to enter a name for the new subclass for the chosen class. If the chosen class is on the selected file, the new subclass is added to that file. If the chosen class is on another file, choose a file from a menu of files to which the new subclass is added. |
| <b>NewInstance</b>      | Creates a new instance of the class and calls <b>PutSavedValue</b> with the new instance as an argument. You are prompted to give the new instance a name, and the new instance is added to the selected file.                                                             |

### 10.4.3.4 Delete (DeleteMethod) and its Suboptions

---

Operates identically to the class browser. See Section 10.3.3.4, "Delete (DeleteMethod) and its Suboptions," for more information.

### 10.4.3.5 Move (MoveMethodTo) and its Suboptions

---

Operates identically to the class browser. See Section 10.3.3.5, "Move (MoveMethodTo) and its Suboptions," for more information.

### 10.4.3.6 Copy (CopyMethodTo) and its Suboptions

---

Operates identically to the class browser. See Section 10.3.3.6, "Copy (CopyMethodTo) and its Suboptions," for more information.

### 10.4.3.7 Rename (RenameMethod) and its Suboptions

---

Operates identically to the class browser. See Section 10.3.3.7, "Rename (RenameMethod) and its Suboptions," for more information.

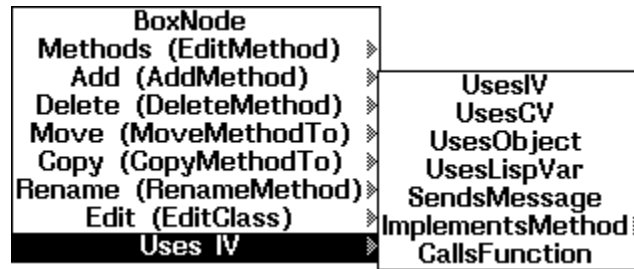
### 10.4.3.8 Edit (EditClass) and its Suboptions

---

Operates identically to the class browser. See Section 10.3.3.8, "Edit (EditClass) and its Suboptions," for more information.

### 10.4.3.9 UsesIV and its Suboptions

Selecting the **UsesIV** option and dragging the mouse to the right causes the following submenu to appear:



These commands operate similarly to the **Uses IV?** commands in the title bar menu. See Section 10.4.1.5, "UsesIV? and its Suboptions," for more information. Here, however, the Masterscope queries are limited to the class in question instead of the entire file.

## 10.5 PROGRAMMER'S INTERFACE TO LATTICE BROWSERS

### 10.5 PROGRAMMER'S INTERFACE TO LATTICE BROWSERS

## 10.5 Programmer's Interface to Lattice Browsers

LOOPS browsers are standard LOOPS objects, so their functionality can be exercised programmatically by messages which invoke their methods. Many browser functions are based on the Lisp Library Module, Grapher, but browsers apply only to dealing with LOOPS objects. All of the functionality in the menu-driven interface to the browsers is available programmatically.

Note: Data not a part of LOOPS data can be graphed with LOOPS calls to the Lisp Library Module, Grapher.

**ClassBrowsers** show the inheritance structure of object classes, a relationship defined at application design time. Browsers can also be used to show dynamic information, not computed until runtime. The LOOPS class **InstanceBrowser** does this, showing links between instances defined at runtime.

**InstanceBrowser** is derived from **LatticeBrowser** by specializing just two methods, **GetSubs** and **NewPath**. In general, you can specialize browsers to your own purposes by specializing these methods and **GetLabel**. An example is given in Section 10.7, "Class Instance Browser Example," producing a class browser which also shows instances, connected with dashed lines.

### 10.5.1 Instance Variables for the Class LatticeBrowser

Instance variables appear in alphabetical order.

|                           |                                                                                                                                                     |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>badList</b>            | A list of objects that are not displayed in the browser window.                                                                                     |
| <b>boxedNode</b>          | The last object boxed, if any.                                                                                                                      |
| <b>browseFont</b>         | The font used for labels. This has two properties: <b>FontFamily</b> and <b>FontFace</b> which are referenced in the method <b>ChangeFontSize</b> . |
| <b>lastSelectedObject</b> | The last object selected.                                                                                                                           |
| <b>goodList</b>           | A list of objects that are displayed in the browser window; if NIL, no objects are displayed.                                                       |

|                           |                                                                                                                                                                                                                                                      |
|---------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>graphFormat</b>        | A list indicating the style of layout for the graph. See the method <b>ChangeFormat</b> and the Lisp Library Module, Grapher.                                                                                                                        |
| <b>LabelMaxCharsWidth</b> | Affects the way labels are generated. This limits the width of a label to <b>LabelMaxCharsWidth</b> times the width of the character "A". See the method <b>ChangeMaxLabelSize</b> . Default value is NIL, which puts no restrictions on label size. |
| <b>LabelMaxLines</b>      | Affects the way labels are generated. This limits the number of lines in a label to <b>LabelMaxLines</b> . Refer to the method <b>ChangeMaxLabelSize</b> . Default value is NIL which puts no restrictions on label size.                            |
| <b>startingList</b>       | List of objects used to compute this browser.                                                                                                                                                                                                        |
| <b>title</b>              | Title passed to Grapher module.                                                                                                                                                                                                                      |
| <b>topAlign</b>           | This flag is used to indicate whether the graph should be aligned with the top or bottom of the window. If <b>topAlign</b> = T (the default), then the Grapher module aligns the graph to the top of the window.                                     |

---

### 10.5.2 Class Variables for the Class **LatticeBrowser**

---

Except for **BoxLineWidth**, the following class variables determine the menus that appear when a mouse is positioned over a node within a browser and the left or middle button is pressed. The default behavior is to send a message to the object represented by the node with the selector returned from the menu selection. The form for the values that these class variables can have is described in Chapter 20, Windows.

Class variables appear in alphabetical order.

|                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>BoxLineWidth</b>     | The width of line that is drawn around a node when it is boxed. See the method <b>BoxNode</b> in Section 10.5.3, "Methods for the Class <b>LatticeBrowser</b> ."                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>LeftButtonItem</b>   | Items for the menu that appears when the mouse is on a node in the browser and the left button is pressed. When an item is selected from a menu, the returned value is sent as a message to an object represented by the node. See <b>LocalCommands</b> , below.                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>LocalCommands</b>    | <p>When the cursor is positioned over a node in a browser and you press the left or middle mouse button, the default behavior is to bring up a menu from which you select an item. The value returned from that item specifies the selector of a message that is sent to the object which is represented by the node in the browser.</p> <p>The class variable <b>LocalCommands</b> provides a way to override that behavior. If the value returned from the menu selection is on the list that is the value of <b>LocalCommands</b>, the message is not sent to the object, but is sent to the browser instead. The object is passed as an argument in that message.</p> |
| <b>MiddleButtonItem</b> | Options for the menu that appears when the mouse is on one of the nodes in a browser and the middle button is pressed. When an option is selected from a menu, the returned value is sent as a message to the object represented by the node. See <b>LocalCommands</b> , above.                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>TitleItem</b>        | Options for the menu that appears when the mouse is on the title bar in a browser and the left or middle button is pressed. When an option is selected from the menu, the returned value is sent as message to the browser.                                                                                                                                                                                                                                                                                                                                                                                                                                               |

The following selectors are associated with this menu:

- **SaveInT**
- **Recompute**
- **RecomputeInPlace**

- **ShapeToHold**
- **ChangeFontSize**
- **ChangeFormat**
- **AddRoot**
- **RemoveFromBadList**

Examine the class **LatticeBrowser** for more information.

### 10.5.3 Methods for the Class LatticeBrowser

The following table shows the methods and variables for the class **LatticeBrowser**.

| Name                            | Type   | Description                                                                                  |
|---------------------------------|--------|----------------------------------------------------------------------------------------------|
| <b>AddRoot</b>                  | Method | Adds a LOOPS name or an object to the starting list of the browser.                          |
| <b>BoxNode</b>                  | Method | Puts a box around the node representing the object.                                          |
| <b>Browse</b>                   | Method | Uses a lattice or tree graph to display the relationship between a number of objects.        |
| <b>BrowserObjects</b>           | Method | Returns the list of objects currently in the graph.                                          |
| <b>ChangeFontSize</b>           | Method | Changes the size of the characters in the labels.                                            |
| <b>ChangeFormat</b>             | Method | Changes between lattice and tree graphs.                                                     |
| <b>ChangeMaxLabelSize</b>       | Method | Changes how labels are printed.                                                              |
| <b>ClearLabelCache</b>          | Method | Recomputes labels.                                                                           |
| <b>DeleteFromBrowser</b>        | Method | Prunes branches in a graph.                                                                  |
| <b>DeleteSubtreeFromBrowser</b> | Method | Prunes branches in a graph.                                                                  |
| <b>FlashNode</b>                | Method | Changes the label of a node from black-on-white to white-on-black several times.             |
| <b>FlipNode</b>                 | Method | Changes the label of a node that represents an object from black-on-white to white-on-black. |
| <b>GetDisplayLabel</b>          | Method | Finds the label for a node.                                                                  |
| <b>GetLabel</b>                 | Method | Computes a label for an object.                                                              |
| <b>GetSubs</b>                  | Method | Computes a list of subnodes of an object.                                                    |
| <b>GraphFits</b>                | Method | Determines if the graph in the browser can be contained within the browser window.           |
| <b>HasObject</b>                | Method | Returns T if an object is in the graph.                                                      |
| <b>HighlightNode</b>            | Method | Changes the way a node is displayed.                                                         |
| <b>IconTitle</b>                | Method | Computes the title to write in the icon.                                                     |
| <b>LeftSelection</b>            | Method | Controls the effect of using the left mouse button.                                          |

|                          |          |                                                                                                                    |
|--------------------------|----------|--------------------------------------------------------------------------------------------------------------------|
| <b>LeftShiftSelect</b>   | Method   | Sends the message <b>PP!</b> to an object.                                                                         |
| <b>MiddleSelection</b>   | Method   | Controls the effect of using the middle mouse button.                                                              |
| <b>MiddleShiftSelect</b> | Method   | Invoked by the mouse operations to edit an object in the TTY process context.                                      |
| <b>NewItem</b>           | Method   | Gets an object.                                                                                                    |
| <b>NodeRegion</b>        | Method   | Returns the region occupied in an object in the browser.                                                           |
| <b>ObjectFromLabel</b>   | Method   | Returns the object in the graph that has a specified label.                                                        |
| <b>PositionNode</b>      | Method   | Places a node at a particular position in the browser window.                                                      |
| <b>Recompute</b>         | Method   | Recomputes the browser graph in the same window.                                                                   |
| <b>RecomputeInPlace</b>  | Method   | Recomputes the browser graph in the same window, trying to maintain the same scroll position in the window.        |
| <b>RecomputeLabels</b>   | Method   | Recomputes the labels in a browser.                                                                                |
| <b>RemoveHighlights</b>  | Method   | Removes all highlights in any node in the graph.                                                                   |
| <b>RemoveShading</b>     | Method   | Removes all shading in any node in the graph.                                                                      |
| <b>SaveInIt</b>          | Method   | Places a pointer in a browser where it can be accessed.                                                            |
| <b>ShadeNode</b>         | Method   | Adds shading to a node.                                                                                            |
| <b>ShapeToHold</b>       | Method   | Reshapes the window to all items.                                                                                  |
| <b>MaxLatticeWidth</b>   | Variable | Restricts the maximum width of a browser window.                                                                   |
| <b>MaxLatticeHeight</b>  | Variable | Restricts the maximum height of a browser window.                                                                  |
| <b>Show</b>              | Method   | Displays items and their subitems in a browser window.                                                             |
| <b>Shrink</b>            | Method   | Shrinks a browser window.                                                                                          |
| <b>SubBrowser</b>        | Method   | Creates a browser that is an instance of the same class as <i>self</i> with a specified object as the root.        |
| <b>TitleSelection</b>    | Method   | Invokes an action when the mouse is in the title bar on a browser window and the left or middle button is pressed. |
| <b>UnmarkNodes</b>       | Method   | Sends the messages <b>RemoveHighlights</b> and <b>RemoveShading</b> to <i>self</i> .                               |

---

(← *self* **AddRoot** *newItem*)

[Method of LatticeBrowser]

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose:    | Adds <i>newItem</i> , which is a LOOPS name, to the starting list of the browser.                                                                                                                                                                                                                                                                                                                                                                                                |
| Behavior:   | First determines if the name <i>newItem</i> points to a LOOPS object. If it does not, a message is printed that nothing has been added to the browser. If <i>newItem</i> is NIL, you are prompted to enter a name through the method <b>NewItem</b> . If the object pointed to by <i>newItem</i> is on the browser's instance variable <b>badList</b> , it is removed from <b>badList</b> . If the instance variable <b>goodList</b> has a value, <i>newItem</i> is added to it. |
| Arguments:  | <i>newItem</i> LOOPS name.                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Returns:    | Class object or NIL.                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| Categories: | LatticeBrowser                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

Example: The following command adds class **Datum** to a class browser instance named **CB1**:

```
55← (← ($ CB1) AddRoot ($ Datum))
```

(← self **BoxNode** object objName unboxPrevious)

[Method of LatticeBrowser]

Purpose: Puts a box around the node in the graph representing the object.

Behavior: First checks to make sure *object* points to a LOOPS object. If not, nothing happens. The previous value of the instance variable **boxedNode** is returned.

- If the instance variable **boxedNode** is NIL, then a box is drawn around the node with a line width equal to the value of the class variable **BoxLineWidth**. The instance variable **boxedNode** is assigned the value of *object*, and *object* is returned.
- If *object* is EQ to the instance variable **boxedNode**, then the box is erased. That is, calling **BoxNode** twice in succession will draw and then erase the box. The instance variable **boxedNode** is assigned the value NIL, and NIL is returned.
- If none of the above conditions hold, the flag **unboxPrevious** is checked. If it is non-NIL, the previously boxed node is unboxed, and the node represented by *object* is boxed. The instance variable **boxedNode** is assigned the value of *object*, and *object* is returned.

It is possible that *object* is not a node in *self*.

Arguments: *object* LOOPS name or object.

*objName* Used internally; can be NIL.

*unboxPrevious*  
Can be NIL or T.

Returns: The object in the browser that is currently boxed, or NIL if nothing is currently boxed.

Categories: LatticeBrowser

Specializations: ClassBrowser

(← self **Browse** browseList windowOrTitle goodList position)

[Method of LatticeBrowser]

Purpose: Uses a lattice or tree graph to display the relationships between a number of objects. **Browse** is the proper message to use for initializing browsers.

Behavior: Sends the message **Show** to *self* passing the arguments *browseList*, *windowOrTitle*, and *goodList*. It next sends **ShapeToHold** to *self*. Finally it sends **Move** to *self* with the argument *position*.

Arguments: *browseList* A list, elements of which can be a LOOPS name or an object, or a single item which can be a LOOPS name or an object. Used as the starting node(s) of a browser. See the **Show** message later in this section for details.

*windowOrTitle*  
If a window, the browser is displayed in this window. If not a window, this becomes the title of the browser window.

*goodList* A list, elements of which can be a LOOPS name or an object. See the **Show** message later in this section for details.

*position*      A position to which the lower left corner of the browser is moved.  
Can be NIL.

Returns:      Used for side effect only.

Categories:    LatticeBrowser

Example:      The following command gets the class browser instance **CB1** to browse class **Datum** and its subclasses:

```
57← (← ($ CB1) Browse 'Datum)
```

---

(← *self* **BrowserObjects**) [Method of LatticeBrowser]

---

Purpose/Behavior: Returns the list of objects currently in the graph of the browser.

Categories:    LatticeBrowser

---

(← *self* **ChangeFontSize** *size*) [Method of LatticeBrowser]

---

Purpose:      Changes the size of the characters of the labels.

Behavior:    Changes the font used to display labels in a browser. The browser is redrawn and the window shaped to fit. If no size is given, this lets you select the size from a menu. This menu is bound to the top level binding of the variable **MenuSize** (the first time it is called). The font family used is the value of the **FontFamily** property of the instance variable **browseFont**. The font face used is the value of the **FontFace** property of the instance variable **browseFont**.

This sends the message **RecomputeLabels** to *self*.

Arguments:    *size*      Integer size of font for node labels.

Returns:      Used for side effect only.

Categories:    LatticeBrowser

---

(← *self* **ChangeFormat** *format*) [Method of LatticeBrowser]

---

Purpose:      Changes between lattice and tree graphs.

Behavior:    If *format* is NIL, then select a format from a menu that appears. The items in this menu are determined by the value of the **choices** property of the instance variable **graphFormat**. Changes the value of the instance variable **graphFormat** to *format* or the value selected from the menu.

Arguments:    *format*      Describes the format layout. The argument *format* is an unordered list of atoms or lists. The following options control the structure of the graph:

- COMPACT, the default, which lays out the graph as a forest (that is, a set of disjoint trees) using the minimal amount of screen space.
- FAST, which lays out the graph as a forest, sacrificing screen space for speed.
- LATTICE, which lays out the graph as a directed acyclic graph, that is, a lattice.

In addition, the following options control the direction of the graph:

- HORIZONTAL, the default, has roots at the left and links that run left-to-right.
- VERTICAL has roots at the top and links that run top-to-bottom.

See the function **LAYOUTGRAPH** in the Grapher library module documentation for more information.

Returns: Used for side effect only.

Categories: LatticeBrowser

Example: The commands:

```
58← (SETQ b1 (←New ($ SupersBrowser) Browse 'ClassBrowser))
#, ($& SupersBrowser (NEW0.1Y%:.;h.eN6 . 506))

59← (← b1 ChangeFormat ' (HORIZONTAL REVERSE (MARK BORDER 3 LABELSHADE 1)))
#, ($& SupersBrowser (NEW0.1Y%:.;h.eN6 . 506))
```

results in:



(← self **ChangeMaxLabelSize** *newMaxWidth* *newMaxLines*)

[Method of LatticeBrowser]

Purpose: Changes how labels are printed.

Behavior: Sets the maximum width of a node label. An argument value of zero means no maximum size, and NIL means no change.

By setting both *newMaxWidth* and *newMaxLines*, you get an abbreviation facility. This binds the values of the instance variables **LabelMaxCharsWidth** and **LabelMaxLines**. The default values for **LabelMaxCharsWidth** and **LabelMaxLines** are both 0, so to return a browser b1 to default performance, send

```
(← ($ b1) ChangeMaxLabelSize 0 0).
```

The resulting labels may be bitmaps or strings. The width of the bitmap is a product of *newMaxWidth* and the width of the character "A" in the current value of the instance variable **browsefont**.

Sends the message **RecomputeLabels** to self.

Arguments: *newMaxWidth*  
Maximum number of characters per line; default is 0.

*newMaxLines*  
Maximum number of lines; default is 0.

Returns: Used for side effect only.

Categories: LatticeBrowser

Example: The commands:

```
60← (SETQ b1 (←New ($ SupersBrowser) Browse 'ClassBrowser))
#, ($& SupersBrowser (NEW0.1Y%:.;h.eN6 . 506))
```

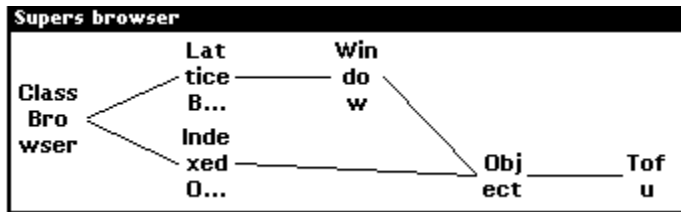


## 10.4 USING FILE BROWSERS

---

```
61←(← b1 ChangeMaxLabelSize 3 3)
#,($& SupersBrowser (NEW0.1Y%:.;h.eN6 . 506))
```

results in:



---

(← self **ClearLabelCache** *objects*)

[Method of LatticeBrowser]

Purpose: Forgets cached labels in the browser.

Behavior: Clears the label cache for item(s) in *objects*. If *objects* is the symbol T, then this clears the entire label cache. The cache for the labels is on the **objectLabels** property of the instance variable **menus**.

Arguments: *objects* An object or a list of objects.

Returns: Used for side effect only.

Categories: LatticeBrowser

---

(← self **DeleteFromBrowser** *object objname*)

[Method of LatticeBrowser]

Purpose: Prunes branches in a graph.

Behavior: Removes *object* from the browser by putting it on the instance variable **badList** and then sending the **Recompute** message to *self*. The object and its subtree are deleted from the browser.

Arguments: *object* An object in the browser.

*objname* Used internally; can be NIL.

Returns: Used for side effect only.

Categories: LatticeBrowser

---

(← self **DeleteSubtreeFromBrowser** *object*)

[Method of LatticeBrowser]

Purpose: Prunes branches in a graph.

Behavior: Similar to **DeleteFromBrowser**, but the subnodes of *object* are also added to the instance variable **badList**.

Arguments: *object* An object in the browser.

Returns: Used for side effect only.

Categories: LatticeBrowser

---

(← *self* **FlashNode** *node N flashTime leaveFlipped?*) [Method of LatticeBrowser]

---

Purpose/Behavior: Changes the label of a node from black-on-white to white-on-black several times.

Arguments: *node* LOOPS name or object.

*N* Number of times *node* will be flipped.

*flashTime* The amount of time in milliseconds that the node is held between transitions. If *flashTime* is NIL, this time defaults to 300 milliseconds.

*leaveFlipped?*  
Can be NIL or T. If T, *node* is left inverted from its original state.

Returns: Used for side effect only.

Categories: LatticeBrowser

---

(← *self* **FlipNode** *object*) [Method of LatticeBrowser]

---

Purpose: Inverts the label of a node.

Behavior: If the node is black-on-white then it is changed to white-on-black, and conversely.

Arguments: *object* LOOPS name or object.

Returns: Used for side effect only.

Categories: LatticeBrowser

---

(← *self* **GetDisplayLabel** *object*) [Method of LatticeBrowser]

---

Purpose: Finds the label for a node in the graph.

Behavior: If there is a cached label on the **objectLabels** property of the instance variable **menus**, return it.

If not, this takes the result of **GetLabel** and breaks it into multiple lines to fit in the maximum label size defined by the instance variables **LabelMaxCharsWidth** and **LabelMaxLines**, if these are non-NIL. This placement tries to break the label after special characters such as . , ; / or space, or at changes from lowercase to uppercase. The resulting bitmap is put into cache so that recomputing the graph is faster.

When a label is broken up into multiple lines, the label is changed from a string to a bitmap, thus causing shading not to work as described later in this section in the method **ShadeNode**.

Arguments: *object* An object.

Returns: Used for side effect only.

Categories: LatticeBrowser

---

(← *self* **GetLabel** *object*) [Method of LatticeBrowser]

---

Purpose: Computes a label for *object*. A label may be a symbol or a bitmap; bitmap labels should be freshly created since the method **ShadeNode** may smash them. (The method **GetDisplayLabel** is used internally to fetch labels for display; it caches label bitmaps to minimize the use of **GetLabel**.)

Behavior: Returns (**GetObjectName** *object*).

Arguments: *object* LOOPS name or object, which can be a bitmap.

Returns: (**GetObjectName** *object*)

Categories: LatticeBrowser

---

(← *self* **GetSubs** *object*) [Method of LatticeBrowser]

---

Purpose: Computes a list of subnodes of *object*.

Behavior: Determines next level of nodes in lattice. Specializations of **LatticeBrowser** typically specialize this method.

Arguments: *object* A LOOPS object.

Returns: NIL or the value of the instance variable **subs** of *object*.

Categories: LatticeBrowser

Specializations: ClassBrowser, InstanceBrowser, MetaBrowser, SupersBrowser

---

(← *self* **GraphFits** *snugly*) [Method of LatticeBrowser]

---

Purpose/Behavior: Determines if the graph in the browser can be contained within the window of the browser.

Arguments: *snugly* If *snugly?* is non-NIL the graph must fit in the window leaving less than twice the **FONTHEIGHT** of the browser's *browseFont* as empty space around it.

Returns: T if the entire graph can be displayed within the window; else NIL.

Categories: LatticeBrowser

---

(← *self* **HasObject** *object*) [Method of LatticeBrowser]

---

Purpose/Behavior: Returns T if *object* is in the graph of the browser.

Arguments: *object* LOOPS name or object.

Categories: LatticeBrowser

---

(← *self* **HighlightNode** *object width shade*) [Method of LatticeBrowser]

---

Purpose: Changes the way a node is displayed.

Behavior: Draws a box around a node for *object* using a given *width* and *shade* for the lines of the box. A shade is a 16-bit number representing a 4x4 bitmap. See **EDITSHADE** in the *Interlisp-D Reference Manual*.

Arguments: *object* LOOPS name or object.

*width* Integer width of box.

*shade* 16-bit number representing a 4x4 bitmap.

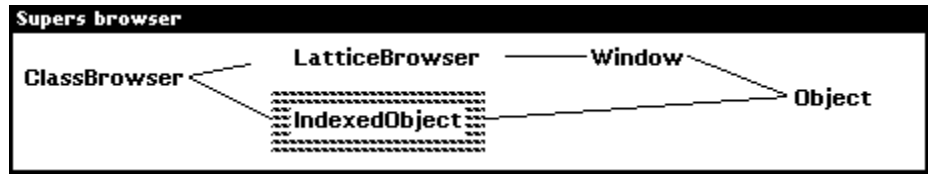
Returns: Used for side effect only.

Categories: LatticeBrowser

Example: The command

```
64← (← b1 HighlightNode 'IndexedObject 10 123)
```

results in the following window:



(← self **IconTitle**)

[Method of LatticeBrowser]

Purpose/Behavior: Computes the title to write in the icon.

Returns: The label of the first root entry in the lattice, that is, the **CAR** of the instance variable **startingList**. If this is NIL, then use "Browser". If **AddRoot** is called, the new root becomes the first entry on **startingList**.

Categories: LatticeBrowser

Specializations: FileBrowser

(← self **LeftSelection**)

[Method of LatticeBrowser]

Purpose: Controls the effect of using the left mouse button. LatticeBrowser provides defaults, but allows these methods to be overwritten or specialized.

Behavior: The instance variable **lastSelectedObject** is bound to the object of the node that the left button selected.

The remaining behavior varies according to the key pressed.

- If the **Move** key or the **Control** key is down, this allows you to move the node the mouse is over when you press the left mouse button.
- If the left shift key or **Copy** key is down while the cursor is over a node, the label of the node is copied to the system buffer. If the cursor is not over a node, the entire graph is copied. This allows you to copy browsers into TEdit documents.
- If the **Meta** key is pressed, the message **LeftShiftSelect** is sent to *self*

# 11. ERRORS AND BREAKS

LOOPS provides an interface to the Medley error system. This allows appropriate detection and recovery from errors that are LOOPS errors rather than Lisp errors. The full power of the Medley error system is available to help you determine and repair the causes of errors. In addition, under certain circumstances, LOOPS will attempt to repair an error and continue if you agree.

This chapter describes the functions and methods LOOPS uses to handle error conditions. It also describes the error messages generated by LOOPS.

## 11.1 Error Handling Functions and Methods

LOOPS provides several ways to trap and process many common errors. A default processing is available for most errors, and this processing can be specialized for actions you may require.

The following table shows the items in this section.

| Name                        | Type                 | Description                                                                                    |
|-----------------------------|----------------------|------------------------------------------------------------------------------------------------|
| <b>HELPCHECK</b>            | Function             | Provides an interface to the Common Lisp error system.                                         |
| <b>LoopsHelp</b>            | NoSpread<br>Function | Generates an error if <b>LoopsDebugFlg</b> =NIL, else calls <b>HELP</b> .                      |
| <b>LoopsDebugFlg</b>        | Variable             | Controls the behavior of <b>LoopsHelp</b> .                                                    |
| <b>ErrorOnNameConflict</b>  | Variable             | Calls <b>HELPCHECK</b> when you attempt to give an object the same name as an existing object. |
| <b>CVMissing</b>            | Method               | Sent by access functions when you attempt to access a class variable that does not exist.      |
| <b>CVValueMissing</b>       | Method               | Sent by access functions when you attempt to access a class variable that has no value.        |
| <b>IVMissing</b>            | Method               | Sent by access functions when you attempt to access an instance variable that does not exist.  |
| <b>IVValueMissing</b>       | Method               | Sent by access functions when you attempt to access an instance variable that has no value.    |
| <b>MessageNotUnderstood</b> | Method               | Sent when a message has no corresponding selector.                                             |

(**HELPCHECK** *mess1 ... messN*)

[Function]

Purpose/Behavior: **HELPCHECK** is the LOOPS interface with the Common Lisp error system. When LOOPS detects an error, it generally calls this function with up to four argument messages describing what is wrong and possibly what to do about

it. **HELPCHECK** calls **BREAK1** to put you into a break window and returns whatever the call to **BREAK1** returns. For example, if you type **OK**, it returns **T**. If you type "RETURN 'someValue'", it returns that value. In some instances, **LOOPS** uses such returned values to repair errors and continue execution.

Arguments: *mess1 ... messN*

Messages to print at the break.

Returns: Value depends on what you type in the break window; see Behavior.

Example: The following code causes a break window with the message "Are you certain?". If you type "OK" in the break window, the message "He said OK" will print.

```
(IF (HELPCHECK "Are you certain?")
 THEN (PRINT "He said OK"))
```

---

**(LoopsHelp *mess1 ... messN*)****[NoSpread Function]**

Purpose/Behavior: Generates an error. Calls **HELP** if **LoopsDebugFlg** is **T**, otherwise calls **ERROR**. Use **LoopsHelp** whenever you want to give the user a way to recover from errors when **LoopsDebugFlg** is **T**. For example, have **LoopsHelp** print messages like "FOO is not the name of a class. Type RETURN '<classname>' to continue using <classname>."

Arguments: *mess1 ... messN*

Messages to print at the break.

Returns: Value depends on what you type in the break window; see **HELPCHECK**, above.

---

**LoopsDebugFlg****[Variable]**

Purpose/Behavior: Controls the behavior of **LoopsHelp**. If it is **T**, all calls to **LoopsHelp** generate a break. If it is **NIL**, such calls that occur near the top of the stack or after a short computation cause a message to be printed and a return to the next level. The default value is **T**. See **BREAKCHK** in the *Interlisp-D Reference Manual* for more information.

---

**ErrorOnNameConflict****[Variable]**

Purpose/Behavior: If **T**, an attempt to give an object the same name as an existing object causes a call to **HELPCHECK**. If you type "OK" in the resulting break window, the process continues and the original object is unnamed. The default value is **NIL**.

---

**(← self **CVMissing** *object varName propName typeFlg newValue*)****[Method of Class]**

Purpose: Sent by access functions when there is an attempt to access a class variable that does not exist.

Behavior: Calls **LoopsHelp** with the message

*varName* not a CV of *self*

This method can be specialized to take more sophisticated action by using the other arguments which are provided.

When, in an instance, an attempt is made to access a class variable that does not exist, the message **CVMissing** is sent to the instance's class with the instance in question as *object*.

Note: This method can be invoked if an instance variable is missing.

Arguments: *object* The object upon which the access was attempted.

*typeFlg* The name of the access function (**GetValue**, **GetValueOnly**, **PutValue**, **PutValueOnly**) which caused this message to be sent. The function name allows the type of access to be determined.

*varName* The name of the variable on which access was attempted.

*propName* The name of the property on which access was attempted. If NIL, the value of the class variable *varName* was accessed.

*newValue* The value to which the class variable was to be set.

Categories: Class

Example: Specialize this method to automatically add the class variable which is missing to the class described by *self*. Assuming the class of *self* is **SomeClass**, the method definition is

```
(Method ((SomeClass CVMissing)
 self object varName propName typeFlg
 newValue)
(← self AddCV varName newValue))
```

(← self CVValueMissing object varName propName typeFlg)

[Method of Class]

Purpose: Sent by access functions when there is an attempt to access a class variable that has no value. This method can also be invoked if an instance variable is missing and you attempt to access it.

Behavior: If *propName* is NIL it returns the value of **NotSetValue**, otherwise it returns the value of **NoValueFound**.

The default setting for **NoValueFound** is NIL. The default setting **NotSetValue** is an annotatedValue. See Chapter 8, Active Values, for an explanation of **NotSetValue**.

This method can be specialized to take more sophisticated action by using the other arguments which are provided. See the example for **CVMissing**, above.

Arguments: *object* The object on which the access was attempted.

*typeFlg* The name of the access function (**GetValue**, **GetValueOnly**, **PutValue**, **PutValueOnly**) which caused this message to be sent. The function name allows the type of access to be determined.

*varName* The name of the variable on which access was attempted.

*propName* The name of the property on which access was attempted. If NIL, the value of the class variable *varName* was accessed.

Returns: Value depends on the arguments; see Behavior.

Categories: Class

(← self IVMissing varName propName typeFlg newValue)

[Method of Object]

Purpose: Sent by access functions when there is an attempt to access an instance variable that cannot be found in *self*.

Behavior: Tries to remedy the situation, but if it fails, it calls **LoopsHelp** with the message

*varName* not an IV of *self*

If the instance variable is present in the object's class, the instance variable will be copied to *self*. This can happen when a class is changed after an instance has been created.

If the instance variable is not present in the class, it attempts to find a class variable of the same name in the class. If one is found, it is used according to its **:allocation** property.

- If the property is **dynamicCached**, the instance variable is added by copying the class variable regardless of the type of access.
- If the property is **dynamic**, the type of access is determined from *typeFlg*, which is the name of the access function. The value of the class variable is returned for a get and the instance variable is created only on a put.
- If the property is **class**, the class variable's value is returned or set and no instance variable is created.

If all else fails, an attempt is made to fix the spelling of *varName* and, if a possible fixed spelling is found, the process starts over.

If an instance variable is not found, the arguments are not used, but could be in a specialization of this method. See the example in **CVMissing** above.

|             |                                               |                                                                                                                                                                                                                       |
|-------------|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Arguments:  | <i>typeFlg</i>                                | The name of the access function ( <b>GetValue</b> , <b>GetValueOnly</b> , <b>PutValue</b> , <b>PutValueOnly</b> ) which caused this message to be sent. The function name allows the type of access to be determined. |
|             | <i>varName</i>                                | The name of the variable on which access was attempted.                                                                                                                                                               |
|             | <i>propName</i>                               | The name of the property on which access was attempted. If NIL, the value of the instance variable <i>varName</i> was accessed.                                                                                       |
|             | <i>newValue</i>                               | The value to which the instance variable was to be set.                                                                                                                                                               |
| Returns:    | Value depends on the arguments; see Behavior. |                                                                                                                                                                                                                       |
| Categories: | Object                                        |                                                                                                                                                                                                                       |

---

(← *self* **IVValueMissing** *varName propName typeFlg newValue*) [Method of Object]

---

Purpose: Sent by access functions when there is an attempt to access an instance variable which has no value in *self*.

Behavior: Looks up the class hierarchy to find a value. If none is found, **SHOULDNT** (see the *Interlisp-D Reference Manual*) is called with the message

Error in Put or GetValue.

The arguments are not used, but could be in a specialization of this method. See the example in **CVMissing**, above.

This method is used internally to handle inheritance of instance variable values. If this error occurs, the LOOPS system has probably been corrupted.

Arguments: *typeFlg* The name of the access function (**GetValue**, **GetValueOnly**, **PutValue**, **PutValueOnly**) and allows the type of access to be determined.

The other arguments are passed from the access function.

Categories: Object



(← self **MessageNotUnderstood** selector messageArguments superFlg) [Method of Object]

- Purpose: Sent when a message has no corresponding selector in *self*.
- Behavior: Attempts to fix the spelling of *selector*. If this fails, it generates an error.
- Arguments: *selector* The name of the message that was not understood.  
*messageArguments* The arguments of the message *selector*.  
*superFlg* If T, an attempt was made to locate the method *selector* in the supers of *self*.
- Categories: Object
- Example: Define a class that acts as the class of Lisp numbers, and use the **MessageNotUnderstood** message to translate messages into function calls.

```
37← (DefineClass 'Number)
#.($ Number)

38← (← ($ Number) SpecializeMethod 'MessageNotUnderstood)
```

The **MessageNotUnderstood** method is defined in the editor, making the body of the method as follows:

```
(if (GETD selector)
 then (APPLY selector messageArguments))
else (←Super))
Number.MessageNotUnderstood
```

Use the class **Number** as the LOOPS class for Lisp numbers.

```
39← (PUTHASH 'SMALLP ($ Number) LispClassTable)
#.($ Number)

40← (PUTHASH 'FIXP ($ Number) LispClassTable)
#.($ Number)

41← (PUTHASH 'FLOATP ($ Number) LispClassTable)
#.($ Number)
```

Test it out.

```
42← (← 4 PLUS 5)
9
```

## 11.2 ERROR MESSAGES

### 11.2 ERROR MESSAGES

## 11.2 Error Messages

This section contains the LOOPS error messages along with their explanations. Atoms which are in *italics* are replaced with specific values when the messages are generated. Messages generated by calls to **SHOULDNT** indicate problems in LOOPS system code. Messages generated by direct calls to **ERROR**, that is not via calls to the LOOPS function **LoopsHelp**, may indicate problems with the system or with user code.

Errors appear in their respective categories:

- Errors that occur when accessing classes and instances in LOOPS.

- Errors that occur when sending messages to LOOPS objects.
- Errors dealing with naming objects.
- Errors encountered when using annotated values and active values.
- Other error messages that may be encountered when using LOOPS.

### 11.2.1 Classes and Instances

---

This section describes errors that occur when accessing classes and instances.

---

#### *type* not recognized part of class

Explanation: The *type* argument to the method **ListAttribute** does not correspond to one of the parts of a class.

---

#### *name* not a CV of *self*

Explanation: A reference has been made to a class value that does not exist.

---

#### Error in Put or GetValue

Explanation: An attempt has been made to access an instance variable that has no value in an object or in any of its supers.

---

#### *varName* not an IV of *self*

Explanation: An attempt has been made to access an instance variable and it does not exist in the object or its supers, and a class variable of the same name does not exist either.

---

#### *varName* is not a local instance variable of class *name*. Type OK to ignore error and go on.

Explanation: An attempt has been made to delete an instance variable which is not in the class.

---

#### *newValue* is not a class. Type OK to replace metaclass of *classRec* with \$Class

Explanation: A call has been made to **PutClass** or **PutClassOnly** with either *propName* erroneously set to NIL or left out, or the new metaclass set to something that is not a valid class.

---

#### *varName* is not a CV of *Class* so cannot be moved from there

Explanation: An attempt has been made to move a class variable from a class where it does not exist. Possible causes include wrong source class or misspelled class variable name.

*class* has subclasses. You cannot **Destroy** classes that have subclasses.  
Type OK to use **Destroy!** if that is what you want.

---

Explanation: Sending the message **Destroy** to a class with subclasses will leave the subclasses referring to nonexistent superclasses. **Destroy!** destroys all of the subclasses as well. Be sure this is what you want before you type "OK" .

### 11.2.2 Methods and Messages

---

This section describes errors that occur when sending messages to LOOPS objects.

---

**GetValue, PutValue, GetValueOnly, PutValueOnly or GetIVHere self args not possible**

---

Explanation: An attempt has been made to access a value in an abstract class, which cannot have any values.

---

**← or ←Super self selector -- not understood**

---

Explanation: Neither the object to which the message was sent nor any of its ancestors has such a method selector.

---

**(← NIL selector --) not understood**

---

Explanation: An attempt has been made to send a message to NIL. One way to do this is to execute `(_$ foo ...)`, where foo does not name a LOOPS object.

---

**class does not contain the selector selector. Type RETURN 'selectorName to try again**

---

Explanation: An attempt has been made to delete a nonexistent method. If the problem is that the wrong method selector was typed or the selector was misspelled, typing "RETURN 'correctName" will fix the problem.

---

**selector is not local for self To copy anyway, type OK**

---

Explanation: The object to which **CopyMethod** was sent does not contain *selector*, but one of its supers does. This is not necessarily an error.

---

**selector is not a selector for self**

---

Explanation: Neither the object to which **CopyMethod** was sent nor any of its supers contains *selector*.

---

**newClass is not a class. Type OK to use oldClass**

---

Explanation: Something may be missing from the argument to **HELPCHECK**, since nothing is printed after oldClass. Alternatively, the destination class specified in **CopyMethod** is neither a class nor a valid class name.

Typing "OK" causes the method to be copied to the class to which the message was sent. The net result can be to copy a method down from one of the class's supers or to make a copy within the class with a new selector.

---

**name is not a defined function**

---

Explanation: The selector named in **CopyMethod** exists but it does not have a function defined for it. It is possible the class has been loaded but the method has not or that the function definition for the method was somehow erroneously destroyed.

---

**name not a currently defined class. Cannot add method to class.  
Type OK to create class and go on.**

---

Explanation: An attempt has been made to add a method to a nonexistent class.  
If the class should exist, but has not been created yet, type "OK" to let LOOPS create it automatically. If the class has yet to be loaded, abort and load it first.

---

Can't find source for *fn*

---

Explanation: The source file containing a method of a class that is being moved via **MoveToFile** cannot be found. **WHEREIS** is used to try to find it. Either add the necessary file to **FILELST** or use **LOADFNS** to load the function(s).

### 11.2.3 Naming Objects

---

This section describes errors that occur when naming objects.

---

*name* is already used as a name for an object

---

Explanation: **ErrorOnNameConflict** has been set to T and an object with the given name already exists. Typing "OK" will cause the new object to be created anyway.

---

Can't name object NIL

---

Explanation: The *name* argument to the method **SetName** has been left out.

---

*name* should be a symbol to be a name

---

Explanation: The method **SetName** has been given a non-symbolic name.

---

*name* cannot be a class name. Type OK to ignore

---

Explanation: A non-symbolic class name has somehow gotten into the CLASSES of a file.  
Typing "OK" will continue writing the file, but will not remove the offending name.

---

Can't rename a class without specifying name.  
Type RETURN <newName> to continue and rename class: *self*

---

Explanation: The *newName* argument has been left out of **Rename**. Classes can not be named NIL.

Typing "RETURN 'aNewName" renames the class.

---

*name* not defined as a class or an instance. Type OK to ignore and go on.

---

Explanation: A name which refers to a nonexistent class or instance is in the **CLASSES** or **INSTANCES** file command of a file.

Typing "OK" continues writing out the file, but does not remove the offending name.

*name* not the name of an instance! Type OK to proceed.

---

Explanation: A name that refers to a nonexistent instance is in the **THESE-INSTANCES** file command of a file.

Typing "OK" continues writing out the file, but does not do anything to correct the source of the problem; that is, it does not remove the name from the filecoms or find out why it does not exist.

*name* is a defined object, but is not a class.

---

Explanation: The name of some LOOPS object that is not a class has been used as an argument where a class name should have been used.

## 11.2.4 Annotated and Active Values

---

This section describes errors that occur when using annotated values and active values.

Active value not found, so can't replace it.

---

Explanation: The old active value specified in **ReplaceActiveValue** does not exist or has been specified incorrectly.

Unknown access type *type*

---

Explanation: An improper *type* has been given to the message **AddActiveValue** or **DeleteActiveValue**.

Invalid type *type*

---

Explanation: An active value has an incorrect type specifier.

Conflicting active value wrapping precedence *self activeValue otherPrecedence*

---

Explanation: An attempt has been made to add an annotated value with wrapping precedence T or NIL to an existing annotated value with the same wrapping precedence.

Unknown access type *type*

---

Explanation: **GetWrappedValue** or **PutWrappedValue** has been given an incorrect type.

Can't set the local state of *#.NotSetValue*

---

Explanation: **PutWrappedValueOnly** has been erroneously sent to a *#.NotSetValue*.

## 11.2.5 Miscellaneous

---

This section describes other errors that can occur when using LOOPS.

Use one of METHODS IVS CVS for type. RETURN one of these symbols to go on.

---

Explanation: An incorrect type has been specified to the method **WhereIs**.

To continue, enter the type into the break window. For example, enter "RETURN 'METHODS'".

---

*Name* not installed because of error in source

---

Explanation: The source specification of a class has been corrupted in some way. It may be necessary to manually redefine the class or edit the file.

---

Time is not set! Call (**SETTIME** dd-mmm-yy hh:mm:ss) and then type in OK

---

Explanation: LOOPS uses the date and time to create unique internal names for objects; thus, the time must be set before any objects are created. Call **SETTIME** and then type "OK". For example, (**SETTIME** "15-APR-87 12:00:00") sets time at noon on April 15, 1987.

---

*self varName propName* not broken. Type OK to go on

---

Explanation: Either an attempt has been made to unbreak a value which was not broken or the value was specified incorrectly.

[This page intentionally left blank]

## 12. BREAKING AND TRACING

A number of functions and methods are available in the LOOPS environment to facilitate the process of finding and correcting bugs in user-written LOOPS code. These give you the capability to interrupt or trace methods so that you can examine the state of the computations by using the Interlisp-D Break package; see the *Interlisp-D Reference Manual*.

In addition to being able to break and trace methods, you can break and trace accesses to data within objects. For example, you can determine when a process is attempting to change a class variable or is trying to read the value of an instance variable. This feature gives you a powerful tool to assist in the understanding of the behavior of a piece of code from both a functional view and a data view.

### 12.1 Breaking and Tracing Methods

The Interlisp-D environment provides a number of features for breaking and tracing functions. LOOPS methods are implemented as Lisp functions, so the breaking and tracing of method invocation is similar to Interlisp-D.

The following table describes the methods in this section.

| Name                      | Type   | Description                                                     |
|---------------------------|--------|-----------------------------------------------------------------|
| <b>BreakMethod</b>        | Method | Breaks a method of a class.                                     |
| <b>TraceMethod</b>        | Method | Traces a method of a class.                                     |
| <b>UnbreakMethod</b>      | Method | Unbreaks or untraces a method of a class.                       |
| <b>SelectorsWithBreak</b> | Method | Returns a list of selectors whose implementations have a break. |

(← *self* **BreakMethod** *selector*) [Method of Class]

Purpose: Breaks a method of a class.

Behavior: Varies according to the argument.

- If *selector* is NIL, a menu appears showing the selectors associated with the class *self* that have not already been broken. If you do not make a choice from the menu, this method returns the symbol **NothingBroken**.
- If *selector* is non-NIL and is not associated with *self*, an error occurs stating that *selector* was not found in *self*.

If a method is broken, this fact is printed in the Prompt Window. The broken method function is added to the list **BROKENFNS**. (See the *Interlisp-D Reference Manual* for more information on **BROKENFNS**.)

Arguments: *self* Must be bound to a class.



*selector*      Must be a selector associated with *self* or NIL.

Returns:      The symbol **NothingBroken** or NIL.

Categories:    Class

Example:      The following command causes a break when the message **Open** is sent to any window:

```
12← (← ($ Window) BreakMethod 'Open)
```

---

(← *self* **TraceMethod** *selector*) [Method of Class]

---

Purpose:      Traces a method of a class.

Behavior:      Varies according to the argument.

- If *selector* is NIL, a menu appears showing the selectors associated with the class *self* that have not already been broken. If you do not make a choice from the menu, this method returns the symbol **NothingTraced**.
- If *selector* is non-NIL and is not associated with *self*, an error occurs stating that *selector* was not found in *self*.

If a method is traced, this fact is printed in the Prompt Window. The traced method function is added to the list **BROKENFNS**. (See the *Interlisp-D Reference Manual* for more information on **BROKENFNS**.) Whenever the function is called a message will be printed to a trace window, when it is exited a message will be printed with the returned value.

Arguments:    *self*      Must be bound to a class.

*selector*      Must be a selector associated with *self* or NIL.

Returns:      The symbol **NothingTraced** or NIL.

Categories:    Class

---

(← *self* **UnbreakMethod** *selector*) [Method of Class]

---

Purpose:      Unbreaks or untraces a method of a class.

Behavior:      Varies according to the argument.

- If *selector* is NIL, a menu appears showing the selectors associated with the class *self* that have been broken. If you do not make a choice from the menu, this method returns the symbol **NothingUnbroken**.
- If *selector* is non-NIL and is not associated with *self*, an error occurs stating that *selector* was not found in *self*.

If a method is unbroken, its method function is removed the list **BROKENFNS**. (See the *Interlisp-D Reference Manual* for more information on **BROKENFNS**.) The value return is a list containing the name of the unbroken method function.

Arguments:    *self*      Must be bound to a class.

*selector*      Must be a selector associated with *self* or NIL.

Returns:      The symbol **NothingUnbroken** or a list containing the name of the unbroken method function.

Categories:    Class

(← *self* **SelectorsWithBreak**)

[Method of Class]

- Purpose: Return a list of selectors whose implementations have a break.
- Behavior: Searches through the list **BROKENFNS** collecting all selectors of method functions that begin with the class name of *self*. (See the *Interlisp-D Reference Manual* for more information on **BROKENFNS**.)
- Arguments: *self* Must be bound to a class.
- Returns: A list of selectors of *self*.
- Categories: Class

## 12.2 BREAKING AND TRACING DATA

## 12.2 BREAKING AND TRACING DATA

## 12.2 Breaking and Tracing Data

Breaking or tracing functions or methods cause interruptions to occur in a computation when a function or method is entered. Breaks or traces on data can be made to occur when either the data is to be read or changed. Only data that is contained within objects can be broken; this feature is not available to arbitrary Lisp data. Breaks and traces on data are implemented through the mechanism of active values. The following **ActiveValue** classes contain this mechanism:

- **BreakOnPut**
- **BreakOnPutOrGet**
- **TraceOnPut**
- **TraceOnPutOrGet**

You can use the methods and functions in this section to place or remove breaks on data. You can also add and remove traces and breaks through the inspector interface. See Chapter 18, User Input/Output Modules, for more information on the inspector.

Note: Breaking or tracing a variable effectively breaks or traces any **IndirectVariable** that points to it.

The following table describes the items in this section.

| Name                   | Type            | Description                                       |
|------------------------|-----------------|---------------------------------------------------|
| <b>BreakIt</b>         | Method          | Puts a break on data within an object.            |
| <b>BreakIt</b>         | Function        | Sends the message <b>BreakIt</b> to <i>self</i> . |
| <b>TraceIt</b>         | Method          | Puts a trace on data within an object.            |
| <b>TraceIt</b>         | Function        | Sends the message <b>TraceIt</b> to <i>self</i> . |
| <b>UnBreakIt</b>       | Function        | Unbreaks broken data; untraces traced data.       |
| <b>BrokenVariables</b> | Global Variable | Contains a list of broken or traced variables.    |

(← *self* **BreakIt** *varName propName* &OPTIONAL (type 'IV) *breakOnGetAlsoFlg*)

[Method of Object]

Purpose: Puts a break on data within an object.

Behavior: Adds an entry to the list **BrokenVariables**.

- If *breakOnGetAlsoFlg* is T, creates an instance of the class **BreakOnPutOrGet** and adds the active value to the data specified by *self*, *varName*, *propName*, and *type*.
- If *breakOnGetAlsoFlg* is NIL, the active value instance is of the class **BreakOnPut**.

When a break occurs, the break window shows the nature of the break and which object and what variable is broken. See examples below.

Arguments: *self* Points to the object that contains the data to be broken.

*varName* The name of the variable.

*propName* If a property access is to be broken, this is the name of the property.

*type* The type of the data. This can be IV, CV, or METHOD; the default is IV.

*breakOnGetAlsoFlg*  
If this is non-NIL, breaks will occur when data is read. If this is NIL, breaks will occur only on attempts to write the data.

Returns: *self*

Categories: Object

Example: The following commands check if a window's width and height are going to change.

```
(← ($ Window) New 'w)
(← ($ w) BreakIt 'width)
(← ($ w) BreakIt 'height NIL NIL T)
```

Trying to change the width causes this break:

```
Setting IV
Setting IV width of #,($ w)
Old value: #,NestedNotSetValue
Value: 100
28+;
```

Trying to read the height causes this break:

```
Fetching
Fetching IV height of #,($ w)
Value: 12
28+;
```

---

**(BreakIt self varName propName type breakOnGetAlsoFlg)**

[Function]

Behavior: Sends the message **BreakIt** to *self* passing the remainder of the arguments. See the method **BreakIt**, above, for details.

(← *self* **TraceIt** *varName propName &OPTIONAL (type 'IV) traceGetAlsoFlg*) [Method of Object]

**Purpose:** Puts a trace on data within an object.

**Behavior:** Adds an entry to the list **BrokenVariables**.

- If *traceGetAlsoFlg* is T, creates an instance of the class **TraceOnPutOrGet** and adds the active value to the data specified by *self*, *varName*, *propName*, and *type*.
- If *traceGetAlsoFlg* is NIL, the active value instance is of the class **TraceOnPut**.

When a trace occurs, a trace window appears if necessary, with the traced information printed in it. See examples below.

**Arguments:** *self* Points to the object that contains the data to be traced.

*varName* The name of the variable.

*propName* If a property access is to be traced, this is the name of the property.

*type* The type of the data. This can be IV, CV, or METHOD; the default is IV.

*traceOnGetAlsoFlg*

If this is non-NIL, trace messages will occur when data is read. If this is NIL, trace messages will occur only on attempts to write the data.

**Returns:** *self*

**Categories:** Object

**Examples:** To monitor if a window's width and height are going to change, enter

```
97←(← ($ Window) New 'w)
#,($& Window (NEW0.1Y%:.;h.eN6 . 495))

98←(← ($ w) TraceIt 'width)
#,($& Window (NEW0.1Y%:.;h.eN6 . 495))

99←(← ($ w) TraceIt 'height NIL NIL T)
#,($& Window (NEW0.1Y%:.;h.eN6 . 495))
```

Trying to change the width or the height causes a trace.

```
100←(change (@ ($ w) width) 100)
```

```
Trace-Output
Setting IV width of #,($ w)
Old value: #,NestedNotSetValue
Value: 100
```

```
100
```

```
101←(@ ($ w) height)
```

```
Trace-Output
Fetching IV height of #,($ w)
Value: 12
```

12

---

**(TraceIt self varName propName type breakOnGetAlsoFlg)** [Function]

Purpose/Behavior: Sends the message **TraceIt** to *self* passing the remainder of the arguments. See the method **TraceIt**, above, for details.

---

**(UnBreakIt self varName propName type)** [Function]

Purpose: Unbreaks broken data; untraces traced data.

Behavior: Varies according to the argument *self*.

- If *self* is NIL, iterates through the list **BrokenVariables** and removes the active values from the objects on that list. **BrokenVariables** is set to NIL.
- If *self* is not NIL, removes the active value from the data described by *self*, *varName*, *propName*, and *type*. The corresponding entry is removed from **BrokenVariables**. If there is no active value on the specified data, a break occurs saying that the specified data is not broken and type OK to continue.

Arguments: *self* Points to the object that contains the data to be traced.

*varName* The name of the variable.

*propName* If a property access is to be traced, this is the name of the property.

*type* The type of the data. This can be IV, CV, or METHOD; the default is IV.

Returns: Value depends on the arguments.

- If *self* is NIL, the value of **BrokenVariables** before it was bound to NIL is returned.
- If *self* is non-NIL and there were no errors, the list containing *self*, *varName*, and *propName* is returned.

Example: The following command removes a break from the instance variable **id#** in the instance named **Datum12**:

```
(UnBreakIt ($ Datum12) 'id#)
```

---

**BrokenVariables** [Global Variable]

Purpose/Behavior: This is initialized to NIL. As data within objects is traced or broken, an entry is added to this list. Each entry contains the object, the variable name, the active value created to implement the break, the property name, and the type.

[This page intentionally left blank]

LOOPS has an interface to the display-based editor, SEdit. This editor is most often used for modifying classes, functions, and methods, but it can also be invoked to modify instances. Instances are typically modified through the inspector interface (see Chapter 18, User Input/Output Modules).

The technique for editing methods is exactly the same as that for editing functions. LOOPS uses Method, an extension of the lambda form that is documented in Chapter 6, Methods, which also gives details about the form of methods and how to invoke the editor upon them.

The process of editing classes and instances is different from editing methods in that you are not editing structure directly. The data structures representing the objects are translated into list structures, those list structures are then edited, and, finally, on exiting from the editor, the list structure is translated back into LOOPS objects. Because of this process, changes in objects do not take effect until you have exited from the editor.

## 13.1 Editing Classes

The **Edit** and **Edit!** methods provide a screen-based way to modify class structure. You can quickly add and delete local class and instance variables, make inherited variables local, and change initial values. The other methods listed are used to interface LOOPS to the display editor.

| Name                      | Type   | Description                                                                   |
|---------------------------|--------|-------------------------------------------------------------------------------|
| <b>Edit</b>               | Method | Edits the structure of a class.                                               |
| <b>Edit!</b>              | Method | Edits a class in a form that includes inherited information.                  |
| <b>InstallEditSource</b>  | Method | Makes a class conform to a description.                                       |
| <b>MakeEditSource</b>     | Method | Makes a list structure for editing a class.                                   |
| <b>MakeFullEditSource</b> | Method | Makes a list structure, including inherited information, for editing a class. |

(← *self* **Edit** *commands*)

[Method of Class]

Purpose: Edits the structure of a class.

Behavior: Translates *self* from a data type to a list structure that is then passed to the editor through **EDITE** (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*). If *commands* is non-NIL, this is passed as the second argument to **EDITE**.

The variable **LASTCLASS** is bound to the class name of *self*.

Generally, *commands* is NIL, which causes you to enter the editor interactively. From this point, you can perform the following actions:

- Change the metaclass of *self*
- Add or delete class properties
- Add or delete class variables, instance variables, and their properties.

Also, as a user convenience, the edited form has a list of methods that you can select and edit, although you cannot delete items from this **MethodFns** list and have that action disassociate the methods from the class.

Arguments: *commands* A list of editing commands to be passed to EDITE.

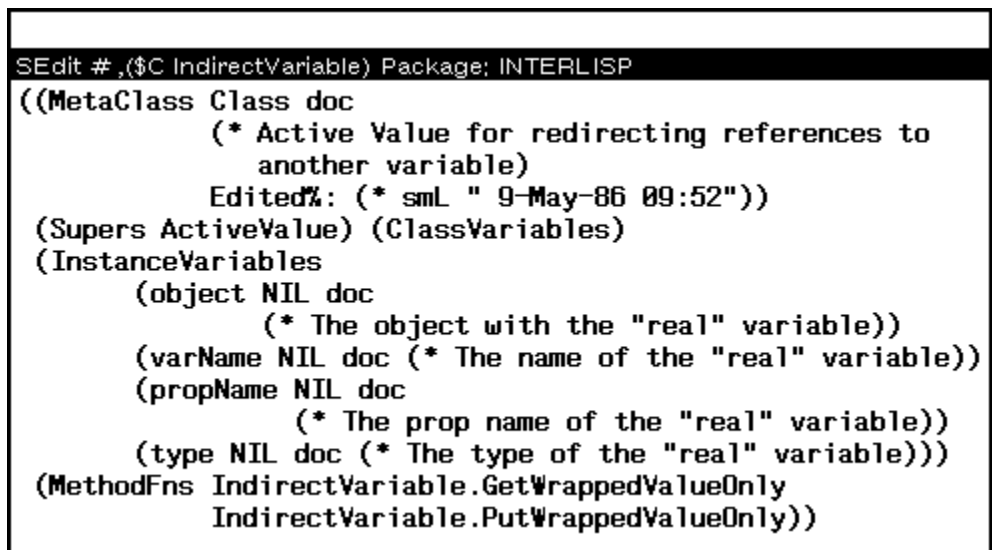
Returns: The class name of *self*.

Categories: Object

Specializes: Object

Example: The following editing window is generated as a result of

```
(← ($ IndirectVariable) Edit)
```



```
SEdit #,($C IndirectVariable) Package: INTERLISP
((MetaClass Class doc
 (* Active Value for redirecting references to
 another variable)
 Edited%: (* smL " 9-May-86 09:52"))
(Supers ActiveValue) (ClassVariables)
(InstanceVariables
 (object NIL doc
 (* The object with the "real" variable))
 (varName NIL doc (* The name of the "real" variable))
 (propName NIL doc
 (* The prop name of the "real" variable))
 (type NIL doc (* The type of the "real" variable)))
(MethodFns IndirectVariable.GetWrappedValueOnly
 IndirectVariable.PutWrappedValueOnly))
```

The following information describes this window:

- The title of the window contains the name of the class being edited and package it uses for displaying symbols. This package should be INTERLISP when using LOOPS.
- It has the metaclass **Class**.
- It has the two class properties **doc** and **Edited%:**.
- It has one super class, the class **ActiveValue**.
- It has no class variables.
- It has four instance variables: **object**, **varName**, **propName**, and **type**. Each instance variable has a **doc** property.
- It has two local methods: **GetWrappedValueOnly** and **PutWrappedValueOnly**.



(← *self* **Edit!** *commands*)

[Method of Class]

Purpose: Edits a class in a form that includes inherited information.

Behavior: This is similar is behavior to the method **Class.Edit**.

The form you are editing includes not only items local to *self* but also class variables and instance variables that are inherited. This allows you to easily move inherited information into *self*. Editing operations that modify the inherited values have no effect.

Arguments: *commands* A list of editing commands to be passed to **EDITE**.

Returns: The class name of *self*.

Categories: Class

Example: The following command puts you into the display editor. Compare this display with the previous one.

```
(← ($ IndirectVariable) Edit!)
```

```
SEdit #,($C IndirectVariable) Package: INTERLISP
((MetaClass Class doc
 (* Active Value for redirecting references to
 another variable)
 Edited%: (* sml " 9-May-86 09:52"))
(Supers ActiveValue) (ClassVariables) (CVsInherited)
(InstanceVariables
 (object NIL doc
 (* The object with the "real" variable))
 (varName NIL doc (* The name of the "real" variable))
 (propName NIL doc
 (* The prop name of the "real" variable))
 (type NIL doc (* The type of the "real" variable)))
(IVsInherited))
```

(← *self* **InstallEditSource** *editedDescription*)

[Method of Class]

Purpose: Makes a class conform to a description.

Behavior: Called by the system to change a class data structure to correspond to a list structure you have edited. If there are errors in the structure, the editor is activated again. If there are errors in the edited structure, an error message is printed in the prompt window and you are returned to the editor to fix it.

If there are no errors in the structure, this successfully translates the structure into the class data type structure. In addition, a class property **Edited:** is added to *self* with the value returned by (**EDITDATE** NIL INITIALS).

Arguments: *editedDescription*  
A list structure similar to that returned by the message **MakeEditSource**.

Returns: Used for side effect only.

Categories: Object

Specializes: Object

**(← self MakeEditSource)**

[Method of Class]

Purpose: Makes a list structure for editing a class.

Behavior: This builds a list structure containing metaclass, super, class variable and instance variable information. In addition, the method function names are included in this list.

Returns: List expression of class structure.

Categories: Object

Specializes: Object

Example: The command

```
(← ($ BreakOnPutOrGet) MakeEditSource)
```

returns

```
((MetaClass Class Edited%: (* nbm " 5-May-87 17:53")
 doc "This is the default metaClass for all classes")
 (Supers BreakOnPut)
 (ClassVariables)
 (InstanceVariables)
 (MethodFns BreakOnPutOrGet.GetWrappedValue))
```

**(← self MakeFullEditSource)**

[Method of Class]

Purpose: Makes a list structure, including inherited information, for editing a class.

Behavior: This is similar to **MakeEditSource**. The constructed list also includes instance variables and class variables that are inherited.

The list does not contain the method functions associated with self that **MakeEditSource** includes.

Returns: List expression of class structure.

Categories: Class

Example: The command

```
(← ($ BreakOnPutOrGet) MakeFullEditSource)
```

returns:

```
((MetaClass Class Edited%: (* nbm " 5-May-87 17:53")
 doc "This is the default metaClass for all classes")
 (Supers BreakOnPut)
 (ClassVariables)
 (CVsInherited)
 (InstanceVariables)
 (IVsInherited
 (localState NIL doc (* The local state of the active value))))
```

13.1 EDITING INSTANCES

---

**13.1 EDITING INSTANCES**

---

## 13.2 Editing Instances

LOOPS instances can also be edited by the standard Medley code editor. From this editor, you can change the values of instance variables and properties, and add new instance variables. Instances follow the same basic editing protocol that classes do.

The following table shows the methods in this section.

| Name                     | Type   | Description                                               |
|--------------------------|--------|-----------------------------------------------------------|
| <b>Edit</b>              | Method | Allows you to change the values contained in an instance. |
| <b>InstallEditSource</b> | Method | Makes an instance conform to a description.               |
| <b>MakeEdit</b>          | Method | Makes a list structure for editing an instance.           |

(← *self* **Edit** *commands*)

[Method of Object]

Purpose: Allows you to change the values contained in an instance.

Behavior: Changes the data structure of *self* to a list and passes that list to **EDITE** (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*.) If *commands* is non-NIL, this is passed as the second argument to **EDITE**.

Deleting a variable does not delete it from the instance.

Arguments: *commands* A list of editing commands to be passed to **EDITE**.

Returns: *self*

Categories: Object

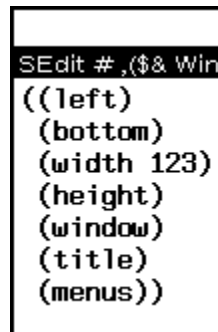
Specializations: Class

Example: The following commands create the edit window shown.

```
71←(← ($ Window) New 'w1)
#,($& Window (NEW0.1Y%:.;h.eN6 . 495))

72←(←@ ($ w1) width 123)
123

73←(←($ w1) Edit)
```



(←*self* **MakeEditSource**)

[Method of Object]

Purpose: Makes a list structure for editing an instance.

|                  |                                                                                                                |
|------------------|----------------------------------------------------------------------------------------------------------------|
| Behavior:        | Returns a list showing all instance variables, values, and properties.                                         |
| Returns:         | A list showing all instance variables, values, and properties.                                                 |
| Categories:      | Object                                                                                                         |
| Specializations: | Class                                                                                                          |
| Example:         | The following shows <b>MakeEditSource</b> results as values are assigned to the instance variables of (\$ w1). |

```
38←(← ($ Window) New 'w1)
#,($& Window (NEW0.1Y%:.;h.eN6 . 495))

39←(← ($ w1) MakeEditSource)
((left) (bottom) (width) (height) (window) (title) (menus))

40←(←@ ($ w1) width 123)
123

41←(← ($ w1) MakeEditSource)
((left) (bottom) (width 123) (height) (window) (title) (menus))

42←(←@ ($ w1) menus Title T)
T

43←(← ($ w1) MakeEditSource)
((left) (bottom) (width 123) (height) (window) (title) (menus
#,NotSetValue Title T))

44←(@ ($ w1) window)
{WINDOW}#74,25554

45←(← ($ w1) MakeEditSource)
((left 31) (bottom 407) (width 123) (height 12) (window #,($AV
LispWindowAV ((NEW0.1Y%:.;h.eN6 . 495)) (localState {WINDOW}#74,25554)))
(title) (menus #,NotSetValue Title T))
```

---

(← *self* **InstallEditSource** *editedDescription*)

[Method of Object]

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| Purpose:    | Makes an instance conform to a description.                                                                          |
| Behavior:   | This is called by the system to change an instance data structure to correspond to a list structure you have edited. |
| Arguments:  | <i>editedDescription</i><br>A list structure similar to that returned by the message <b>MakeEditSource</b> .         |
| Returns:    | Used for side effect only.                                                                                           |
| Categories: | Object                                                                                                               |

[This page intentionally left blank]

LOOPS data structures are fully integrated into Medley. This includes the definition of new File Manager commands so that any LOOPS object or method can be saved on files and loaded into the environment in exactly the same way that normal Medley data types are saved and loaded.

In addition, the LOOPS file browser provides a menu-driven interface to the File Manager. When using a LOOPS file browser, newly created objects are associated with files automatically. If you are not familiar with LOOPS file browsers see Chapter 10, Browsers.

This chapter describes the functions, methods, and variables used to load and store files containing LOOPS objects. It describes the File Manager commands related to LOOPS objects. It also describes how to add objects to files, delete them from files, and move them from file to file. These are primarily of interest when customizing either the File Manager or LOOPS file browser.

---

## 14.1 Manipulating Files

---

LOOPS takes advantage of the ability to create user-defined File Manager commands to fully integrate LOOPS into the Medley environment. As a result, the same steps used to manipulate files containing Medley data structures are used to manipulate files containing LOOPS data structures. Furthermore, both LOOPS and Medley data structures can be saved together in the same file. This section contains a brief review of the three basic functions used to manipulate files. For a more detailed description which includes additional functions, see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*.

In addition, there is a LOOPS file browser which provides a convenient way of loading files and guaranteeing that newly created classes and methods are associated with files during the development of LOOPS programs. The LOOPS file browser is different from the Lisp Library Module **FILEBROWSER**. Files can be loaded and put into new or existing file browsers by a series of menu selections.

You can manipulate files with these basic steps:

- Assign data structures to a specific file using **FILES?**.
- Write data structures to a file using **MAKEFILE**.
- Enter data structures stored in a file into the environment using **LOAD**.

The following example shows these steps.

```
30← (FILES?)
PIPEANDTANK, LOOPSPRINT, LOOPSUTILITY...to be dumped.
 plus the instances: FFAV1, Datum1, TestW
 plus the class definitions: Datum
 want to say where the above go ? Yes
(instances)
FFAV1 File name: LOOPSFILE
create new file LOOPSFILE ? Yes
Datum1 File name: LOOPSFILE
TestW File name: LOOPSFILE
(class definitions)
Datum File name: LOOPSFILE
NIL

31← (MAKEFILE 'LOOPSFILE)
Copyright owner for file LOOPSFILE: XEROX
{DSK}<LISPFILS>LOOPSFILE.;1

32← (LOAD 'LOOPSFILE)
{DSK}<LISPFILS>LOOPSFILE.;1
FILE CREATED 7-Jan-87 16:25:24
LOOPSFILECOMS
{DSK}<LISPFILS>LOOPSFILE.;1
```

See the *Lisp Release Notes* and the *Interlisp-D Reference Manual* for more information on **FILES?** and **MAKEFILE**. See the following section for details on **LOAD**.

14.2 Loading Files

The following table shows the functions and commands described in this section.

| Name    | Type        | Description                                                                                                            |
|---------|-------------|------------------------------------------------------------------------------------------------------------------------|
| LOAD    | Function    | Loads Medley symbolic files which includes LOOPS objects and methods.                                                  |
| LOADFNS | Function    | Allows selective loading from Medley symbolic files.                                                                   |
| UNDO    | Prog. Asst. | Undoes previous entries into the Medley Executive which are stored on a history list, including calls to <b>LOAD</b> . |

(LOAD FILE LDFLG ) [Function]

|                   |                                                                                                                                                           |                                                                                                                                                                                                                                                                                                     |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose/Behavior: | Loads Medley symbolic files which includes all LOOPS objects and methods; see the <i>Lisp Release Notes</i> and the <i>Interlisp-D Reference Manual</i> . |                                                                                                                                                                                                                                                                                                     |
| Arguments:        | FILE                                                                                                                                                      | File to be loaded.                                                                                                                                                                                                                                                                                  |
|                   | LDFLG                                                                                                                                                     | Alters the effect of loading a file. <ul style="list-style-type: none"><li>If it is set to <b>PROP</b>, the definitions of functions, including <b>METHOD</b> functions, are stored on the property <b>EXPR</b> of the function name. Thus, any existing definitions are not overwritten.</li></ul> |

- If it is set to **ALLPROP**, the values of variables are also saved on property lists.

Returns: Full file name.

**(LOADFNS FNS FILE)**

[Function]

Purpose/Behavior: Allows selective loading from Medley symbolic files including LOOPS files. The most likely use for this facility is to load the source code for method functions when the compiled versions are already loaded. The methods must be specified by their explicit function names in the form **ClassName.Selector**, for example,

```
(LOADFNS ' (SomeClass.AMethod OtherClass.AMethod) ' {DSK}<LISPFILES>SOMEFILE ' PROP)
```

It is not recommended that LOOPS objects be selectively loaded by using **VARs** (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*), because it is not possible to guarantee that all necessary related objects, such as superclasses or methods of a class, are also loaded.

Arguments: **FNS** Selected functions to be loaded.  
**FILE** File from which functions specified in **FNS** are to be loaded.

Returns: List of functions that have been loaded

**UNDO**

[Program Assistant Command]

Purpose/Behavior: LOOPS saves enough information about objects that are created as a result of loading a file to allow the call to **LOAD** to be undone. The objects are destroyed and any preexisting objects that were deleted by the load are restored. See the *Lisp Release Notes* and the *Interlisp-D Reference Manual*.

## 14.3 LOOPS FILE PACKAGE COMMANDS

## 14.3 LOOPS FILE PACKAGE COMMANDS

**14.3 LOOPS File Manager Commands**

Four File Manager types are defined to allow LOOPS objects to be stored in Medley files:

- CLASSES
- METHODS
- INSTANCES
- THESE-INSTANCES

These types and the functions and methods used by LOOPS to process these types are described in this section.

Note: The order of items in the filecoms is important. In particular, class definitions must appear in the file before any methods on that class or any instances of that class. Similarly, methods on a class must appear before any instances of that class.

| Name              | Type             | Description                                                                                          |
|-------------------|------------------|------------------------------------------------------------------------------------------------------|
| <b>CLASSES</b>    | File Mgr Command | Writes the appropriate <b>DEFCLASSES</b> and <b>DEFCLASS</b> expressions for the named classes.      |
| <b>DEFCLASSES</b> | NLambda NoSpread | Creates a series of empty classes in preparation for reading their definitions via <b>DEFCLASS</b> . |



|                        |                     |                                                                                                                           |
|------------------------|---------------------|---------------------------------------------------------------------------------------------------------------------------|
| <b>DEFCLASS</b>        | NLambda<br>NoSpread | Takes a source specification of a class from a file and causes the appropriate internal representation to be constructed. |
| <b>METHODS</b>         | File Mgr<br>Command | Writes the appropriate <b>METH</b> and <b>DEFINEQ</b> expressions for each method object and its associated function.     |
| <b>METH</b>            | NLambda<br>NoSpread | Creates a method object and attaches it to the appropriate class.                                                         |
| <b>INSTANCES</b>       | File Mgr<br>Command | Writes the appropriate <b>DEFINST</b> expressions for each instance in the list.                                          |
| <b>THESE-INSTANCES</b> | File Mgr<br>Command | Appears as a sublist in a filecoms.                                                                                       |
| <b>DEFINSTANCES</b>    | NLambda<br>NoSpread | Creates empty structures for each instance name in a list.                                                                |
| <b>DEFINST</b>         | NLambda<br>NoSpread | Creates internal representations for source specifications of an instance.                                                |
| <b>FileIn</b>          | Method              | Creates internal representations for source specifications of an instance.                                                |

---

**(CLASSES *ClassName1...ClassNameN*)** [File Manager Command]

Purpose/Behavior: Appears as a sublist in a filecoms. The keyword **CLASSES** tells the File Manager to use the appropriate **DEFCLASSES** and **DEFCLASS** expressions for the named classes when writing to a file.

Arguments: *ClassName* Accepts any symbol, but only gives meaningful result when you use **DEFCLASS** to actually create the class.

Example: (CLASSES Myclass)

---

**(DEFCLASSES CLASSES)** [NLambda NoSpread Function]

Purpose/Behavior: Used in a file to create a series of empty classes in preparation for reading in their definitions via **DEFCLASS**. This allows the classes to be read in any order. Otherwise, superclasses would have to be read in before their subclasses.

Arguments: **CLASSES** Accepts any symbol, but only gives meaningful result when you use **DEFCLASS** to actually create the class.

Returns: NIL

Example: The command  
(DEFCLASSES MyClass)  
returns NIL.

---

**(DEFCLASS FORM)** [NLambda NoSpread Function]

Purpose/Behavior: Takes a source specification of a class, such as produced by the method **MakeFileSource**, from a file and causes the appropriate internal representation to be constructed.

Arguments: **FORM** The source specification of a class.

Returns: NIL

Example: 

```
(DEFCLASS MyClass
 (MetaClass Class doc (* Something for my project)
 Edited: (* nbm "18-Oct-87 13:20"))
 (Supers Object)
 (InstanceVariables (Iv1 (22) doc
 (* Initial value for my instances])
```

**(METHODS** *ClassName.Message1...ClassName.MessageN*) [File Manager Command]

Purpose/Behavior: Appears as a sublist in a filecoms. The keyword **METHODS** tells the File Manager to use the appropriate **METH** and **DEFINEQ** expressions for each method object and its associated function.

Arguments: *ClassName.Message*  
The source specification of a class.

Example: 

```
(METHODS MyClass.Method1)
```

**(METH** *methDescr*) [NLambda NoSpread Function]

Purpose/Behavior: Creates a method object and attaches it to the appropriate class.

Arguments: *methDescr* Method object to create.

Returns: NIL

Example: 

```
(METH MyClass MyClass.Method1 NIL
 (category (Datum)))
```

**(INSTANCES** *InstName1...InstNameN*) [File Manager Command]

Purpose/Behavior: Appears as a sublist in a filecoms. The keyword **INSTANCES** tells the File Manager to use the appropriate **DEFINST** expressions for each instance in the list and also for any other instances that are referenced inside any instances in the list. This assures that there are no references to nonexistent instances when read back in. The method **SaveInstance?** can be specialized to prevent instances from being saved in more than one file when they are referred to by instances in different files.

Example: 

```
(INSTANCES TestW)
```

**(THESE-INSTANCES** *InstName1...InstNameN*) [File Manager Command]

Purpose/Behavior: Appears as a sublist in a filecoms. The keyword **THESE-INSTANCES** tells the File Manager to use the appropriate **DEFINST** expressions for each instance in the list. Unlike the **INSTANCES** File Manager command, **THESE-INSTANCES** does not recursively dump instances that are pointed by *InstName1...InstNameN*.

**(DEFINSTANCES *Instances*)**

[NLambda NoSpread Function]

Purpose/Behavior: Takes a list of instance names and creates empty structures for them in preparation for reading in their structures from a file.

Arguments: *Instances* Accepts any symbol but result is useless unless you use **DEFINST** to actually create the *Instance*.

Returns: NIL

Example: (DEFINSTANCES TestW)

**(DEFINST *DEFINST% FORM*)**

[NLambda NoSpread Function]

Purpose/Behavior: Takes a source specification of an instance and causes the appropriate internal representation to be created. It does this by sending the message **FileIn** to the instance's class. It creates the class if it does not exist.

Arguments: *DEFINST% FORM*  
The source specification of an instance.

Returns: NIL

Example: [DEFINST Window  
(TestW (JEW0.0X:.H<4.NZ9 . 532))  
(left 179)  
(bottom 446)  
(width 12)  
(height 12)]

**(← *self FileIn fileSource*)**

[Method of Class]

Purpose/Behavior: Takes a source specification for an instance as it appears in a file and causes the appropriate internal representation to be constructed.

Arguments: *self* Class of the instance to be created.  
*fileSource* Loadable form of an instance as stored in a file.

Returns: *self*

Categories: Class

14.4 SAVING LOOPS OBJECTS ON FILES

---

14.4 SAVING LOOPS OBJECTS ON FILES

---

---

**14.4 Saving LOOPS Objects on Files**

---

Adding LOOPS classes, methods and instances to files can be done in the same way that functions and variables are saved in Medley. In addition, the LOOPS browser allows newly created objects to be automatically associated with files. LOOPS also provides the means for moving objects from file to file.

Whenever a class, method, or named instance is created or edited, it is marked as changed. This allows the File Manager to prompt for a file in which to store new objects and see to it that changed objects are written out when **MAKEFILE** is called.

The following table shows the items in this section.

| Name                  | Type       | Description                                                                                                                                          |
|-----------------------|------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FILES?</b>         | Function   | LOOPS adds a prompt for classes, methods and instances along with the normal Medley types.                                                           |
| <b>ObjectModified</b> | Method     | Notifies the File Manager that an object has been changed or created.                                                                                |
| <b>OnFile</b>         | Method     | Determines if a class is in <b>FILELST</b> .                                                                                                         |
| <b>SaveInstance</b>   | Method     | Causes newly created instances to be noticed by the File Manager.                                                                                    |
| <b>SaveInstance?</b>  | Method     | Determines if an instance needs to be added to the list of instances to be saved.                                                                    |
| <b>DelFromFile</b>    | Method     | Deletes an object from any file in <b>FILELST</b> in which it appears.                                                                               |
| <b>MoveToFile</b>     | Method     | <b>Class.MoveToFile</b> moves a class and its methods from one file to another. <b>Object.MoveToFile</b> moves an instance from one file to another. |
| <b>MoveToFile!</b>    | Method     | Moves a class, all of its methods, and all of its subclasses and their methods from one file to another.                                             |
| <b>DontSave</b>       | IVProperty | Controls what parts of an instance are saved in a file.                                                                                              |
| <b>OldInstance</b>    | Method     | Sends a message to an object after it is loaded from a file.                                                                                         |

**(FILES?)**

[Function]

**Purpose/Behavior:** The File Manager types have been extended so that, when a call is made to **FILES?**, you are prompted to add classes, methods and instances to files along with the normal Medley. For an example of **FILES?**, see Section 14.1, "Manipulating Files."

After a class is associated with a file, any methods that are added to it are automatically added to that file as well. Thus, it makes sense to put classes in files as soon as possible. This could be done by repeated calls to **FILES?**, but the LOOPS file browser allows classes to be automatically added to files as they are created. Any class that is created by adding a root to a file browser or by specializing a class in a file browser is added to that browser's file. If more than one file is associated with the browser, a menu appears to prompt you to specify a file for the new class. The LOOPS browser also can be used to create a new file and associate it with a file browser. Thus, there is never any need to wait until the end of a session to put classes and methods in files.

You can also save instances on files. Of course, only those instances which should be present after a file is first loaded should be saved. Instances which are constructed "on the fly" as a consequence of running a LOOPS program should not be saved. Only named instances are marked as changed so many such temporary instances may never be noticed. However, if named instances which should not be saved are created, then you are prompted to put them into files after a call to **FILES?** and must respond by typing a right square bracket (]) to each one. Alternatively, it is possible to specialize the method **ObjectModified** so that it does not call **MARKASCHANGED**. Then any instances of classes which have or inherit the specialized method are not noticed by the File Manager regardless of whether or not they are named.

**(← self ObjectModified name)**

[Method of Object]

**Purpose:** Notifies the File Manager that an object has been changed or newly created.

|                  |                                                                                                                                          |                                                                                                                          |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| Behavior:        | Uses the File Manager command <b>MARKASCHANGED</b> . It does nothing if <i>name</i> is not given, thus unnamed objects are never marked. |                                                                                                                          |
| Arguments:       | <i>self</i>                                                                                                                              | A LOOPS object.                                                                                                          |
|                  | <i>name</i>                                                                                                                              | Name of object specified in <i>self</i> .                                                                                |
|                  | <i>reason</i>                                                                                                                            | Reason is <b>MARKEDASCHANGED</b> (see the <i>Interlisp Reference Manual</i> for information on <b>MARKEDASCHANGED</b> ). |
| Returns:         | <i>self</i>                                                                                                                              |                                                                                                                          |
| Categories:      | Object                                                                                                                                   |                                                                                                                          |
| Specializations: | Method                                                                                                                                   |                                                                                                                          |

---

(← *self* **OnFile** *file*)

[Method of Class]

|             |                                                                                                                                                                                                                                                                                                                                                                                                                           |                          |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------|
| Purpose:    | Determines if an object is in a file in <b>FILELST</b> .                                                                                                                                                                                                                                                                                                                                                                  |                          |
| Behavior:   | Calls <b>WHEREIS</b> (see the <i>Lisp Release Notes</i> and the <i>Interlisp-D Reference Manual</i> ). <ul style="list-style-type: none"><li>• If <i>file</i> is not given, it returns the name of the file in <b>FILELST</b> that the object is contained in or NIL if <i>self</i> is not in a file.</li><li>• If <i>file</i> is given, it must still be a member of <b>FILELST</b>, and T or NIL is returned.</li></ul> |                          |
| Arguments:  | <i>self</i>                                                                                                                                                                                                                                                                                                                                                                                                               | A LOOPS object.          |
|             | <i>file</i>                                                                                                                                                                                                                                                                                                                                                                                                               | The file to be searched. |
| Returns:    | Value depends on the arguments; see <b>Behavior</b> .                                                                                                                                                                                                                                                                                                                                                                     |                          |
| Categories: | Class                                                                                                                                                                                                                                                                                                                                                                                                                     |                          |

---

(← *self* **SaveInstance** *name* *reason*)

[Method of Object]

|             |                                                                   |                                                                                                                          |
|-------------|-------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| Purpose:    | Causes newly created instances to be noticed by the File Manager. |                                                                                                                          |
| Behavior:   | Sends <i>self</i> the message <b>ObjectModified</b> .             |                                                                                                                          |
| Arguments:  | <i>self</i>                                                       | A LOOPS object.                                                                                                          |
|             | <i>name</i>                                                       | Name of object specified in <i>self</i> .                                                                                |
|             | <i>reason</i>                                                     | Reason is <b>MARKEDASCHANGED</b> (see the <i>Interlisp Reference Manual</i> for information on <b>MARKEDASCHANGED</b> ). |
| Returns:    | <i>self</i>                                                       |                                                                                                                          |
| Categories: | Object                                                            |                                                                                                                          |

---

(← *self* **SaveInstance?** *file* *outInstances*)

[Method of Object]

|           |                                                                                                                                                                                                                                                                                                                         |  |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| Purpose:  | Determines whether an instance needs to be added to the list of instances to be saved in <i>file</i> .                                                                                                                                                                                                                  |  |
| Behavior: | Checks to see if the current instance is a member of <i>outInstances</i> . It is used by the LOOPS File Manager command <b>INSTANCES</b> to guarantee that the same instance does not appear more than once in a given file. <p>This method must be specialized to be used; it cannot be used directly by the user.</p> |  |

Arguments: *self* A LOOPS object.  
*file* The file to be searched.  
*outInstances* A list of LOOPS names. See Behavior.

Returns: T if the instance should be saved on the file; NIL if it should not be saved.

Categories: Object

---

(← *self* **DelFromFile**) [Method of Object]

Purpose: Deletes an object from any file in **FILELST** in which it appears.

Behavior: Searches through the filecoms of all files in **FILELST** and deletes the object everywhere it appears.

Arguments: *self* A LOOPS object.

Returns: Used for side effect only.

Categories: Object

Specializations: Class, Method

---

(← *self* **MoveToFile** *file*) [Method of Class]

Purpose: Moves an object from one file to another. If an object is a class, it, and all its methods, move.

Behavior: Adds the object to the filecoms of *file* so that the object will be saved on that file. If *file* is NIL, it prompts for a file form **FILELST** via a menu.

Arguments: *self* A class or method.  
*file* File to which object is moving.

Returns: NIL

Categories: Object

Specializes: Object

---

(← *self* **MoveToFile!** *file fromFiles*) [Method of Class]

Purpose: Moves a class, all of its methods, and all of its subclasses and their methods from one file to another.

Behavior: Similar to **MoveToFile**.

Arguments: *self* A LOOPS class.  
*file* File to which object is moving.  
*fromFiles* A list of files from which classes may be moved.

Returns: NIL

Categories: Class

---

**DontSave** [IV Property Name]

Purpose/Behavior: Controls what parts of an instance are saved in a file. Its value is a list of property names of the instance variable which should not be written out when

the instance is dumped. If **Value** is in the list, the instance variable's value is not saved. If the property is **Any**, nothing is saved except the instance variable name. (Must be added by the user.)

|                                                          |                                                                                                                                                                                                                                                                                                                                                     |                                                                    |
|----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------|
| (←self <b>OldInstance</b> name arg1 arg2 arg3 arg4 arg5) |                                                                                                                                                                                                                                                                                                                                                     | [Method of Object]                                                 |
| Purpose:                                                 | Sends a message to an object after it is loaded from a file. This method can be specialized by applications that need to perform some operation on every object when it is created.                                                                                                                                                                 |                                                                    |
| Behavior:                                                | If <i>name</i> is non-NIL, the message <b>SetName</b> is sent to <i>self</i> .<br><br>Instance variables with an <b>:initForm</b> property are filled. See the discussion of <b>:initForm</b> in Chapter 2, Instances.<br><br>Sends the message <b>SaveInstance</b> to <i>self</i> with the arguments <i>name</i> , <i>arg1</i> , and <i>arg2</i> . |                                                                    |
| Arguments:                                               | <i>self</i>                                                                                                                                                                                                                                                                                                                                         | Evaluates to a class.                                              |
|                                                          | <i>name</i>                                                                                                                                                                                                                                                                                                                                         | LOOPS name of the class or instance.                               |
|                                                          | <i>arg1...arg5</i>                                                                                                                                                                                                                                                                                                                                  | Optional arguments referenced by user-written specialization code. |
| Categories:                                              | Object                                                                                                                                                                                                                                                                                                                                              |                                                                    |
| Specializations:                                         | IndexedObject                                                                                                                                                                                                                                                                                                                                       |                                                                    |

## 14.5 Storing Files

This section describes the functions and methods used by LOOPS and Medley to store files.

| Name                       | Type     | Description                                                                            |
|----------------------------|----------|----------------------------------------------------------------------------------------|
| <b>MAKEFILE</b>            | Function | Writes files that contain Medley data types which include LOOPS objects and methods.   |
| <b>PrettyPrintClass</b>    | Function | Prints classes in a file in a form that can be read back in.                           |
| <b>PrettyPrintInstance</b> | Function | Prints instances in a file in a form that can be read back in.                         |
| <b>MakeFileSource</b>      | Method   | Constructs the representation of an object that is appropriate for printing in a file. |
| <b>FileOut</b>             | Method   | Controls the printing of a LOOPS object in a file.                                     |

|                   |                                                                                                                                                                                                                                                                                      |                                 |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------|
| (MAKEFILE FILE )  |                                                                                                                                                                                                                                                                                      | [Function]                      |
| Purpose/Behavior: | When all LOOPS objects are associated with their files, the files are written by a call to <b>MAKEFILE</b> or <b>MAKEFILES</b> . This is identical to the standard use of <b>MAKEFILE</b> in Medley. See the <i>Lisp Release Notes</i> and the <i>Interlisp-D Reference Manual</i> . |                                 |
| Arguments:        | FILE                                                                                                                                                                                                                                                                                 | Name of file to be written out. |

Returns: Full file name

**(PrettyPrintClass *className file*)**

[Function]

Purpose/Behavior: Used by the File Manager command **CLASSES** to print out classes in a file in a form that can be read back in. It checks to make sure the class exists and then sends it the message **FileOut**. It is also used by the method **PP** to print classes to a display stream.

Arguments: *className* The name of the class to be printed on the file *file*.  
*file* The file on which the class *className* is to be printed.

Returns: Pointer to class in the form #,(\$ *className*)

**(PrettyPrintInstance *instanceName file*)**

[Function]

Purpose: Used by the File Manager command **INSTANCES** to print instances in a file in a form which can be read back in. Sends the message **FileOut** to instance.

Arguments: *instanceName* Name of a LOOPS instance.  
*file* The file on which the instance *instancename* is to be printed.

Returns: NIL

**(← self MakeFileSource *file*)**

[Method of Object]

Purpose: Constructs the representation of an object that is appropriate for printing in a file.

Behavior: Uses the relevant access functions to obtain the parts of the object and then stores them into a list structure.

Arguments: *self* A LOOPS object.  
*file* The file on which *self* is to be printed.

Returns: Loadable form of a LOOPS object.

Categories: Object

Specializations: Class, Method

Example: 63← (← (\$ TestW) MakeFileSource)  
 (DEFINST Window  
 (TestW (NEW0.1Y%.:;h.eN6 . 501)))

**(← self FileOut *file*)**

[Method of Object]

Purpose: Controls the printing of a LOOPS object in a file.

Behavior: Gets the appropriate source representation by sending the object the message **MakeFileSource** and prettyprints the result.

Arguments: *self* A LOOPS object.  
*file* The file on which *self* is to be printed on if T prints to the Lisp Executive window.

Returns: *self*

Categories: Object



Specializations:    Class, Method

Example:

```
62_(_ ($ TestW) FileOut T)
(DEFINST Window (TestW (NEW0.1Y%:.;h.eN6 . 501)))
#,$& TestW (NEW0.1Y%:.;h.eN6 . 501))
```

14.6 COMPILING FILES

---

## 14.6 COMPILING FILES

---

---

## 14.6 Compiling Files

---

LOOPS uses the new XAIE compiler and its macrolet facilities. When doing **CLEANUP** on LOOPS files your **\*DEFAULT-CLEANUP-COMPILER\*** should be set to 'CL:COMPILE-FILE. More information on this cleanup flag and the new compiler are available in the *Lisp Release Notes*.

[This page intentionally left blank]

## 15. PERFORMANCE ISSUES

---

Three main areas in LOOPS can affect performance:

- Garbage collection
- Instance variable access
- Method lookup

This chapter describes the impact of these areas on LOOPS. Also included is a section on cache clearing.

---

### 15.1 Garbage Collection

---

The Interlisp-D garbage collector maintains reference counts of each piece of data in the system. (Refer to the *Lisp Release Notes* and the *Interlisp-D Reference Manual* for information on reference counts.) There is potential for noticeable performance degradation if many items have reference counts greater than one. Object-oriented systems in general, and LOOPS in particular, can easily create objects that have multiple references.

The LOOPS system uses a number of methods to avoid creating items with large reference counts. Classes, for example, can easily have large reference counts since each instance of the class points to the class. Because of this, LOOPS does not maintain reference counts of classes. Performance is enhanced, but classes in LOOPS are not garbage collected. This should not present a problem as classes are not often destroyed.

Unique Identifiers (UIDs) also have multiple references: from the instance they name and from the table used by the LOOPS system to associate UIDs with instances. LOOPS avoids this problem by storing copies of the instance UID in the instance. This complicates testing for equality of UIDs, which is a rare event, but removes a potential garbage collection problem.

These and other implementation details substantially reduce the impact of LOOPS on the Interlisp-D garbage collector. In a typical running system, LOOPS objects accounted for less than 16% of the data items with reference count greater than one.

15.2 INSTANCE VARIABLE ACCESS

---

### 15.2 INSTANCE VARIABLE ACCESS

---

---

### 15.2 Instance Variable Access

---

LOOPS uses macros to speed the instance variable access from compiled code. Instance variable property access is compiled differently from instance variable value access, and various caching schemes are used to speed up repeated access to a given slot.

LOOPS uses two layers of caching to speed up instance variable access:

- Local cache.

Instance variable access from compiled code uses a local cache. This cache remembers the class of *self* and the instance variable index the last time this piece of code was executed. If the class of *self* on the next pass through the code matches the stored value, then the stored instance variable index is used. In this case, instance variable access is very fast.

- Global cache.

A global cache is used by the instance variable access functions when the local cache fails. This global cache is a fixed size table of instance variable pairs. Looking in this cache for a given class is typically faster than computing the instance variable index.

You should be aware that instance variable access is optimized to be faster than accessing the properties of instance variables. Also, be aware that when instances are first created, the data for an instance variable may need to be found by performing a lookup through the class hierarchy. If the lookup goes through several classes, this can be slow. By guaranteeing that the instance variable data is stored in the instance, this lookup delay can be avoided.

The following macros are used to access instance variables. They are mentioned here to point out that calls to **GetValue** and **PutValue** could result in the compilation of any one of several different functions.

---

**(GetValue self varName &OPTIONAL propName)** [Macro]

Purpose/Behavior: Compiles to a call to one of the functions **Cached-GetIVValue**, **Cached-GetIVProp**, **GetIVValue**, or **GetIVProp**. The particular function depends on details of the arguments to **GetValue**.

Arguments: *self* A class or an instance.  
*varName* Instance or class variable name.  
*propName* Property name.

Returns: Used for side effect only.

---

**(PutValue self varName value &OPTIONAL propName)** [Macro]

Purpose/Behavior: Compiles to a call to one of the functions **Cached-PutIVValue**, **Cached-PutIVProp**, **PutIVValue**, or **PutIVProp**. The particular function depends on details of the arguments to **PutValue**.

Arguments: *self* A class or an instance.  
*varName* Instance or class variable name.  
*value* The new value for *varName* or *propName*.  
*propName* Property name.

Returns: Used for side effect only.

15.3 METHOD LOOKUP

---

## 15.3 METHOD LOOKUP

---

---

## 15.3 Method Lookup

---

LOOPS uses two layers of caching to speed the method lookup:

- Local cache

Method lookup from compiled code uses a local cache when the selector can be determined at compile time. This cache remembers the class of *self* and the computed method the last time this message was sent. If the class of *self* on the next pass through the code matches the stored value, then the method is used. In this case, method lookup is very fast.

- Global cache

A global cache is used by the method lookup functions when the local cache fails. This global cache is a fixed size table of class / selector / method triples. Looking in this cache for a given class and selector is typically faster than searching the class hierarchy for the appropriate method.

15.4 CACHE CLEARING

---

## 15.4 CACHE CLEARING

---

---

## 15.4 Cache Clearing

---

Code that directly manipulates the structure of LOOPS objects sometimes needs to invalidate the caches used for instance variable access and message sending.

The following functions can be used to clear these caches if you suspect that they might be invalid.

---

**(ClearAllCaches)****[Function]**

Purpose/Behavior: Clears all LOOPS and Interlisp-D runtime caches. This includes local and global instance variable access caches, local and global method lookup caches, and the system CLISP translations hash array.

Returns: NIL

[This page intentionally left blank]

LOOPS provides two special versions of message sending that start a separate process to run LOOPS methods. These are `←Process` and `←Process!` which are analogous to `←` and `←!`.

`(←Process obj sel arg1 ... argn)`

[Macro]

Purpose: Starts a new process to run the selected method on the object, *obj*.

Behavior: The method indicated by *sel* is run in a separate process for the given instance or class, *obj*. See the *Interlisp-D Reference Manual* for a discussion of processes.

Arguments: *obj* A LOOPS object.  
*sel* Name of the method to be executed as a process.  
*arg1 ... argn* Arguments for the method specified in *sel*.

Returns: Pointer to a process data type.

Example: Assume the method **ClockTime** is added to the class **LCD**, as follows:

```
[Method ((LCD ClockTime
self WaitTime DisplaySeconds?)
(while T
 do (←@ self reading
 [MKATOM (DATE (if DisplaySeconds?
 then (DATEFORMAT NO.DATE)
 else (DATEFORMAT NO.DATE NO.SECONDS])
 (← self Update)
 (BLOCK (OR WaitTime 1000]))
(LCD.ClockTime)
```

**ClockTime** takes two arguments: **WaitTime**, the wait time between updates of the **LCD** reading, and **DisplaySeconds?**, a flag used to determine if seconds are to be displayed on the **LCD**. **ClockTime** runs an infinite loop which sets the **LCD** reading, updates the **LCD** display, and blocks the **ClockTime** loop to allow other system processes to run. The command

```
(←Process ($ LCDInstance1) ClockTime 60000)
```

adds the process **ClockTime** to the process list and (**\$ LCDInstance1**) becomes a digital clock which updates itself every minute.

**19:33**

**(←Process! obj sel arg1 .... argn)**

[Macro]

- Purpose:** Starts a new process to run the selected method on the object *obj*. Like ←**Process**, except the argument *sel* is evaluated.
- Behavior:** Evaluates *sel* returns a selector for a method of *obj*. This method is run on a separate process for the given instance or class, *obj*.
- Arguments:** *obj*            A LOOPS object.  
*sel*                Name of the method to be executed as a process.  
*arg1 .... argn*            Arguments needed for the method.
- Returns:** Pointer to the process data type.
- Example:** Assume the variable **LCDClock** is set to **ClockTime**, which is the method added to the **LCD** class as described for ←**Process**. The command
- ```
(←Process! ($ LCDInstance1) LCDClock 2000 T)
```
- adds the process **LCDClock** to the process list and (**\$ LCDInstance**) becomes a digital clock with a seconds display which updates itself every two seconds.

19:49:39

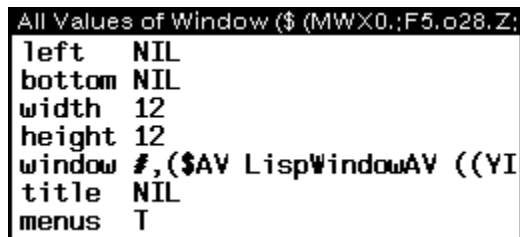
18. USER INPUT/OUTPUT MODULES

This chapter presents two modules that have been developed for displaying and allowing you to enter information. The Interlisp-D Inspector module and ?= handler have been enhanced to support LOOPS data types and message sending.

18.1 Inspector

The LOOPS interface uses and extends the capabilities of the Medley Inspector module. Instances and classes can be easily examined and modified through the interface that the Inspector module provides. This section describes the operations available with the LOOPS interface. For information on the Inspector module, see the *Interlisp-D Reference Manual*.

An inspector is a window opened on a specific piece of data, which for LOOPS means a class or an instance. Figure 18-1 shows an inspector on an instance of a window.



All Values of Window (\$ (MWX0.;F5.o28.Z:	
left	NIL
bottom	NIL
width	12
height	12
window #	(\$AV LispWindowAV ((YI
title	NIL
menus	T

Figure 18-1. Sample Inspector

Inspector windows contain two columns of information; the left column is called the property column, and the right column is called the value column.

You can scroll inspector windows, but these windows are not reshaped by actions such as switching from instance variables inspection to property inspection or adding new instance variables. This may cause some confusion, for example, if you create an inspector to be the correct size and add an instance variable, and that instance variable fails to appear.

An inspector is primarily an interactive facility. A programmatic interface is also available, which uses the LOOPS methods to customize the generic Interlisp-D functionality.

18.1.1 Overview of the User Interface

LOOPS provides two ways to create an inspector:

- Call the Lisp function **INSPECT** with the object to be inspected as the first argument.

- Use the LOOPS method **Inspect**, which is described below.

The user interface to the inspector is the same as that for the Medley environment; that is, you select an option from the left or right column with the left mouse button, and then trigger an action with the middle mouse button. The action opens a menu from which you can choose further options, like assigning a new value or adding an active value for breaking.

Another menu appears when you position the cursor on the title bar of an inspect window and press the middle mouse button. This menu allows you to change the inspector's contents, for instance to show all values or local ones for instance variables.

Three types of inspectors are available in LOOPS:

- Instance inspector
- Class inspector
- Class instance variable inspector

The following sections describe the user interface for each inspector.

<code>(← self Inspect <i>INSPECTLOC</i>)</code>	<code>[Method of Object]</code>

Purpose:	This provides a message form of the function to inspect the item <i>self</i> .
Behavior:	Calls <code>(INSPECT self NIL <i>INSPECTLOC</i>)</code> .
Arguments:	<i>INSPECTLOC</i> A region where the inspector window should appear. If it is NIL, you are prompted to place a ghost image.
Categories:	Object
Example:	The following command inspects an instance (\$ W1). <code>17← (← (\$ W1) Inspect)</code>

18.1.2 Using Instance Inspectors

Using an inspector window on an instance provides a clear, direct interface to all of the instance's variables and values. This interface also provides the mouse and keyboard options to change the contents of the inspector to show various aspects of the instance.

18.1.2.1 Titles of Instance Inspector Windows

When you inspect an object, the title of the inspector window reflects the contents of the inspector. If the object is an instance, the title contains the name of the class of the instance and the LOOPS name of the instance, if it has one, or the UID of the instance. Other types of inspectors have different title bars, as described in Section 18.1.3, "Using Class Inspectors," and Section 18.1.4, "Using ClassIV Inspectors."

Contrast the title bar of the following two examples.

`(INSPECT (← ($ Window) New))` generates

```

All Values of Window ($ (MWX0.:F5.o28.Z;
left    NIL
bottom  NIL
width   12
height  12
window  #,($AV LispWindowAV ((YI
title   NIL
menus   T

```

(INSPECT (← (\$ Window) New 'w1)) generates

```

All Values of Window ($ w1).
left    NIL
bottom  NIL
width   12
height  12
window  #,($AV LispWindowAV ((YI
title   NIL
menus   T

```

18.1.2.2 Menu for the Title Bar

The following menu appears when you position the cursor on the title bar of the inspector window and press the middle mouse button.

```

Local Values of Window ($ w1).
left    468
bottom  472
width   116
height  74
window  #,($AV LispWindowAV ((|MWX0.:F5
title   #,NotSetValue
menus   #,NotSetValue

```

The rest of this section describes the actions that occur as a result of selecting one of the menu options.

Class Opens a second inspector, a Class Inspector as described in Section 18.1.3, "Using Class Inspectors," which inspects the class of the instance within this inspector.

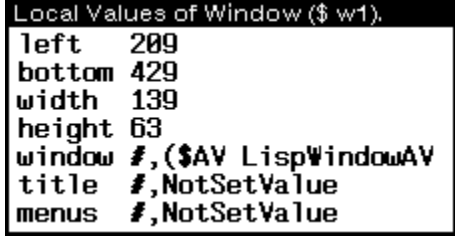
AllValues The default mode for instance inspectors. The values displayed in the right column of the inspector are determined by the function **GetValueOnly**, so active values (except #,NotSetValue) can be seen. The title of the inspector states that all values are being displayed.

```

All Values of Window ($ w1).
left
  NIL
bottom
  NIL
width
  12
height
  12
window
  #,($AV LispWindowAV ((YIV0.C=N5.W←7 . 10)))
title
  NIL
menus
  T

```

LocalValues The values displayed in the right column of the inspector are determined by the function **GetIVHere**. The title of the inspector states that only local values are being displayed. As with **AllValues**, active values are seen, and values that are not yet stored locally in the instance show a value of **#,NotSetValue**.



```
Local Values of Window ($ w1).
left      200
bottom    429
width     139
height    63
window    #, ($AV LispWindowAV
title     #,NotSetValue
menus     #,NotSetValue
```

Add/Delete Allows you to add or delete instance variables. Selecting this option pops up a new menu with two options:

- **Add**

If you select **Add**, you are prompted to enter a name for the new instance variable. That instance variable is added locally to the instance and given the value of the variable **NotSetValue**. If you enter a name for an instance variable that currently exists, its value is reset to the value defined in the class.

- **Delete**

If you select **Delete**, a menu appears with options that are the instance variables of the instance. If you select one that is not defined within the class, it is deleted. If the selected instance variable is defined by the class, a break occurs.

If the inspector is viewing the properties of an instance variable as opposed to all of the instance variables (see **IVs** below), the name entered under **Add** will be added as a property to that instance variable and given the value of the variable **NotSetValue**. If you try to delete an existing property, the menu that appears is a menu of property names.

IVs Changes the view to be one that shows all of the instance variables and their values, not the properties. It is possible to change the view an inspector has on an instance to show only the properties of a given instance variable. This is described in Section 18.1.2.3, "Using Commands in the Left Column," in the description of the Properties option.

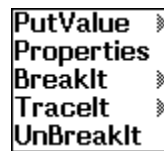
Save Value Calls **PutSavedValue** with its value argument bound to the instance being inspected.

Refetch Refreshes the inspector. Inspectors do not automatically update when a change is made to an instance, unless made with the **Edit** command.

Edit Opens a display editor window in which you can modify the value of instance variables and properties, and add or delete instance variables local to the instance.

18.1.2.3 Menu for the Left Column

The following menu appears when the view of the inspector is all of the instance variables of the instance, and you select an item in the left column and press the middle mouse button.



If the view of the inspector is only of properties, this menu contains only one option: **PutValue**.

The rest of this section describes the actions that occur as a result of selecting one of the menu options.

PutValue Allows you to assign a new value to the variable selected. Selecting this option and dragging the mouse to the right causes a submenu with the following options to appear:

- **PutValue**

Prompts you to enter a new value for this instance variable. The new value is stored using **PutValue**.

- **PutValueOnly**

Prompts you to enter a new value for this instance variable. The new value is stored using **PutValueOnly**.

- **Use saved value**

The new value to be stored using **PutValueOnly** is the value of **(SavedValue)**.

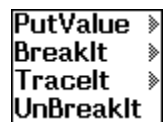
Properties Changes the view of the inspector to include only the value and properties of the selected instance variable as shown here:

All IVProps of Window (\$ w1).menus	
Value	T
doc	"Cache For Saved Menus. Will Cache
TitleItems	NIL
MiddleButtonItem	NIL
LeftButtonItem	NIL
Title	NIL
DontSave	Any

The title bar changes to indicate that the properties of an instance variable are being displayed, which instance is being displayed, and which instance variable of that instance is being displayed.

The Value item is provided purely as a convenience in this view and its menu options will only allow Putting a new value in it.

The following menu appears when the view of the inspector is an instance variable's properties, and you select an item in the left column and press the middle mouse button.



When the inspector's view is limited to IV properties the menu options act in a manner similar to that of IVs. This allows Putting, Breaking, Tracing and unBreaking of the properties instead of the IVs themselves.

If you now select the option **LocalValues** from the title bar menu, the title of the inspector changes to indicate that fact, and only properties that are stored locally in the instance appear, as shown here:



To return to a view that shows all instance variables, choose the **IVs** option from the title bar menu.

BreakIt Wraps a **BreakOnPutOrGet** active value around the value of an instance variable. Any read or write accesses to this instance variable will cause a break (see Chapter 12, Breaking and Tracing).

Note: Breaking a variable effectively breaks any IndirectVariable that points to it.

Selecting this option and dragging the mouse to the right causes a submenu with the following options to appear:

- **Break on Access**

Performs the same action as **BreakIt**.

- **Break on Put**

Installs a **BreakOnPut** active value. Trying to store a new value into this instance variable will cause a break, but reading the variable will not.

TraceIt Wraps a **TraceOnPutOrGet** active value around the value of this field. Any read or write accesses to this instance variable will be traced (see Chapter 12, Breaking and Tracing).

Note: Tracing a variable effectively traces any IndirectVariable that points to it.

Selecting this option and dragging the mouse to the right causes a submenu with the following options to appear:

- **Trace on Access**

Performs the same action as **TraceIt**.

- **Trace on Put**

Installs a **TraceOnPut** active value. All writes into this instance variable will be traced, but reads will not.

UnBreakIt Removes any of the breaks or traces that have been installed on an instance variable. If there are multiple traces or breaks, this will remove the outermost one.

18.1.2.4 Menu for the Right Column

The following menu appears when the view of the inspector is all of the instance variables of the instance, and you select an item in the right column and press the middle mouse button.



If the view of the inspector is only of properties of an instance variable, this menu contains only three options: **PutValue**, **Save Value**, and **Inspect**.

The only differences between these menus and the ones associated with the left column is the addition of two more options: **Save Value** and **Inspect**. The remaining options trigger identical behaviors as those of the menu associated with the left column.

Save Value Calls **PutSavedValue** with the selected value as its argument.

Inspect Calls the Lisp function **INSPECT** with the selected value as its argument, opening an additional inspector window.

In an inspector viewing the properties of an IV the right hand column middle button menu allows only the Put Value, SaveValue and Inspect options.

18.1.3 Using Class Inspectors

Classes can be inspected by using the Lisp **INSPECT** function or the LOOPS **Inspect** method (see Section 18.1.1, "Overview of the User Interface"). For example, to inspect the class **Window**, enter either of the following commands:

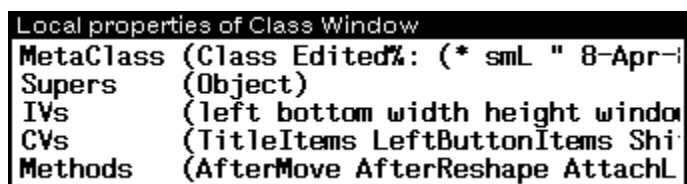
```
(INSPECT ($ Window))
(← ($ Window) Inspect)
```

The contents of a class cannot be changed from within an inspector window, so it is generally used for display as opposed to editing. However, the menu interface does provide ways to edit the contents of a class.

18.1.3.1 Titles of Class Inspector Windows

When you inspect a class, the title states that you are inspecting only local properties, and contains the name of the class.

The title contains the name of the class. The following example shows an inspector on the class **Window**. Since the value column is quite long, it has been truncated here.



18.1.3.2 Menu for the Title Bar

The title bar menu is associated with each inspector of a class. This menu appears when you position the cursor inside the title bar of the class inspector window and press the middle mouse button.



The rest of this section describes the actions that occur as a result of selecting one of the menu options.

Browse Provides a quick way to open a class browser on the class being inspected. Selecting this option and dragging the mouse to the right pops up a submenu with the following options:

- **Browse**

Opens a class browser with the class being inspected as the root class.

- **BrowseSupers**

Open a supers browser on the inspected class.

Edit Opens an editing window on the class.

All Causes the values shown in the right column to contain inherited as well as locally defined information. The title bar of the inspector changes to indicate this, as shown here:



Local The default mode for class inspectors. This causes the values shown in the right column to contain only locally defined information, which is indicated in the title bar of the inspector.

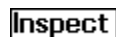
Refetch Refreshes the inspector. Inspectors do not automatically update when a change is made to an instance, unless made with the **Edit** command.

18.1.3.3 Menu for the Left Column

No actions occur when you select an item in the left column of a class inspector and press the middle mouse button.

18.1.3.4 Menu for the Right Column

Only one option, **Inspect**, is in the menu that appears when you select an item in the right column of a class inspector and press the middle mouse button.



For the fields **MetaClass**, **Supers**, **CVs**, and **Methods**, selecting **Inspect** from the menu allows a choice of Interlisp-D inspectors. If the selected item in the class inspector is the values of the **IVs** field, then a ClassIVs inspector, described below, is created.

18.1.4 Using ClassIVs Inspectors

ClassIV inspectors provide an interface to the default values for all of the instance variables defined in a class. To create a ClassIV inspector,

- Open a class inspector.
- Select the values of the **IVs** field.
- Press the middle mouse button. This pops up and selects and **Inspect** menu, and automatically opens a ClassIVs inspector.

18.1.4.1 Titles of ClassIVs Inspector Windows

The title for a ClassIVs inspector indicates that the instance variables of a particular class are being inspected. This example shows how a ClassIVs inspector looks for the class **ClassBrowser**.

All IVs of Class ClassBrowser	
left	NIL
bottom	NIL
width	64
height	32
window	#,(\$AV LispWindowAV ((YIV0.C=N5.W←7 . 10)))
title	"Class browser"
menus	T
topAlign	NIL
startingList	NIL
goodList	NIL
badList	NIL
lastSelectedObject	NIL
browseFont	#, (Defer (FONTCREATE (QUOTE (HELVETICA 10 BOLD)
labelMaxLines	NIL
labelMaxCharsWidth	NIL
boxedNode	NIL
graphFormat	(LATTICE)
showGraphFn	SHOWGRAPH
viewingCategories	(Public)

18.1.4.2 Menu for the Title Bar

The following title bar menu is associated with each inspector of an instance. This menu appears when you position the cursor inside the title bar of the inspector window and press the middle mouse button.

AllValues
LocalValues
Add/Delete
IVs
Refetch

The rest of this section describes the actions that occur as a result of selecting one of the menu options.

AllValues

The default mode for ClassIV inspectors. Causes the inspector to show all instance variables, whether inherited or locally defined for the class, and states "AllIVs" in the title.

LocalValues Causes the inspector to show only locally defined instance variables. The following window shows how the title changes to indicate this:



Add/Delete Allows you to add or delete a ClassIV.

Selecting this option and dragging the mouse to the right causes a submenu with the following options to appear:

- **Add**

If you select **Add**, you are prompted to enter a name for the new instance variable. That instance variable is added to the class and given the default value NIL and a **doc** property with the value (* IV added by (USERNAME)). If you enter a name for an instance variable that currently exists, its default value is reset to NIL.

- **Delete**

If you select **Delete**, a menu appears with the locally defined instance variables of the class. Selecting one deletes it from the class.

If the inspector is viewing the properties of an instance variable as opposed to all of the instance variables (see **IVs** below), the name entered under **Add** is added as a property to that instance variable and given the value NIL. If you try to delete an existing property, the menu that appears is a menu of property names.

IVs Returns the view to show the instance variables and their values, not the properties.

It is possible to change the view a ClassIVs inspector has on the instance variables of a class to show only the properties of a given instance variable. This is described in Section 18.1.4.4, "Menu for the Right Column."

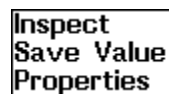
Refetch Refreshes the inspector.

18.1.4.3 Menu for the Left Column

No actions occur if you select an item in the left column of a ClassIVs inspector with the middle mouse button.

18.1.4.4 Menu for the Right Column

The following menu appears when you select an item in the right column and press the middle mouse button:



The rest of this section describes the actions that occur as a result of selecting one of the menu options.

Inspect Calls the Lisp function **INSPECT** with the selected value as its argument.

Save Value Calls **PutSavedValue** with the selected value as its argument.

Properties Changes the view of the inspector to display the value and properties of the selected instance variable, as shown in this example:

```

All properties of window of Class ClassBrowser
Value      #,($AV LispWindowAV ((YIV0.C=N5.W←7 . 1
doc        "Holds real window. Ensured to be window
DontSave   (Value)

```

The title changes to include the following information:

- The properties of an instance variable.
- The name of the instance variable.
- The name of the class.

If you now select either **AllValues** or **LocalValues** from the title menu, the title of the inspector changes to indicate that fact, and the appropriate information is displayed.

18.1.5 Functional Interface for Instance Inspectors

The methods described in this section belong mostly to the classes **Class** or **Object**. Inspectors are not LOOPS objects, so these methods are invoked indirectly within the system functionality of the inspector as a customization of the Interlisp-D inspectors. These methods are meant to be called only from within the context that you create interactively by pressing a mouse button when the cursor is on some portion of an inspector window; you do not invoke them directly. Many of the parameters are simply passed along in case the method creates a menu, and the option selected from the menu needs additional arguments.

In these methods, the arguments *self* and *datum* may be the same; that is, the item being inspected. The message is sent to the item being inspected, so its position in the inheritance lattice determines which method from the classes **Class** or **Object** is invoked.

The following table shows the items in this section.

Name	Type	Description
InspectFetch	Method	Returns the value of a left column inspector property that is displayed in the right column of an inspector window.
InspectStore	Method	Stores the value for an instance variable or its property.
InspectPropCommand	Method	After an item is selected in the left column of an inspector window, this triggers an action when the middle mouse button is pressed.
InspectProperties	Method	Determines what is displayed in the left column of an inspector window.
InspectTitle	Method	Creates a string to be used for an inspector window's title.
InspectValueCommand	Method	After an item is selected in the right column of an inspector window, this triggers an action when the middle button is pressed.
TitleCommand	Method	Triggers an action when the cursor is inside the title bar of the inspector window and the middle mouse button is held down.

(← *self* **InspectFetch** *datum property window*)

[Method of Object]

Purpose/Behavior:	Message sent by inspector to get the value of a left column inspector property that is displayed in the right column of an inspector window. Either GetValueOnly or GetIVHere is used to determine the value.	
Arguments:	<i>self</i>	The object being inspected.
	<i>datum</i>	This may or may not be a list. If it is not a list, it is bound to the object being inspected; that is, <i>self</i>. It is set to a list within various methods associated with the inspectors. The contents of this list are interpreted by a number of the methods to control what data is displayed within the inspector window. • The first element of the list is the object being inspected. • The second element of the list, if not NIL, is typically the name of an instance variable. In the terminology of the inspector, it is an inspector property. For inspectors of instances, the inspector properties (the items in the left column) are the instance variables of the object being inspected. (There can be some confusion here caused by using the word properties either when referring to the left column data of an inspector or when referring to the properties associated with an instance variable). • The third element of the list, if NIL, indicates that inherited values are to be displayed in the inspector window; if its value is LocalValues, then only locally stored information is displayed. The value of <i>datum</i> is stored on the inspector window property DATUM.
	<i>property</i>	Used if <i>datum</i> is not a list. Refers to an element (instance variable or property name) contained within the left column of an inspector. For an instance inspector, this could be either the name of an instance variable or the name of a property, depending upon the state of the inspector; that is, whether you are viewing instances variable or the properties of a particular variable.
	<i>window</i>	Lisp window of the inspector.
Returns:	The value of a left column <i>property</i> that is displayed in the right column.	
Categories:	Object	
Specializations:	Class, InspectorClassIVs	
Example:	The following command fetches the value of instance W1's instance variable bottom: <pre>15←(← (\$ W1) InspectFetch (LIST (\$ W1) 'window)) #,\$AV LispWindowAV ((YIV0.C=N5.W←7 . 10)))</pre> The following command fetches the value of class Window's supers: <pre>16←(← (\$ Window) InspectFetch (\$ Window) 'Supers) (Object)</pre>	

($\leftarrow$ *self* **InspectStore** *datum property newValue window*)

[Method of Object]

Purpose/Behavior: Stores *newValue* as the value for an instance variable or its property using **PutValueOnly**. Where the value is stored, whether in the instance variable or one of its properties, depends upon the values for *datum* and *property*.

Arguments:

- datum* Instance or class being inspected. See **InspectFetch**, above, for details.
- property* Instance variable or property where value is to be stored. See **InspectFetch**, above, for details.
- newValue* New value for *property*.
- window* Lisp window of the inspector.

Categories: Object

Specializations: Class, InspectorClassIVs

Example: The following command changes the value of instance **W1**'s instance variable **height**:

```
17←(← ($ W1) InspectStore ($ W1) 'height 400)
400
```

or

```
18←(← ($ W1) InspectStore 'W1 'height 400)
400
```

($\leftarrow$ *self* **InspectPropCommand** *datum property window*)

[Method of Object]

Purpose: This method is an interface between LOOPS and the mouse functions of the inspector, and should only be called through the inspector. It is invoked when an item is selected in the left column of an inspector window and the middle mouse button is pressed.

Behavior: Opens a menu with a number of options. See Section 18.1.2.3, "Menu for the Left Column."

Arguments:

- datum* Instance or class being inspected. See **InspectFetch**, above, for details.
- property* Instance variable or property where value is to be stored. See **InspectFetch**, above, for details.
- window* Lisp window of the inspector. A prompt window will be attached to this window if you ask to **PutValue**, and the window's **INSPECTW.FETCH** function will be called, so the window must be an inspector window.

Categories: Object

Specializations: Class

($\leftarrow$ *self* **InspectProperties** *datum*)

[Method of Object]

Purpose: Determines what should be displayed in the left column of an inspector.

Behavior: Depending on the value of *datum* as described above, this will return either the instance variables of the object being inspected, or the properties of a particular instance variable.

Arguments:	<i>datum</i>	Instance or class being inspected. See InspectFetch , above, for details.
Returns:	Value depends on the arguments; see Behavior.	
Categories:	Object	
Specializations:	Class, InspectorClassIVs	
Example:	The following command first shows the instance variables of instance W1 , then the properties of the instance variable height : <pre>27←(← (\$ W1) InspectProperties 'W1') (left bottom width height window title menus) 28←(← (\$ W1) InspectProperties (LIST 'W1 'height)) (Value doc)</pre>	

(← self InspectTitle datum)	[Method of Object]

Purpose:	Creates a string to be used as a title for an inspector window.	
Behavior:	If <i>datum</i> is not a list, this sets <i>datum</i> to (<i>datum</i> NIL NIL). Depending on the values within the list <i>datum</i> , this creates a title showing whether all values or local values are shown and whether all instance variables or the properties of an instance variable are shown. The title also contains the LOOPS name or UID of <i>self</i> .	
Arguments:	<i>datum</i>	Instance or class being inspected. See InspectFetch , above, for details.
Categories:	Object	
Specializations:	Class, InspectorClassIVs	
Example:	Some examples of the behavior of InspectTitle : <pre>35←(← (\$ Window) InspectTitle) "Local properties of Class Window" 36←(← (\$ W1) InspectTitle) "All Values of Window (\$ W1)." 37←(← (\$ Window) InspectTitle (LIST 'Window T)) "All properties of Class Window" 38←(← (\$ W1) InspectTitle (LIST 'W1 'height)) "All IVProps of Window (\$ W1).height"</pre>	

(← self InspectValueCommand datum property value window)	[Method of Object]

Purpose:	This method is an interface between LOOPS and the mouse functions of the inspector, and should only be called through the Inspector. It is invoked when an item is selected in the right column of an inspector window and the middle mouse button is pressed.	
Behavior:	Opens a menu with several options. See Section 18.1.2.4, "Menu for the Right Column."	
Arguments:	<i>datum</i>	Instance or class being inspected. See InspectFetch , above, for details.
	<i>property</i>	Instance variable or property being inspected. See InspectFetch , above, for details.

value This is inspected only if you select **Inspect** from menu.

window Lisp window of the inspector.

Categories: Object

Specializations: Class, InspectorClassIVs

Example: When the value of the instance variable **height** in instance **W1** is selected in an inspector window, and the middle mouse button is pressed, a message like the following is sent:

```
(← ($ W1) InspectValueCommand ($ W1) 'height 200 (WHICHW))
```

(← self **TitleCommand** datum window)

[Method of Object]

Purpose: This method is an interface between LOOPS and the mouse functions of the inspector, and should only be called through the Inspector. It is invoked when the cursor is in the title bar of an inspector window and the middle mouse button is pressed.

Behavior: Brings up a menu with several options. See Section 18.1.2.2, "Menu for the Title Bar."

Arguments: *datum* Instance or class being inspected. See **InspectFetch**, above, for details.

window Lisp window of the inspector.

Categories: Object

Specializations: Class, InspectorClassIVs

Example: If you position the cursor inside the title bar of the inspector window for instance **W1** and press the middle mouse button, you send a message like the following:

```
(← ($ W1) TitleCommand NIL (WHICHW))
```

18.1.6 Customizing the Inspector

The methods in Section 18.1.5, "Functional Interface for Instance Inspectors," have been specialized in the classes **Class** and **InspectorClassIVs** to create the behavior of the inspectors described in Section 18.1.1, "Overview of the User Interface."

If you want to create a specialized inspector, you need to create a subclass of **Object** or perhaps **Class** and specialize the methods within that new class. The class **InspectorClassIVs** has an instance variable named **class** that contains the name of the class being inspected within a particular instance of **InspectorClassIVs**. Similarly, the user-created inspector class may need an instance variable which contains the object being inspected so that the methods of this class can easily access it.

The methods that you need to specialize will depend upon how the behavior of the newly created inspector class should differ from those of an instance or class inspector.

As an example, assume that you want an inspector to show a subset of the instance variables of an instance. You could specialize the method **InspectProperties** to return that subset. To make **Window** show only the dimensions of a window, define the following method:

```
SEdit Window.InspectProperties Package; INTERLISP
(Method ((Window InspectProperties) self)
  "Have Window inspectors show only the window dimensions"
  '(width height left bottom))
```

(←New (\$ Window) Inspect) creates the following:

```
All Values of W
width 12
height 12
left NIL
bottom NIL
```

18.2 EXTENSIONS TO ?=

18.2 EXTENSIONS TO ?=

18.2 Extensions to ?=

The Interlisp-D environment allows you to begin typing a function to be evaluated in the Executive and pause in the midst of typing the arguments. At this point, if you type a "?=" followed by a carriage return, Interlisp-D prints the arguments to the pending function call and shows the bindings. LOOPS has extended this facility to include similar functionality for message sending and for record creation.

18.2.1 Message Sending

The ?= interface works with the following message-sending forms:

- ←
- ←Super
- ←New
- ←Proto
- ←Process
- SEND

LOOPS first tries to determine the class of the object receiving the message by examining the form following one of the above. If the message form begins with one of ←**New** or ←**Proto**, the object receiving the message is the class desired.

- If the system cannot determine the class of the object, you are prompted in the Prompt Window to enter in the name of the class or to type a right square bracket (]) to evaluate the form and determine that class from that. This handles cases such as

```
(← (← ($ Window) New) ?=<CR>
```

- If the class can be determined and if you have not typed in a selector, a menu appears containing the options ***generics*** and ***inherited*** and any selectors local to the class. A submenu is associated with ***generics*** that contains selectors from the classes **Tofu**, **Object**, or **Class**, depending

upon the class previously determined. The submenu associated with ***inherited*** are those selectors that are neither generic nor local to the class.

This is the menu that appears when you type

```
(←New ($ NonRectangularWindow) ?= <CR>
```



If you choose one of these options, it is placed into the input buffer and the system prints the binding for *self*, what method will be executed, and the arguments expected. In the prompt window, the system prints the documentation of the method to be executed.

As an example, if you create an instance of a class browser **cb1**, type

```
22← (← ($ cb1) ?= <CR>
```

and then choose **Shape** from the ***inherited*** drag-through menu, the Executive changes as shown here.

```
22← (← ($ cb1) Shape
(←
self = ($ cb1)
Method = Window.Shape
newRegion noUpdateFlg)
```

A similar output occurs if you type in a selector and "!=" instead of choosing a selector from the menu.

If you type "!=" after entering one or more of the arguments, the arguments are printed with the bindings, as shown here.

```
23← (← ($ cb1) Shape ' (100 150 200 250) ?= <CR>
(←
self = ($ cb1)
Method = Window.Shape
newRegion (QUOTE (100 150 ...))
noUpdateFlg)
```

This interface also works when you are typing to the edit buffer window when using the display editor. It does not work to pick a selector within a display editor window and choose the **?=** item from the **EditCom** submenu.

18.2.2 Record Creation

The same mechanism the LOOPS uses to handle **?=** for LOOPS objects is also used to extend it for the Interlisp Record module.

If you begin an input to the Executive with one of **CREATE**, **Create**, or **create**, type the record name or data type to be created next, and then type

```
?=<CR>
```

the system prints the names but not the bindings of the fields within the record being created.

For example, when you type

```
48← (CREATE POSITION XCOORD ← 123 ?=<CR>
```

the response is

```
(XCOORD YCOORD)
```

on the next line. The caret moves to the position of the ?= in the original line, and waits for you to enter a value.

[This page intentionally left blank]

This chapter describes the ways to manipulate LOOPS windows.

19.1 The Class Window

The class **Window** is the LOOPS interface to the Medley environment window system, which is used by LOOPS browsers and inspectors.

Window

[Class]

Description: This class provides a mechanism for manipulating Lisp windows through an object-oriented interface. See Section 19.4, "Mouse and Menu Functionality," for a discussion of how menus work with LOOPS Windows.

When an instance of a LOOPS window is created, it has an instance variable that points to a Lisp window. This Lisp window is initialized with various window properties:

- The property **LoopsWindow** points to the window object.
- The property **RIGHTBUTTONFN** is set to **WindowRightButtonFn**.
- The property **BUTTONEVENTFN** is set to **WindowButtonEventFn**.
- The property **AFTERMOVEFN** is set to **WindowAfterMoveFn**.
- The property **RESHAPEFN** is set to **WindowReshapeFn**.

MetaClass: Class

Supers: Object

Class Variables: **TitleItems** A list that defines the menu that will appear when the left or middle mouse button is pressed and the cursor is in the title bar of the window. The default value is NIL.

LeftButtonItem

A list that defines the menu for the left button in the main window. The default value is ((Update ...)).

ShiftLeftButtonItem

A list that defines the menu for the left button in the main window when the **Meta** key is down. The default value is NIL.

MiddleButtonItem

A list that defines the menu for the middle button in the main window. The default value is NIL.

ShiftMiddleButtonItem

A list that defines the menu for the middle button in the main window when the **Meta** key is down. The default value is NIL.

RightButtonItems

A list that defines the menu for the right button in the main window. The default value is ((Close ...)).

Instance Variables:	left	The location of the left side of the outside of the window in screen coordinates. The default value is NIL.
	bottom	The location of the bottom side of the outside of the window in screen coordinates. The default value is NIL.
	width	The outside width of the window. The default value is 12.
	height	The outside height of the window. The default value is 12.
	window	An active value that contains the Lisp window. The default value is #,\$AV LispWindowAV ...).
	title	The title of the window. Default Value: NIL.
	menus	Also has the properties Title , LeftButtonItems , MiddleButtonItems , and TitleItems . These properties are caches for menus only if the value of the instance variable is T. Default Value: T.

19.2 BASIC WINDOW METHODS

19.2 BASIC WINDOW METHODS

19.2 Basic Window Methods

This section describes the basic methods to operate on windows.

Name	Type	Description
AfterMove	Method	Updates the instance variables left and bottom.
AfterReshape	Method	Updates the instance variables left, bottom, width, and height.
Blink	Method	Causes the window to blink.
Bury	Method	Buries the window.
Clear	Method	Clears the window.
Close	Method	Closes the window.
CursorInside?	Method	Determines if the cursor is inside a window.
Destroy	Method	Destroys the window instance.
GetProp	Method	Gets a property from a specified window.
Hardcopy	Method	Makes a hardcopy on the default device.
HardcopyToFile	Method	Makes a hardcopy on a file.
HardcopyToPrinter	Method	Makes a hardcopy to a printer.
Invert	Method	Inverts the window; that is, reverses its black-white pattern.
Move	Method	Moves the window.

MousePackage	Method	Returns a package (defined to return the INTERLISP package).
MouseReadtable	Method	Returns a readtable (defined to return the INTERLISP readtable).
Open	Method	Opens the window.
Paint	Method	Calls PAINTW on the window.
ScrollWindow	Method	Scrolls the window.
SetProp	Method	Sets the property in the specified window.
Shape	Method	Reshapes the window.
Shape?	Method	Returns the current region for the window.
Shrink	Method	Shrinks the window to an icon.
Snap	Method	Takes a snapshot of the screen.
ToTop	Method	Opens the window and brings it to the top.
Update	Method	Makes the window consistent with the instance variables.
WindowAfterMoveFn	Function	Sends the message AfterMove .
WindowReshapeFn	Function	Sends the message AfterReshape .

(← *self* **AfterMove**) [Method of Window]

Purpose/Behavior: Updates the instance variables **left** and **bottom** of *self*.

Arguments: *self* An instance of a window.

Returns: Used for side effect only.

Categories: Window

(← *self* **AfterReshape** *oldBitmapImage oldRegion oldScreenRegion*) [Method of Window]

Purpose/Behavior: Updates the instance variables **left**, **bottom**, **width**, and **height** of *self*. Calls **RESHAPEBYREPAINTFN**; see the *Interlisp-D Reference Manual*.

Arguments: *self* An instance of a window.

Returns: Used for side effect only.

Categories: Window

(← *self* **Blink** *numBlinks*) [Method of Window]

Purpose/Behavior: Inverts the window; that is, reverses its black-white pattern, and then returns to normal *numBlinks* times.

Arguments: *self* Pointer to a window instance.

numBlinks Number of times for window to blink.

Returns: NIL

Categories: Window

Example: The command

(← *self* Blink 5)

sends a message to *self* to blink five times.

(← *self* **Bury**)

[Method of Window]

Purpose/Behavior: Calls **BURYW** to bury the specified window.

Arguments: *self* Pointer to a window instance.

Returns: The LOOPS window.

Categories: Window

(← *self* **Clear**)

[Method of Window]

Purpose/Behavior: Calls **CLEARW** to clear the specified window.

Arguments: *self* Pointer to a window instance.

Returns: NIL

Categories: Window

Specializations: LatticeBrowser

(← *self* **Close**)

[Method of Window]

Purpose/Behavior: Closes the specified window and prompt window, if there is one.

Arguments: *self* Pointer to a window instance.

Returns: NIL

Categories: Window

(← *self* **CursorInside?**)

[Method of Window]

Purpose/Behavior: Determines if the cursor is inside the window.

Arguments: *self* Pointer to a LOOPS window.

Returns: Returns T if the cursor is inside the window, otherwise returns NIL.

Categories: Window

(← *self* **Destroy**)

[Method of Window]

Purpose/Behavior: Destroys the calling instance, removes all **ButtonFns**, and closes the window.

Arguments: *self* Pointer to a window instance.

Returns: NIL

Categories: Object

Specializes: Object

(← *self* **GetProp** *prop*) [Method of Window]

Purpose/Behavior: Gets a property from a window.

Arguments: *self* Pointer to a window instance.
prop Property to get.

Returns: The value of the specified property, if it exists; else NIL.

Categories: Window

Example: To determine the value of the window property **BUTTONEVENTFN**, enter
 (← (\$ window) GetProp 'BUTTONEVENTFN)

(← *self* **Hardcopy**) [Method of Window]

Purpose/Behavior: Makes a hardcopy of the window on the default printer.

Arguments: *self* Pointer to a window instance.

Categories: Window

(← *self* **HardcopyToFile**) [Method of Window]

Purpose/Behavior: Makes a hardcopy of the window to a file. You are prompted for the file name.

Arguments: *self* Pointer to a window instance.

Categories: Window

(← *self* **HardcopyToPrinter**) [Method of Window]

Purpose/Behavior: Makes a hardcopy of the window on a printer. You are prompted for the name of the printer.

Arguments: *self* Pointer to a window instance.

Categories: Window

(← *self* **Invert**) [Method of Window]

Purpose/Behavior: Inverts the window; that is, reverses its black-white pattern.

Arguments: *self* Pointer to a window instance.

Returns: T if successful.

Categories: Window

Specializations: NonRectangularWindow

(← *self* **Move** *xOrPos* *y*) [Method of Window]

Purpose/Behavior: Moves the specified window. If no arguments are supplied, you will be prompted to position the window.

Arguments: *self* Pointer to a window instance.

x New **left** in screen coordinates or a new position for **left** and **bottom**. If *x* is a position, *y* is ignored.

y New **bottom** in screen coordinates.

Returns: (*x* . *y*)

Categories: Window

Example: The command
 (`← ($ window) Move`)
 causes window to become attached to the cursor, prompting for new location.

 The command
 (`← self Move 200 100`)
 moves the lower left corner of the window to (200 . 100).

`(←self MousePackage)` [Method of Window]

Purpose/Behavior: Returns the package used during mouse interactions with the window *self*. (LOOPS now uses the INTERLISP package exclusively.) The **MousePackage** method protects LOOPS windows from the new packages. To remove this protection, specialize this method to return **PACKAGE**.

Arguments: *self* An instance of a window.

Returns: The INTERLISP package.

`(←self MouseReadtable)` [Method of Window]

Purpose/Behavior: Returns the readtable used during mouse interactions with the window *self*. Medley now has many different readtables; some readtables do not work well with LOOPS. (The Common Lisp readtables are not case-sensitive.) This method protects LOOPS windows from the new readtables. To remove this protection, specialize this method to return **READTABLE**.

Arguments: *self* An instance of a window.

Returns: The INTERLISP readtable.

`(← self Open)` [Method of Window]

Purpose/Behavior: Opens the specified window instance.

Arguments: *self* Pointer to a window instance.

Returns: NIL

Categories: Window

`(← self Paint)` [Method of Window]

Purpose/Behavior: Calls **PAINTW** on the specified window. You are prompted for instructions in the prompt window.

Arguments: *self* Pointer to a window instance.

Returns: NIL

Categories: Window

(← *self* **ScrollWindow** *dspX dspY windowX windowY*)

[Method of Window]

Purpose/Behavior: Scrolls the window to move the point *dspX*, *dspY* to *windowX*, *windowY*. If *windowX* and *windowY* are NIL, the default is to scroll so that the point *dspX*, *dspY* appears in the lower left corner of the window. Any of the arguments can be **FIXP** or **FLOATP**. If the value is **FIXP**, then it is treated as an absolute coordinate. If the value is **FLOATP**, then it is treated as a relative position.

Arguments:

<i>self</i>	Pointer to a window instance.
<i>dspX</i>	The x point in the given window to move; x is in window coordinates if FIXP . If FLOATP , the value to move is based upon the width of the EXTENT property of the window; see the <i>Interlisp-D Reference Manual</i> .
<i>dspY</i>	The y point in the given window to move; y is in window coordinates if FIXP . If FLOATP , the value to move is based upon the height of the EXTENT property of the window; see the <i>Interlisp-D Reference Manual</i> .
<i>windowX</i>	The x point to scroll to in window coordinates if FIXP . If FLOATP , the value to move is based upon the width of the window.
<i>windowY</i>	The y point to scroll to in window coordinates if FIXP . If FLOATP , the value to move is based upon the height of the window.

Returns: The lower left corner of the new **DSPCLIPPINGREGION**; see the *Interlisp-D Reference Manual*.

Categories: Window

(← *self* **SetProp** *prop value*)

[Method of Window]

Purpose/Behavior: Sets the Interlisp window property of the specified LOOPS window, passing its *prop* and *value* arguments through Interlisp function **WINDOWPROP**.

Arguments:

<i>self</i>	Pointer to a window instance.
<i>prop</i>	Property to set.
<i>value</i>	New value for property.

Returns: Previous value of *prop* if it existed; else NIL.

Categories: Window

(← *self* **Shape** *newRegion noUpdateFlg*)

[Method of Window]

Purpose/Behavior: Reshapes the specified window. If *newRegion* is not specified, you are prompted to reshape the window with the cursor.

Arguments:

<i>self</i>	Pointer to a window instance.
<i>newRegion</i>	A list specifying the new outer dimensions; the format for the list is (left bottom height width).
<i>noUpdateFlg</i>	If NIL, reshapes the window.

Returns: A list specifying the new region.

Categories: Window

Specializations: NonRectangularWindow

Example: The command

```
(← ($ window1) Shape ' (100 200 300 400))
```

returns

```
(100 200 300 400)
```

(← *self* **Shape?**)

[Method of Window]

Purpose/Behavior: Returns the current region for the window.

Arguments: *self* Pointer to a window instance.

Returns: A list specifying outer dimensions of the window.

Categories: Window

Example: The command

```
(← ($ window1) Shape?)
```

returns

```
(100 200 300 400)
```

(← *self* **Shrink** *toWhat* *iconPos* *expandFn*)

[Method of Window]

Purpose/Behavior: Shrinks the window to a given icon.

Arguments: *self* Pointer to a window instance.

toWhat The icon to shrink to; if NIL, an icon is created.

iconPos Position of icon on screen.

expandFn Function to be called on expansion.

Returns: The icon.

Categories: Window

Specializations: LatticeBrowser

(← *self* **Snap**)

[Method of Window]

Purpose/Behavior: Calls **SnapW** to take a snapshot of the window.

Arguments: *self* Pointer to a window instance.

Returns: The window.

Categories: Window

(← *self* **ToTop**)

[Method of Window]

Purpose/Behavior: Opens the window and brings it to the top of the screen.

Arguments: *self* Pointer to a window instance.

Returns: The window.

Categories: Window

(← *self* **Update**)

[Method of Window]

Purpose/Behavior: Makes the window consistent with the instance variables.

Arguments: *self* Pointer to a window instance.

Returns: NIL

Categories: Window

Specializations: NonRectangularWindow

(**WindowAfterMoveFn** *window*)

[Function]

Purpose/Behavior: This function is installed as the **AFTERMOVEFN** property of the Lisp window pointed to by a window object. This function extracts the window object from the property **LoopsWindow** and sends it the message **AfterMove**.

This **AFTERMOVEFN** is installed automatically by the system.

Arguments: *window* The window just moved.

Returns: Used for side effect only.

(**WindowShapeFn** *window oldBitmapImage oldRegion*)

[Function]

Purpose/Behavior: This function is installed as the **RESHAPEFN** property of the Lisp window pointed to by a window. This function extracts the window object from the property **LoopsWindow** and sends it the message **AfterReshape** with the arguments *oldBitmapImage oldRegion*.

This **RESHAPEFN** is installed automatically by the system.

Arguments: *window* The window just reshaped.

oldBitmapImage

See the *Lisp Release Notes* and the *Interlisp-D Reference Manual* for a discussion of window **ReShapeFns**.

oldRegion

See the *Lisp Release Notes* and the *Interlisp-D Reference Manual* for a discussion of window **ReShapeFns**.

Returns: Used for side effect only.

19.3 PROMPT WINDOWS

19.3 PROMPT WINDOWS

19.3 Prompt Windows

Prompt windows are windows attached to other windows and are used for displaying messages and for getting input. In LOOPS, these operate similarly to prompt windows in Lisp. Prompt windows are not instances of the class Window; they are only instances of the Interlisp data type **Window**.

The following table lists the methods and functions described in this section.

Name	Type	Description
Clear PromptWindow	Method	Clears the prompt window.
ClosePromptWindow	Method	Closes the prompt window.
GetPromptWindow	Method	Associates a prompt window with a LOOPS window.
PromptEval	Function	Prompts for, reads, and evaluates an expression.
PromptForList	Method	Prompts for a list of items.
PromptForString	Method	Prompts for a string.
PromptForWord	Method	Prompts for a word.
PromptPrint	Method	Prints a message in the prompt window.
PromptRead	Function	Prompts for and reads data.
NiceMenu	Function	Creates a menu.
SelectFile	Lambda NoSpread	Prompts for a file name.

Note: The methods **PromptForList**, **PromptForString**, and **PromptForWord**, as well as the functions **PromptRead** and **PromptEval** when called with a prompt window for a LOOPS window, all disable normal mouse button events in the prompting browser and will not allow it to close until the prompt is completed.

(← *self* **ClearPromptWindow**) [Method of Window]

Purpose/Behavior: Clears the prompt window associated with the window *self*.

Arguments: *self* Evaluates to a window instance.

Returns: NIL

Categories: Window

(← *self* **ClosePromptWindow**) [Method of Window]

Purpose/Behavior: Closes the prompt window associated with the window *self*.

Arguments: *self* Evaluates to a window instance.

Returns: The symbol **CLOSED** if a prompt window existed; else NIL.

Categories: Window

(← *self* **GetPromptWindow** *lines fontDef*) [Method of Window]

Purpose/Behavior: Gets a prompt window for window *self*. If one exists, it is returned; else a prompt window is created.

Arguments: *self* Pointer to a window instance.

lines Number of lines in window; default is 2.

fontDef Font used in the window; if NIL, this defaults to DEFAULTFONT.

Returns: Pointer to a prompt window.

Categories: Window

(PromptEval *promptString window sameLine?*)

[Function]

Purpose: Prompts for, reads, and evaluates an expression.

Behavior: Temporarily moves the TTYDISPLAYSTREAM to *window*, if *window* is non-NIL, else to the system prompt window.

The *promptString* is printed followed by a carriage return and the string "The expression read will be EVALuated."

The prompt "> " is printed on the same line as the above if *sameLine?* is non-NIL, else it is printed on a new line. Data entered by the user is evaluated and returned. **LISPX** and **LISPXREAD** are used so that the entered data is placed on the **LISPX** history list. (See the *Lisp Release Notes* and the *Interlisp-D Reference Manual*).

Note: When called with a prompt window for a LOOPS window, **PromptEval** disables normal mouse button events in the prompting browser and will not allow it to close until the prompt is completed.

Arguments: *promptString*

A string to be printed.

window

A window where the prompting and reading should occur. Defaults to the system prompt window.

sameLine?

If non-NIL, the data is read from the same line as the string "The expression read will be EVALuated."

Returns: The data entered by the user after it has been evaluated.

Example: The command

```
26←(← ($ Window) New (PromptEval "Specify new window for object name."))
```

causes the following to appear in the Prompt Window:

```
Specify new window for object name.
The expression read will be EVALuated.
```

```
>
```

Entering

```
`NewWindow
```

after the > causes the following return in the Executive Window.

```
#, ($& Window (NEW0.1Y%:.;h.eN6 . 501))
```

(← self PromptForList *promptStr initialString*)

[Method of Window]

Purpose/Behavior: Prompts you in prompt window for a list of symbols. If prompt window does not exist, one is created. Input is terminated by a carriage return.

TTYIN is used for editing the user's input; see the *Interlisp-D Reference Manual*.

Note: **PromptForList** disables normal mouse button events in the prompting browser and will not allow it to close until the prompt is completed.

Arguments: *self*

Pointer to a window instance.

promptStr Displayed in prompt window.

initialString Can be used as the default or the first item of the list.

Returns: The list of words entered in prompt window.

Categories: Window

Example: If (\$ Window1) is a window, then the command

```
27←(← ($ Window1) PromptForList "ENTER THE CODES ")
```

causes the prompt ENTER THE CODES to be displayed in an attached prompt window. The system waits for user input.

(← *self* **PromptForString** *promptStr initialStr*)

[Method of Window]

Purpose/Behavior: Prompts you in prompt window for a string. If a prompt window does not exist, one is created. Input is terminated by a carriage return.

TTYIN is used for editing the user's input; see the *Interlisp-D Reference Manual*.

Note: **PromptForString** disables normal mouse button events in the prompting browser and will not allow it to close until the prompt is completed.

Arguments: *self* Pointer to a window instance.

promptStr Displayed in prompt window.

initialStr Can be used as the default or the prefix to the string.

Returns: The string entered in prompt window.

Categories: Window

Example: If (\$ Window1) is a window, then the command

```
28←(← ($ Window1) PromptForString "ENTER THE CODES ")
```

causes the prompt ENTER THE CODES to be displayed in an attached prompt window. The system waits for user input.

(← *self* **PromptForWord** *promptStr initialWord*)

[Method of Window]

Purpose/Behavior: Returns (CAR (← *self* **PromptForList** *promptStr initialWord*))

Arguments: *self* Evaluates to a window instance.

promptStr Displayed in prompt window.

initialWord Can be used as the default.

Returns: See Behavior.

Categories: Window

Example: If (\$ Window1) is a window, then the command

```
29←(← ($ Window1) PromptForWord "NEW WORD " )
```

prompts you with NEW WORD in an attached prompt window.

(← *self* **PromptPrint** *msg*) [Method of Window]

Purpose/Behavior: Displays a message in the prompt window associated with the specified window instance. Creates the prompt window if it does not exist.

Arguments: *self* Evaluates to a window instance.
msg Message displayed.

Returns: The message printed.

Categories: Window

(**PromptRead** *promptString window sameLine?*) [Function]

Purpose: Prompts for and reads data.

Behavior: Temporarily moves the **TTYDISPLAYSTREAM** to *window*, if *window* is non-NIL, else to the system prompt window.

The *promptString* is printed. The prompt "> " is printed on the same line as the above if *sameLine?* is non-NIL, else it is printed on a new line. Data that you entered is read and returned.

This contrasts with **PromptEval** in that the entered data is not placed on the **LISPX** history list (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*).

Note: When called with a prompt window for a LOOPS window **PromptRead** disables normal mouse button events in the prompting browser and will not allow it to close until the prompt is completed.

Arguments: *promptString* A string to be printed.
window A window where the prompting and reading should occur. Defaults to the system prompt window.
sameLine? If non-NIL, the data is read from the same line as the *promptString*.

Returns: The data entered by the user.

(**NiceMenu** *items title*) [Function]

Purpose: Provides an interface to create a menu and displays the menu.

Behavior: Varies according to the arguments.

- If *items* is NIL, prints "No items for *title*" in the system prompt window and returns NIL.
- If *items* is non-NIL, this builds a menu with the **TITLE** *title*, with the **ITEMS** *items*, and with **CHANGEOFFSETFLG** set to T (see the *Interlisp-D Reference Manual*). If the length of *items* is more than 35, the menu has multiple columns.

Arguments: *items* A form that can be placed in the **ITEMS** field of a menu.
title A value that will be placed in the **TITLE** field of a menu.

Returns: Value depends on the arguments; see Behavior.

(**SelectFile** *prompts*) [Lambda NoSpread Function]

Purpose:	Prompts you for a file name.
Behavior:	Takes on unlimited number of arguments and will PROMPTPRINT all the arguments. Builds a menu with the items *newFile* and the files found on the variable FILELST. If you select one of the files, that is returned. If you select *newFile*, you are prompted to enter a file name. An empty filecoms is built for that file name, and the file name is returned. *newFile* has three subitems: • *newFile* See Behavior. • *loadFile* You are prompted to enter a file name. A search is performed to try to find and load the compiled file. If that is not found, an attempt is made to load the source file. Returns NIL if the file is not found. • *hiddenFile* A menu is displayed containing files that are on the variable LOADEDFILELST but not on FILELST.
Arguments:	<i>prompts</i> A number of expressions to be printed in the system prompt window.
Returns:	Value depends on the arguments; see Behavior.

19.4 MOUSE AND MENU FUNCTIONALITY

19.4 MOUSE AND MENU FUNCTIONALITY

19.4 Mouse and Menu Functionality

When a LOOPS window is instantiated, its instance variable **window** points to an instance of a Lisp window. This window has several properties set, among which are the following that are described in this section:

- The property **RIGHTBUTTONFN** that is set to **WindowRightButtonFn**.
- The property **BUTTONEVENTFN** that is set to **WindowButtonEventFn**.

This section will also explain the functionality of the above and how menus associated with LOOPS windows operate. For more information on Medley windows, see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*.

Name	Type	Description
ButtonEventFn	Method	Sends either the message TitleSelection , LeftSelection , or MiddleSelection .
ClearMenuCache	Method	Deletes menus saved on the menus field of a browser.
ItemMenu	Method	Creates a simple one-level menu.
LeftSelection	Method	Triggers functionality when the cursor is in a window and the left button is pressed.

MiddleSelection	Method	Triggers functionality when the cursor is in a window and the middle button is pressed.
RightButtonFn	Method	Sends the message RightSelection .
RightSelection	Method	Triggers functionality when the cursor is in a window and the right button is pressed.
TitleSelection	Method	Triggers functionality when the cursor is in a window's title bar and the left or middle button is pressed.
WhenMenuItemHeld	Method	Displays in the prompt window what happens when option is selected.
WindowButtonEventFn	Function	Invokes the method ButtonEventFn .
WindowRightButtonFn	Function	Invokes the method RightButtonFn .

(← *self* **ButtonEventFn**)

[Method of Window]

Purpose/Behavior:	If the cursor is not inside of the window pointed to by <i>self</i> , this sends the message TitleSelection to <i>self</i> . If the left mouse button is pressed, this sends the message LeftSelection to <i>self</i> . If the left middle button is pressed, this sends the message MiddleSelection to <i>self</i> .	
Arguments:	<i>self</i>	An instance of a window.
Returns:	Used for side effect only.	
Categories:	Window	

(← *self* **ClearMenuCache**)

[Method of Window]

Purpose/Behavior:	Deletes menus saved in any properties of the instance variable menus of a window. Use this method if you ever change the class variables describing a menu, and you want the new menu to take effect.	
Arguments:	<i>self</i>	Pointer to a window instance.
Returns:	<i>self</i>	
Categories:	Window	

(← *self* **ItemMenu** *items* *title*)

[Method of Window]

Purpose/Behavior:	Creates a simple one-level menu guaranteed not to be more than 750 bits high. A large number of menu options will cause a multiple column menu to be formed.	
Arguments:	<i>self</i>	Pointer to a window instance.
	<i>items</i>	The value of this is passed to the ITEMS field when the menu is created.
	<i>title</i>	The title for the menu's window.
Returns:	A menu.	

Categories: Window

Example: The command

```
32← (← (←New ($ Window)) ItemMenu ' (a b c))
```

will create a menu with the three options.

(← self LeftSelection)**[Method of Window]**

Purpose/Behavior: Invokes a number of internal methods of **Window**. A menu will pop up. The options in the menu will be defined by the class variable **LeftButtonItems** (or **ShiftLeftButtonItems** if the **Meta** key is also pressed). If an option is selected from the menu, a message will be sent to *self* with a selector as specified by the chosen menu option.

Arguments: *self* Pointer to a window instance.

Returns: Used for side effect only.

Categories: Window

Specializations: LatticeBrowser

(← self MiddleSelection)**[Method of Window]**

Purpose/Behavior: Invokes a number of internal methods of **Window**. A menu will pop up. The options in the menu will be defined by the class variable **MiddleButtonItems** (or **ShiftMiddleButtonItems** if the **Meta** key is also pressed). If an option is selected from the menu, a message will be sent to *self* with a selector as specified by the chosen menu option.

Arguments: *self* Pointer to a window instance.

Categories: Window

Specializations: LatticeBrowser

(← self RightSelection)**[Method of Window]**

Purpose/Behavior: Invokes a number of internal methods of **Window**. A menu will pop up. The options in the menu will be defined by the class variable **RightButtonItems**. If an option is selected from the menu, a message will be sent to *self* with a selector as specified by the chosen menu option.

Arguments: *self* Pointer to a window instance.

Returns: The menu.

Categories: Window

(← self TitleSelection)**[Method of Window]**

Purpose/Behavior: Invokes a number of internal methods of **Window**. A menu will pop up. The options in the menu will be defined by the class variable **TitleItems**. If an option is selected from the menu, a message will be sent to *self* with a selector as specified by the chosen menu option .

Arguments: *self* Pointer to a window instance.

Returns: The choice if selected; else NIL.

Categories: Window

Specializations: LatticeBrowser

(← *self* **WhenMenuItemHeld** *item* - -)

[Method of Window]

Purpose/Behavior: Displays in the system prompt window what will happen when the menu item is chosen. The information displayed will either be the help string for the item or the documentation for the method pointed to by the item.

Arguments: *self* Pointer to a window instance.

item The menu item selector.

Returns: NIL

Categories: Window

WindowButtonEventFn

[Function]

Purpose/Behavior: Invokes the method **ButtonEventFn**.

WindowRightButtonFn

[Function]

Purpose/Behavior: Invokes the method **RightButtonFn**.

(**WindowButtonEventFn** *window*)

[Function]

Purpose/Behavior: This retrieves the value of the Lisp window property **LoopsWindow**. It sends the message **ButtonEventFn** to that window object. If the window object is an instance of **NonRectangularWindow** or one of its subclasses and if the cursor is not within the icon bitmap, nothing occurs.

This is invoked automatically by the system when the cursor is inside of a window object and the left or middle mouse button is pressed.

Arguments: *window* The window that contained the cursor when the mouse button was pressed.

Returns: Used for side effect only.

(**WindowRightButtonFn** *window*)

[Function]

Purpose/Behavior: This retrieves the value of the Lisp window property **LoopsWindow**. It sends the message **RightButtonFn** to that window object. If the window object is an instance of **NonRectangularWindow** or one of its subclasses and if the cursor is not within the icon bitmap, nothing occurs.

This is invoked automatically by the system when the cursor is inside of a window object and the left or middle mouse button is pressed.

Arguments: *window* The window that contained the cursor when the mouse button was pressed.

Returns: Used for side effect only.

19.4.1 Menu Item Structure

The default behavior for the methods **LeftSelection**, **MiddleSelection**, and **RightSelection** causes a menu to pop up. The options that will appear in a menu are defined in various class variables of the window object being selected. (**IconWindows** are an exception). When an option is selected from a menu, a message is sent to the window with no arguments.

The value of the various class variables can be an item list, such as (item1....itemn) where each item can be one of:

- *selector*

In this case, the *selector* appears in the menu, and it is the selector of the message sent to the window.

- (*prompt selector help-string*)

In this case, the *prompt* appears in the menu, and *selector* is the selector of the message sent to the window. *help-string* is printed when the cursor is over the item and the mouse is pressed.

- (*prompt subitemStructure help-string*) where *subitemStructure* = (*defaultSelector itemList*)

This form allows a menu to contain submenus. In this case, the *prompt* appears in the main menu, and *defaultSelector* is the selector of the message sent to the window if the main menu item is selected. *itemlist* defines the submenu behavior.

For example, in the class **Window**, the class variable **LeftButtonItem**s has the following value:

```
((Update (QUOTE Update) "Update window to agree with object IVs"))
```

The class variable **RightButtonItem**s has the value:

```
((Close (Close (Close Destroy))) Snap Paint Clear Bury Repaint  
(Hardcopy (Hardcopy (HardcopyToFile HardcopyToPrinter))) Move Shape  
Shrink)
```

19.4.2 Caching Menus

When a menu is created by pressing a mouse button on a LOOPS window, the menu is cached on a property of the instance variable **menus** if **menus** has the value T. The name of the property where it is stored has the same name as the class variable that describes the menu. The method **ClearMenuCache** will set these properties to NIL, causing the menus to be deleted from the window instance.

19.5 SUBCLASSES OF WINDOW

19.5 SUBCLASSES OF WINDOW

19.5 Subclasses of Window

This section describes the classes **NonRectangularWindow**, **IconWindow**, and **LoopsIcon** and functionality associated with them.

Name	Type	Description
NonRectangularWindow	Class	Provides the capability for windows to act as icons.
CreateWindow	Method	Creates a window that acts like an icon.
EditIcon	Method	Edits an icon bitmap.
EditMask	Method	Edits a mask bitmap.
Invert	Method	Inverts the image of an icon; that is, reverses its black-white pattern.
Shape	Method	Prevents the window from being shaped by calling LoopsHelp .
IconWindow	Class	Provides some menu options for icon windows.
LoopsIcon	Class	Provides an icon that is part of the LOOPS user interface.
PutSavedValue	Function	Stores a value. This is called from within browser and inspector menu events.
SavedValue	Function	Retrieves a saved value.

NonRectangularWindow [Class]

Description:	Provides the capability for windows to act as icons.	
MetaClass:	Class	
Supers:	Window	
Instance Variables:	icon	Allows a bitmap to be used as an icon to be stored in the instance. If the bitMap property is set to a symbol whose value is a bitmap, then that bitmap will be used. The default value is NIL.
	mask	Allows a bitmap to be used as an icon mask to be stored in the instance. If the bitMap property is set to a symbol whose value is a bitmap, then that bitmap will be used. The default value is NIL.

(← self CreateWindow) [Method of NonRectangularWindow]

Purpose:	Creates a window that acts like an icon.	
Behavior:	Determines if icon and mask or the property bitMap have values. If so, it uses those within a call to ICONW . If not, it sends the messages EditIcon and EditMask .	
	This method is invoked by the system if the instance variable window is not yet bound to a value and it is accessed.	
Arguments:	<i>self</i>	A window instance.
Returns:	Value returned from ICONW .	
Categories:	Window	
Specializes:	Window	

(← self EditIcon) [Method of NonRectangularWindow]

Purpose: Edits an icon bitmap.

Behavior: Calls **EDITBM** with the value of the instance variable **icon** and assigns **icon** to the returned value.

Arguments: *self* A window instance.

Returns: Value returned from **EDITBM**.

Categories: NonRectangularWindow

(← *self* **EditMask**) [Method of NonRectangularWindow]

Purpose: Edits a mask bitmap.

Behavior: Calls **EDITBM** with the value of the instance variable **mask** if it is non-NIL, or a copy of **icon**, and assigns **mask** to the returned value.

Arguments: *self* A window instance.

Returns: Value returned from **EDITBM**.

Categories: NonRectangularWindow

(← *self* **Invert**) [Method of NonRectangularWindow]

Purpose: Inverts the image of an icon; that is, reverses its black-white pattern.

Behavior: Modifies the bitmap of the **ICONIMAGE** window property of the Lisp window pointed to by *self*.

Arguments: *self* A window instance.

Returns: Used for side effect only.

Categories: Window

Specializes: Window

(← *self* **Shape**) [Method of NonRectangularWindow]

Purpose/Behavior: Prevents the window from being shaped by calling **LoopsHelp**.

This method is provided to restrict the shaping of this class of window, not to provide additional functionality.

Arguments: *self* A window instance.

Returns: Used for side effect only.

Categories: Window

Specializes: Window

IconWindow [Class]

Description: Provides some menu options for icon windows.

The menu behavior of this class is different from the class **Window** in that the item lists are stored on instance variables and not class variables.

MetaClass: Class

Supers: NonRectangularWindow

Instance Variables: **RightButtonItem**s

A list that defines the menu that will appear when the right mouse button is pressed when the cursor is in the window. The default value is (Move).

MiddleButtonItems

A list that defines the menu that will appear when the middle mouse button is pressed when the cursor is in the window. The default value is NIL.

LeftButtonItems

A list that defines the menu that will appear when the left mouse button is pressed when the cursor is in the window. The default value is (Move).

ShiftMiddleButtonItems

A list that defines the menu that will appear when the middle mouse button is pressed when the cursor is in the window and the **Meta** key is pressed. The default value is NIL.

ShiftLeftButtonItems

A list that defines the menu that will appear when the left mouse button is pressed when the cursor is in the window and the **Meta** key is pressed. The default value is (Move).

LoopsIcon

[Class]

Description: Implements the LOOPS icon which is part of the LOOPS user interface to LatticeBrowsers (see Chapter 10, Browsers).

MetaClass: Class

Supers: NonRectangularWindow

Class Variables: **RightButtonItem**s

A list that defines the menu that will appear when the right mouse button is pressed when the cursor is in the window. The default value is (Close Move).

MiddleButtonItems

A list that defines the menu that will appear when the middle mouse button is pressed when the cursor is in the window. The default value is ("Browse File" (...)).

LeftButtonItems

A list that defines the menu that will appear when the left mouse button is pressed when the cursor is in the window. The default value is ("Browse Class" (...)).

Instance Variables: **savedValue** Used by the functions **PutSavedValue** and **SavedValue**. The default value is NIL.

icon The property **bitMap** has the value **BlackLoopsIconBM**. The default value is NIL.

mask The property **bitMap** has the value **LoopsIconShadow**. The default value is NIL.

(PutSavedValue value)

[Function]

Purpose: Stores a value. This is called from within browser and inspector menu events.

Behavior: Sets the instance variable **savedValue** of the prototype instance of the class **LoopsIcon** to *value*. Also sets the top level binding of **IT** to *value*; see the *Interlisp-D Reference Manual* for information on **IT**.

Arguments: *value* Any arbitrary data.

Returns: *value*

(**SavedValue**)

[Function]

Purpose: Retrieves a saved value.

Behavior/Returns: Gets the value of the instance variable **savedValue** of the prototype instance of the class **LoopsIcon**.

19.6 LISP WINDOWS

19.6 LISP WINDOWS

19.6 Lisp Windows

The methods in this section define the interface between LOOPS windows and Lisp windows. These methods are used internally by the system, and will rarely be used or specialized by users.

Name	Type	Description
AttachLispWindow	Method	Gives a LOOPS window a Lisp window.
CreateWindow	Method	Creates a Lisp window.
DetachLispWindow	Method	Forgets about the current Lisp window.
GetWrappedValue	Method	Gets the value wrapped in the active value.
HasLispWindow	Method	Checks if a Lisp window has ever been created for the LOOPS window.
PutWrappedValue	Method	Replaced the value wrapped in the active value.

(← *self* **AttachLispWindow** *window*)

[Method of Window]

Purpose/Behavior: Used to associate a LOOPS window with a Medley window. This detaches any currently attached window before attaching a new one, and fills in the instance variables **left**, **bottom**, **width**, **height**, and **title** from the Lisp window.

Arguments: *self* An instance of a LOOPS window.
window Must be a window pointer.

Returns: Used for side effect only.

Categories: Window

(← *self* **CreateWindow**)

[Method of Window]

Purpose/Behavior: Creates a Lisp window for a LOOPS window but does not open it.

Arguments: *self* Pointer to a LOOPS window.

Returns: The window.
 Categories: Window
 Specializations: NonRectangularWindow

(← *self* **DetachLispWindow**) [Method of Window]

Purpose/Behavior: Removes the pointer from the LOOPS window to the Lisp window.
 Arguments: *self* Pointer to a LOOPS window.
 Returns: Used for side effect only.
 Categories: Window

(← *self* **GetWrappedValue** *containingObj* *varName* *propName* *type*) [Method of LispWindowAV]

Purpose/Behavior: Used by the system to fetch the Medley window from a LOOPS window. If the local state of this active value is not a window, it is made a window.
 Arguments: *self* An instance of **LispWindowAV**.
 containingObj A LOOPS window.
 varName Variable associated with the wrapped value.
 propName Used internally.
 type Used internally.
 Returns: A Lisp window.
 Categories: LispWindowAv
 Specializes: LocalStateActiveValue

(← *self* **HasLispWindow**) [Method of Window]

Purpose/Behavior: Checks if Lisp window has ever been created for this LOOPS window.
 Arguments: *self* A LOOPS window.
 Returns: The window pointer, if one exists; else NIL.
 Categories: Window

(← *self* **PutWrappedValue** *containingObj* *varName* *newvalue* *propName*) [Method of LispWindowAV]

Purpose/Behavior: Places the window *newvalue* as local state of the active value.
 Arguments: *self* An instance of **LispWindowAV**.
 containingObj A LOOPS window.
 varName Variable associated with the wrapped value.
 propName Used internally.
 type Used internally.

Returns: The window pointer, if one exists.

Categories: LispWindowAV

Specializes: LocalStateActiveValue

[This page intentionally left blank]

20. SYSTEM VARIABLES AND FUNCTIONS

This section describes the following system variables and functions. These variables are set within the file **LOADLOOPS** or when the function **LOADLOOPS** is executed.

Name	Type	Description
LoopsVersion	Variable	Identifies a release of LOOPS.
LoopsDate	Variable	Identifies the date when the function LOADLOOPS is executed.
FEATURES	Variable	Has the symbol LOOPS added to it when the function LOADLOOPS is evaluated.
LoadLoopsForms	Variable	Contains a list of forms that are evaluated when LOOPS is loaded.
LispUserFilesForLoops	Variable	Contains a list of files required by LOOPS .
OptionalLispuserFiles	Variable	Contains a list of files that is loaded when LOOPS is loaded.
LOOPSDIRECTORY	Variable	Contains the connected directory when LOADLOOPS is loaded.
LOOPSLIBRARYDIRECTORY	Variable	Contains the directory where the LOOPS library files reside.
LOOPUSERSDIRECTORY	Variable	Contains the directory where the LOOPS Users' Modules files reside.
LOOPUSERSRULESDIRECTORY	Variable	Contains the directory where the LOOPS Rules User Module file resides.
LoopsPatchFiles	Variable	Contains a list of LOOPS files that are loaded when LOOPS is loaded.
LOOPFILES	Variable	Contains a list of LOOPS files that are loaded by the function LOADLOOPS .
ClearAllCaches	Function	Clears all caches used by LOOPS .
ClearAllCaches	Variable	Contains a list of forms that are evaluated within a call to the function ClearAllCaches .

LoopsVersion	[Variable]

Set to uniquely identify a release of **LOOPS**.

LoopsDate	[Variable]

Set to the value of (DATE) when the function **LOADLOOPS** is evaluated.

FEATURES [Variable]

Has the symbol **LOOPS** added to it when the function **LOADLOOPS** is evaluated. See the *Common Lisp: the Language* for more information on ***FEATURES***.

LoadLoopsForms [Variable]

Contains a list of forms that are evaluated when **LOOPS** is loaded. Initialized to **NIL** using the File Manager command **INITVARS** (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*).

LispUserFilesForLoops [Variable]

Contains a list of files required by **LOOPS**. Initialized to (**GRAPHER**).

OptionalLispuserFiles [Variable]

Contains a list of files that is loaded when **LOOPS** is loaded. Initialized to **NIL** using the File Manager command **INITVARS**.

LOOPSDIRECTORY [Variable]

Initialized to the directory from which the file **LOADLOOPS** is loaded using the File Manager command **INITVARS**.

LOOPSLIBRARYDIRECTORY [Variable]

Contains the directory where the **LOOPS** library files reside.

LOOPUSERSDIRECTORY [Variable]

Contains the directory where the **LOOPS** Users' Modules files reside.

LOOPUSERSRULESDIRECTORY [Variable]

Contains the directory where the **LOOPS** Rules User Module file resides.

LoopsPatchFiles [Variable]

Contains a list that can be passed to **FILESLOAD** (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*) and is used during the loading of **LOOPS**. Initialized to ((**LOAD FROM VALUEOF LOOPSDIRECTORY**) **MASTERSCOPE MSPARSE**).

LOOPFILES [Variable]

Contains the list of **LOOPS** files loaded by **LOADLOOPS** when building a **LOOPS** sysout.

(ClearAllCaches) [Function]

Purpose: Clears all caches used by **LOOPS**

Behavior: In addition to clearing some caches used to speed up method and instance variable lookup, this clears the hash array **CLISPARRAY** (see the *Lisp Release Notes* and the *Interlisp-D Reference Manual*) and sends the **ClearMenuCache** message to any open LOOPS windows or their icons.

Returns: NIL

ClearAllCaches

[Variable]

Contains a list of forms, each of which is evaluated within a call to the function **ClearAllCaches**. Initially set to NIL.

[This page intentionally left blank]

XEROX LOOPS REFERENCE MANUAL

XEROX

XEROX LOOPS REFERENCE MANUAL

Lyric/Medley Release

July 1988

Copyright © 1988 by Xerox Corporation.

Xerox LOOPS is a trademark.

All rights reserved.

TABLE OF CONTENTS

PREFACE

xv

1. INTRODUCTION

1.1 Introduction to Objects	1-1
1.1.1 Object	1-2
1.1.2 Message	1-3
1.1.3 Method	1-3
1.1.4 Selector	1-3
1.1.5 Class	1-3
1.1.6 Instance	1-4
1.2 Storage of Data in Objects	1-4
1.2.1 Class Variables and Instance Variables	1-4
1.2.2 Properties	1-5
1.3 Metaclasses	1-6
1.4 Introduction to Inheritance	1-6
1.4.1 Single Superclasses	1-7
1.4.2 Multiple Superclasses	1-8
1.5 Introduction to Access-Oriented Programming: Using Active Values	1-9
1.6 Introduction to the Xerox LOOPS User Interface	1-10
1.6.1 SEdit	1-10
1.6.2 Inspector	1-10
1.6.3 Masterscope	1-10
1.6.4 File Manager	1-11
1.6.5 Grapher Module	1-11

2. INSTANCES

2.1 Instance Naming Conventions	2-1
2.2 Creating Instances	2-4
2.3 Data Storage in Instances at Creation Time	2-8
2.4 Changing the Number of Instance Variables in an Instance	2-10
2.5 Moving Variables	2-13
2.6 Destroying Instances	2-15
2.7 Methods Concerning the Class of an Object	2-16
2.8 Copying Instances	2-19
2.9 Querying Structure of Instances	2-21
2.10 Other Instance Items	2-24

3. CLASSES

3.1 Creating Classes	3-1
3.1.1 Function Calling and Message Sending	3-2
3.1.2 Dynamic Mixins	3-4
3.2 Destroying Classes	3-5
3.3 Inheritance	3-7
3.4 Editing Classes	3-10
3.5 Modifying Classes	3-11
3.6 Methods for Manipulating Class Names	3-16
3.7 Querying the Structure of a Class	3-17
3.8 Copying Classes and Their Contents	3-23
3.9 Enumerating Instances of Classes	3-24
3.10 Dealing with Inheritance	3-27

4. METACLASSES

4.1 Specific Metaclasses	4-1

TABLE OF CONTENTS

4.1.1 The Metaclass Class	4-1
4.1.2 The Metaclass Metaclass	4-2
4.1.3 The Metaclass AbstractClass	4-2
4.1.4 The Metaclass DestroyedClass	4-2
4.2 Pseudoclasses	4-2
4.3 Defining New Metaclasses	4-5
4.4 Tofu	4-6

5. ACCESSING DATA

5.1 Generalized Get and Put Functions	5-1
5.2 Accessing Data in Instances	5-4
5.2.1 Compact Accessing Forms	5-10
5.2.2 Support for Changetran	5-13
5.3 Accessing Data in Classes	5-13
5.3.1 Metaclasses and Property Access	5-13
5.3.2 Class Variable Access	5-16
5.3.3 Instance Variable Access	5-19

6. METHODS

6.1 Categories	6-1
6.2 Structure of Method Functions	6-3
6.3 Creating, Editing, and Destroying Methods	6-4
6.4 Escaping from Message Syntax	6-6
6.5 Movement between Classes	6-8
6.5.1 Movement of Methods	6-8
6.5.2 Stack Method Macros	6-10

7. MESSAGE SENDING FORMS

8. ACTIVE VALUES

8.1 Using Active Values	8-2
8.2 Specializations of the Class <code>ActiveValue</code>	8-2
8.2.1 <code>IndirectVariable</code>	8-3
8.2.2 <code>LocalStateActiveValue</code>	8-6
8.2.2.1 <code>ExplicitFnActiveValue</code>	8-8
8.2.2.2 <code>NoUpdatePermittedAV</code>	8-9
8.2.2.3 <code>LispWindowAV</code>	8-10
8.2.2.4 Breaking and Tracing Active Values	8-10
8.2.2.5 <code>AppendSuperValue</code>	8-11
8.2.2.6 <code>FirstFetchAV</code>	8-12
8.2.3 <code>InheritingAV</code>	8-14
8.2.4 <code>ReplaceMeAV</code>	8-15
8.2.5 <code>NotSetValue</code>	8-15
8.2.5.1 <code>NestedNotSetValue</code>	8-16
8.2.6 User Specializations of Active Values	8-16
8.3 Active Value Methods	8-16
8.3.1 Adding and Deleting Active Values	8-17
8.3.2 Fetching and Replacing Wrapped Values	8-19
8.3.3 Get and Put Functions Bypassing the <code>ActiveValue</code> Mechanism	8-22
8.3.4 Shared Active Values in Variable Inheritance	8-22
8.3.5 Creating Your Own Active Values	8-23
8.4 Annotated Values	8-24
8.4.1 Explicit Control over Annotated Values	8-25
8.4.2 Saving and Restoring Annotated Values	8-26
8.5 Active Values in Class Structures	8-27

TABLE OF CONTENTS

9. DATA TYPE PREDICATES AND ITERATIVE OPERATORS

9.1 Data Type Predicates	9-1

9.2 Iterative Operators	9-3

10. BROWSERS

10.1 Types of Built-in Browsers	10-1

10.1.1 Lattice Browsers	10-2

10.1.2 Class Browsers	10-2

10.1.3 File Browsers	10-2

10.1.4 Supers Browsers	10-2

10.1.5 Metaclass Browsers	10-2

10.1.6 Instance Browsers	10-3

10.2 Opening Browsers	10-3

10.2.1 Using Menu Options to Open Browsers	10-3

10.2.1.1 Overview of Background Menu and LOOPS Icon	10-3

10.2.1.2 Command Summary	10-4

10.2.2 Using Commands to Open Browsers	10-5

10.3 Using Class Browsers, Meta Browsers, and Supers Browsers	10-7

10.3.1 Selecting Options in the Title Bar Menu	10-8

10.3.1.1 Recompute and its Suboptions	10-8

10.3.1.2 AddRoot and its Suboptions	10-10

10.3.1.3 Add Category Menu	10-10

10.3.2 Selecting Options in the Left Menu	10-11

10.3.2.1 PrintSummary and its Suboptions	10-12

10.3.2.2 Doc (ClassDoc) and its Suboptions	10-13

10.3.2.3 WhereIs and its Suboptions	10-14

10.3.2.4 DeleteFromBrowser and its Suboptions	10-16

10.3.2.5 SubBrowser	10-16

10.3.2.6 TypeInName	10-16

10.3.2.7 Extending Functionality with the Left Mouse Button	10-16

<u>10.3.3 Selecting Options in the Middle Menu</u>	<u>10-17</u>
<u>10.3.3.1 Box/UnBoxNode</u>	<u>10-17</u>
<u>10.3.3.2 Methods (EditMethod) and its Suboptions</u>	<u>10-18</u>
<u>10.3.3.3 Add (AddMethod) and its Suboptions</u>	<u>10-20</u>
<u>10.3.3.4 Delete (DeleteMethod) and its Suboptions</u>	<u>10-21</u>
<u>10.3.3.5 Move (MoveMethodTo) and its Suboptions</u>	<u>10-22</u>
<u>10.3.3.6 Copy (CopyMethodTo) and its Suboptions</u>	<u>10-23</u>
<u>10.3.3.7 Rename (RenameMethod) and its Suboptions</u>	<u>10-23</u>
<u>10.3.3.8 Edit (EditClass) and its Suboptions</u>	<u>10-24</u>
<u>10.4 Using File Browsers</u>	<u>10-24</u>
<u>10.4.1 Selecting Options in the Title Bar Menu</u>	<u>10-25</u>
<u>10.4.1.1 Recompute and its Suboptions</u>	<u>10-25</u>
<u>10.4.1.2 AddRoot and its Suboptions</u>	<u>10-25</u>
<u>10.4.1.3 Add Category Menu</u>	<u>10-25</u>
<u>10.4.1.4 Change display mode and its Suboptions</u>	<u>10-25</u>
<u>10.4.1.5 Uses IV? and its Suboptions</u>	<u>10-26</u>
<u>10.4.1.6 Edit FileComs and its Suboptions</u>	<u>10-28</u>
<u>10.4.1.7 CLEANUP file and its Suboptions</u>	<u>10-30</u>
<u>10.4.2 Selecting Options in the Left Menu</u>	<u>10-30</u>
<u>10.4.2.1 PrintSummary and its Suboptions</u>	<u>10-30</u>
<u>10.4.2.2 Doc (ClassDoc) and its Suboptions</u>	<u>10-30</u>
<u>10.4.2.3 WhereIs (WhereIsMethod) and its Suboptions</u>	<u>10-30</u>
<u>10.4.2.4 DeleteFromBrowser and its Suboptions</u>	<u>10-31</u>
<u>10.4.2.5 SubBrowser</u>	<u>10-31</u>
<u>10.4.2.6 TypeInName</u>	<u>10-31</u>
<u>10.4.2.7 AddSubs and its Suboptions</u>	<u>10-31</u>
<u>10.4.3 Selecting Options in the Middle Menu</u>	<u>10-31</u>
<u>10.4.3.1 BoxNode</u>	<u>10-32</u>
<u>10.4.3.2 Methods (EditMethod) and its Suboptions</u>	<u>10-32</u>
<u>10.4.3.3 Add (AddMethod) and its Suboptions</u>	<u>10-32</u>
<u>10.4.3.4 Delete (DeleteMethod) and its Suboptions</u>	<u>10-32</u>
<u>10.4.3.5 Move (MoveMethodTo) and its Suboptions</u>	<u>10-32</u>
<u>10.4.3.6 Copy (CopyMethodTo) and its Suboptions</u>	<u>10-32</u>

TABLE OF CONTENTS

10.4.3.7	Rename (RenameMethod) and its Suboptions	10-32
10.4.3.8	Edit (EditClass) and its Suboptions	10-32
10.4.3.9	UsesIV and its Suboptions	10-33
10.5	Programmer's Interface to Lattice Browsers	10-33
10.5.1	Instance Variables for the Class LatticeBrowser	10-33
10.5.2	Class Variables for the Class LatticeBrowser	10-34
10.5.3	Methods for the Class LatticeBrowser	10-35
10.6	Instance Browsers	10-50
10.6.1	Instance Variables for the Class InstanceBrowser	10-50
10.6.2	Methods for the Class InstanceBrowser	10-50
10.6.3	Selecting Options in the Title Bar Menu	10-51
10.6.4	Selecting Options in the Left Menu	10-51
10.6.5	Selecting Options in the Middle Menu	10-52
10.7	Automatic Updates of Class Browsers	10-52

11. ERRORS AND BREAKS

11.1	Error Handling Functions and Methods	11-1
11.2	Error Messages	11-5
11.1.1	Classes and Instances	11-6
11.1.2	Methods and Messages	11-7
11.1.3	Naming Objects	11-8
11.1.4	Annotated and Active Values	11-9
11.1.5	Miscellaneous	11-9

12. BREAKING AND TRACING

12.1	Breaking and Tracing Methods	12-1
12.2	Breaking and Tracing Data	12-3

13. EDITING

13.1 Editing Classes	13-1

13.2 Editing Instances	13-5

14. FILE MANAGER

14.1 Manipulating Files	14-1

14.2 Loading Files	14-2

14.3 Xerox LOOPS File Manager Commands	14-3

14.4 Saving Xerox LOOPS Objects on Files	14-6

14.5 Storing Files	14-10

14.6 Compiling Files	14-12

15. PERFORMANCE ISSUES

15.1 Garbage Collection	15-1

15.2 Instance Variable Access	15-1

15.3 Method Lookup	15-3

15.4 Cache Clearing	15-3

16. PROCESSES

17. READING AND PRINTING

17.1 Reading Objects	17-1

17.2 Print Flags	17-2

17.3 Printing Classes	17-4

17.4 Printing Objects	17-8

17.5 Printing Active Values	17-11

TABLE OF CONTENTS

<u>17.6 Printing Methods</u>	<u>17-12</u>
<u>17.7 Unique Identifiers (UIDs)</u>	<u>17-14</u>

18. USER INPUT/OUTPUT MODULES

<u>18.1 Inspector</u>	<u>18-1</u>
<u>18.1.1 Overview of the User Interface</u>	<u>18-1</u>
<u>18.1.2 Using Instance Inspectors</u>	<u>18-2</u>
<u>18.1.2.1 Titles of Instance Inspector Windows</u>	<u>18-2</u>
<u>18.1.2.2 Menu for the Title Bar</u>	<u>18-3</u>
<u>18.1.2.3 Menu for the Left Column</u>	<u>18-4</u>
<u>18.1.2.4 Menu for the Right Column</u>	<u>18-6</u>
<u>18.1.3 Using Class Inspectors</u>	<u>18-7</u>
<u>18.1.3.1 Titles of Class Inspector Windows</u>	<u>18-7</u>
<u>18.1.3.2 Menu for the Title Bar</u>	<u>18-8</u>
<u>18.1.3.3 Menu for the Left Column</u>	<u>18-8</u>
<u>18.1.3.4 Menu for the Right Column</u>	<u>18-8</u>
<u>18.1.4 Using Class IVs Inspectors</u>	<u>18-9</u>
<u>18.1.4.1 Titles of Class IVs Inspector Windows</u>	<u>18-9</u>
<u>18.1.4.2 Menu for the Title Bar</u>	<u>18-9</u>
<u>18.1.4.3 Menu for the Left Column</u>	<u>18-10</u>
<u>18.1.4.4 Menu for the Right Column</u>	<u>18-10</u>
<u>18.1.5 Functional Interface for Instance Inspectors</u>	<u>18-11</u>
<u>18.1.6 Customizing the Inspector</u>	<u>18-15</u>
<u>18.2 Extensions to ?=</u>	<u>18-16</u>
<u>18.2.1 Message Sending</u>	<u>18-16</u>
<u>18.2.2 Record Creation</u>	<u>18-17</u>

19. WINDOWS

<u>19.1 The Class Window</u>	<u>19-1</u>

19.2 Basic Window Methods	19-2
19.3 Prompt Windows	19-9
19.4 Mouse and Menu Functionality	19-14
19.4.1 Menu Item Structure	19-17
19.4.2 Caching Menus	19-18
19.5 Subclasses of a Window	19-18
19.6 Lisp Windows	19-22

20. SYSTEM VARIABLES AND FUNCTIONS

INDEX	INDEX-1

GLOSSARY	GLOSSARY-1

LIST OF FIGURES

1-1. Xerox LOOPS Lattice	1-2
1-2. An Object	1-2
1-3. An Object Responding to a Message	1-3
1-4. A Message Containing a Selector	1-3
1-5. Class with Several Objects	1-4
1-6. Class Variables and Instance Variables	1-5
1-7. A Metaclass and its Instances	1-6
1-8. A Sample Inheritance Network	1-7
1-9. A Class with a Single Superclass	1-8
1-10. A Class with Multiple Superclasses	1-9
3-1. Simple Inheritance Lattice	3-8
3-2. Multiple Inheritance Lattice	3-9
3-3. Sample Display Editor Window	3-11
4-1. Class Browser Showing Metaclasses	4-1
4-2. Specializations of Tofu	4-6
8-1. The Class ActiveValue and its Specializations	8-3
10-1. Sample Lattice Browser	10-2
10-2. Sample Supers Browser	10-2
10-3. Sample Metaclass Browser	10-3
10-4. Xerox LOOPS Icon	10-4
10-5. Shading Available for a Node	10-47
18-1. Sample Inspector	18-1

[This page intentionally left blank]

Overview of the Manual

The *Xerox LOOPS Reference Manual* provides a detailed description of all the methods, functions, classes, and other items available in Xerox's Lisp Object-Oriented Programming System, Xerox LOOPS (TM). This manual describes the Lyric/Medley Release of Xerox LOOPS, which runs under the Lyric and Medley (with a small patch) releases of Xerox Lisp.

This manual is for people who are familiar with Xerox LOOPS programming principles, and is not intended to teach you Xerox LOOPS or how to use it. Please contact your LOOPS distributor for information about classes and training material.

Organization of the Manual and How to Use It

This manual is divided into chapters, with most chapters focusing on a particular aspect of Xerox LOOPS. The organization of this manual is similar to the *Interlisp-D Reference Manual*.

A Table of Contents is included at the beginning of the manual to help you find specific material. Chapters that have four levels of headings also have internal Tables of Contents. At the end of the manual, a Glossary is included to define terms within the context of Xerox LOOPS.

All readers should review Chapter 1, Introduction, before referring to specific material.

Conventions

This manual uses the following conventions:

- Case is significant in Xerox LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown. Typically, this means that method names are capitalized and variables are not.
- Arguments appear in italic type. Optional arguments are indicated by a dash (-).
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a message sending form appears as follows:

```
(_ self Selector Arg1 Arg2 -)
```

- Examples appear in the following typeface:

```
89_(_LOGIN)
```

- All examples are typed into an Interlisp Exec. This is the recommended Exec for all Xerox LOOPS expressions.

- Methods with an exclamation mark (!) suffix usually perform operations deeply into class structure instead of only on a given object.
- Methods with a question mark (?) suffix usually are predicates; that is, truth functions.
- Method names often appear in the form **ClassName.SelectorName**.
- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

This manual describes the Xerox LOOPS items (functions, methods, etc.) by using the following template:

Purpose:	Gives a short statement of what the item does.		
Behavior:	Provides the details of how the item operates.		
Arguments:	Describes each argument in the following format: <table> <tr> <td><i>argument</i></td><td>Description</td></tr> </table>	<i>argument</i>	Description
<i>argument</i>	Description		
Returns:	States what the item returns, and does not appear if the item does not return a value. The phrase "Used as a side effect only." means that the purpose of the item is to perform a computation or action that is independent of any returned value, not to return a particular value.		
Categories:	A way to group related methods. For example, all the methods related to Masterscope on the class FileBrowser have the category Masterscope, not FileBrowser . This item appears only for methods.		
Specializes:	The next higher class in the class hierarchy that contains a method with the same selector; only appear for methods. For example, the manual entry for RectangularWindow.Open would say that it specializes Window.Open , since Window is the first superclass of RectangularWindow that implements a method for Open .		
Specializations:	The next lower class(es) in the class hierarchy that contains method(s) with the same selector; only appears for methods. For example, the manual entry for Window.Open would say that it has a specialization of RectangularWindow.Open since RectangularWindow is a subclass of Window and has its own version of Open method.		
Example:	An example is often included to show how to use the item and what result it produces. Some examples may appear differently on your system, depending on the settings of various print flags. See Chapter 18, Reading and Printing, for details.		

References

The following books and manuals augment this manual.

Xerox LOOPS Release Notes

Xerox LOOPS Library Modules Manual

Xerox LOOPS Users' Modules Manual

Interlisp-D Reference Manual

Common Lisp: the Language by Guy Steele

Xerox Common Lisp Implementation Notes, Lyric and Medley Releases

Xerox Lisp Release Notes, Lyric and Medley Releases

Xerox Lisp Library Modules Manual, Lyric and Medley Releases

[This page intentionally left blank]

Writer's Notes -- Conventions

This file includes notes on conventions for *Xerox LOOPS Reference Manual*, Koto Release. This manual is packaged with the *Xerox LOOPS Release Notes* and *Xerox LOOPS Library Packages Manual* to form one binder, part number 3103310.

Writer: Rosie (Raven) Kontur

Printing Date: <DD> <MM> 1987

Directories and Files

The directory {ERIS}<Doc>LoopsKoto>Ref> contains the files for the manual. This directory has the following subdirectories:

- {ERIS}<Doc>LoopsKoto>Ref>X-Index> contains IMPTR files needed to produce the index as well as the index itself.
- {ERIS}<Doc>LoopsKoto>Ref>Z-ReleaseInfo> contains this file on writing conventions and a file on production details.

Filenames describe the contents of the file. For example, the filename

{ERIS}<Doc>LoopsKoto>Ref>Ch01-Intro.tedit

contains Chapter 1, Introduction.

Some chapters (for example, 5, 10, and 19) have internal tables of contents. Chapter 10, which is the largest chapter and has the most bitmaps, is divided among three files.

Assemble the files in the following order for the manual:

```
{ERIS}<Doc>LoopsKoto>Ref>A1-TitlePage.tedit
{ERIS}<Doc>LoopsKoto>Ref>A2-TOC.tedit
{ERIS}<Doc>LoopsKoto>Ref>A3-LOF.tedit
{ERIS}<Doc>LoopsKoto>Ref>A4-Preface.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch01-Intro.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch02-Instances.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch03-Classes.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch04-Metaclasses.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch05-TOC.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch05-ActiveValue.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch06-Methods.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch07-MsgForms.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch08-IterStmts.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch09-Misc.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch10-TOC.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch10a-Browsers1.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch10b-Browsers2.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch10c-Browsers3.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch11-Errors.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch12-Breaking.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch13-Editing.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch14-FilePkg.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch15-Masterscope.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch16-Performance.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch17-Processes.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch18-ReadPrint.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch19-TOC.tedit
{ERIS}<Doc>LoopsKoto>Ref>Ch19-UserIOPkgs.tedit
```

```
{ERIS}<Doc>LoopsKoto>Ref>Ch20-Windows.tedit  
{ERIS}<Doc>LoopsKoto>Ref>PndxA-Previous.tedit  
{ERIS}<Doc>LoopsKoto>Ref>ZZ-Glossary.tedit  
{ERIS}<Doc>LoopsKoto>Ref>X-Index>Index-Final.tedit
```

About the Index

Creating a properly formatted index takes a certain amount of work (about 2 hours).

- Type in your Executive (`TEDIT (MAKE.IM.INDEX T NIL '(filenames) NIL)`)
- Save the resulting file in {ERIS}<Doc>LoopsKoto>Ref>RefIndex-Raw.tedit.
- Now start dithering. The task is to change all the separators in the file. For example, the index program returns pages in the form 1.2; 10.5,38; 18.3-4. The Index format we use is in the form 1-2; 10-5; 10-38; 18-3. Unfortunately, you can't make all the necessary changes in a global substitute, since many of the separators are also in the text and the comma requires that you repeat the chapter number. Here goes:
 - Fix the commas. Do a Find on a comma (,). Everywhere a comma occurs in a page number, change it to a semicolon (;) and repeat the chapter number and a dot before the page number. For example, 10.5,38 becomes 10-5; 10-38.
 - Fix the dashes. Do a Find on a dash (-). Everywhere a dash occurs in a page number, delete it and the following page number.
 - Fix the period. Except for the form #., you can use a global substitute from a period to a dash (. to -). This takes about 20-30 minutes, as there are about 700 substitutions.

I really recommend saving the file at this point.

- Hardcopy the index. Make whatever page breaks and other changes you feel are necessary. Make a final hardcopy and save the file.

Conventions

This manual uses the following conventions:

- Case is significant in Xerox LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown. Typically, this means that method names are capitalized and variables are not.
- Arguments appear in italic type.
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a method appears as follows:

```
(_ self Selector Arg1 Arg2)
```

- Examples appear in the following typeface:

```
89_ ( _ LOGIN )
```

- Methods with an exclamation mark (!) suffix usually perform operations deeply into class structure instead of only on a given object.
- Methods with a question mark (?) suffix usually are predicates; that is, truth functions.
- Methods often appear in the form **ClassName.SelectorName**.
- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

This manual describes the Xerox LOOPS items (functions, methods, etc.) by using the following template:

Purpose:	Gives a short statement of what the item does.		
Behavior:	Provides the details of how the item operates.		
Arguments:	Describes each argument in the following format: <table><tr><td><i>argument</i></td><td>Description</td></tr></table>	<i>argument</i>	Description
<i>argument</i>	Description		
Returns:	States what the item returns, and does not appear if the item does not return a value. The phrase "Used as a side effect only." means that the purpose of the item is to perform a computation or action that is independent of any returned value, not to return a particular value.		
Categories:	A way to group related methods. For example, all the methods related to Masterscope on the class FileBrowser have the category Masterscope, not FileBrowser . This item appears only for methods.		
Specializes:	The next higher class in the class hierarchy that contains a method with the same selector. For example, RectangularWindow.Open can specialize Window.Open . This appears only for methods.		
Specializations:	The next lower class in the class hierarchy that contains a method with the same selector. For example, Window.Open is a specialization of RectangularWindow.Open . This appears only for methods.		
Example:	An example is often included to show how to use the item and what result it produces. Some examples may appear differently on your system, depending on the settings of various print flags. See Chapter 18, Reading and Printing, for details.		

Style Sheet Addenda

Here are some guidelines I used when writing the LOOPS manuals. Items appear in rather random order.

- Avoid contractions.
- Avoid subscripts. Use WORD1 rather than WORD to avoid inconsistent line leading.
- Avoid wording that starts "Note that..." or "Notice that...". Either make it a note with correct format or eliminate the "Note that".
- Use semicolons rather than m-dashes.
- Each item in the template starts with an initial capital letter; e.g., "Describes..."
- The arguments are identical in the call and in the argument description.
- Parentheses appear around expressions and square brackets appear around the name of the functionality.
- The arrow in the expression is the NS character ←, not _ . These characters appear similarly when printed, but differently on the screen. See the section, "Special Notes and Cautions," for details.
- A period appears after the word None, after argument descriptions, and Returns: item.
- Items are set to or return T (instead of true).

- Menus contain options, not items or selections.
- You drag (not roll) the mouse to the right of a menu option to see its submenu.
- Use "above" and "below" when referring to things in the same section, section numbers and names when referring to things in the same chapter, and chapter numbers and names when referring to things in another chapter.
- Please study the following style sheet carefully before you start to edit. The various appearances of active value and annotated values are especially crazy making.

These things appear in **bold**:

class variables
functions
instance variables
messages
methods
variables

ActiveValue - specific class/instance

active value - general information

activeValue - previous implementation of **ActiveValue**

annotatedValue - data type

AnnotatedValue - specific class

annotated values - general information

bitmap

data type

file package

filecoms

inspector

Lisp Library package

localState - instance variable

non-NIL

prettyprints

supers list

Paragraph Formatting

The text has the following format:

Paragraph-Looks Menu

APPLY SHOW NEUTRAL

Left Right Centered Justified Page-Heading type: {}

Line leading: {}pts Para Leading: {}pts Special Locn: X {}picas, Y {}picas

New Page: ~~Before~~ **After** Display mode: ~~Hardcopy~~ Keep: ~~Heading~~

Tab Type: **Left Right Centered Decimal Dotted Leader** Default Tab Size: {}

	13.0	42.0
	13.0	

0 6 12 18 24 30 36 42

Bulleted lists have the following format:

Paragraph-Looks Menu

APPLY SHOW NEUTRAL

Left Right Centered Justified Page-Heading type: {}

Line leading: {}pts Para Leading: {}pts Special Locn: X {}picas, Y {}picas

New Page: ~~Before~~ **After** Display mode: ~~Hardcopy~~ Keep: ~~Heading~~

Tab Type: **Left Right Centered Decimal Dotted Leader** Default Tab Size: {}

	13.0	42.0
	14.0	

0 6 12 18 24 30 36 42

The template has the following format:

Paragraph-Looks Menu

APPLY SHOW NEUTRAL

Left Right Centered Justified Page-Heading type: {}

Line leading: {}pts Para Leading: {}pts Special Locn: X {}picas, Y {}picas

New Page: ~~Before~~ **After** Display mode: ~~Hardcopy~~ Keep: ~~Heading~~

Tab Type: **Left Right Centered Decimal Dotted Leader** Default Tab Size: {}

0		42.0
	13.0	

0 6 12 18 24 30 36 42

The Arguments section of the template has the second line start at 18 instead of 13.

Paragraph-Looks Menu

APPLY SHOW NEUTRAL

Left **Right** **Centered** **Justified** **Page-Heading** type: {}
 Line leading: {0}pts Para Leading: {10}pts Special Locn: X {0}picas, Y {0}picas
 New Page: **Before** **After** Display mode: **Hardcopy** Keep: **Heading**
 Tab Type: **Left** **Right** **Centered** **Decimal** **Dotted** **Leader** Default Tab Size: {}

0 18.0 42.0

0 6 12 18 24 30 36 42

Page Layout

Page numbering, especially "text before", varies with the chapter.

Page Layout Menu

APPLY SHOW

For page: **First(&Default)** **Other Left** **Other Right**
 Starting Page #: {} Paper Size: **Letter** **Legal** **A4** **Landscape**

Page numbers: **No** **Yes** X: {4,5} Y: {1,25} Format: **123** **xiv** **XIV**
 Alignment: **Left** **Centered** **Right**
 Textbefore number: {16-} Textafter number: {}
 Margins: Left {4,5} Right {4,5} Top {4,5} Bottom {4,5}
 Columns: {1} Col Width: {42,0} Space between cols: {0,0}

Page Layout Menu

APPLY SHOW

For page: **First(&Default)** **Other Left** **Other Right**
 Starting Page #: {} Paper Size: **Letter** **Legal** **A4** **Landscape**

Page numbers: **No** **Yes** X: {46,5} Y: {1,25} Format: **123** **xiv** **XIV**
 Alignment: **Left** **Centered** **Right**
 Textbefore number: {16-} Textafter number: {}
 Margins: Left {4,5} Right {4,5} Top {4,5} Bottom {4,5}
 Columns: {1} Col Width: {42,0} Space between cols: {0,0}

Bitmaps, Graphs, and Sketches

Scale for bitmaps is 0.8.

Special Notes and Cautions

Make sure you have changed the underscore to be a left arrow before loading any files. To do this,

- Enter the following commands into your Executive:

```
(GETCHARBITMAP (CHARCODE _) '(MODERN 10 MRR))
(EDITBM IT)
```

- When the bitmap editor appears, delete the underscore and insert the following left arrow:

```
.....
.....
.....
.....
.....
...X....
...XX...
..XXXXXX.
...XX...
...X....
.....
.....
.....
.....
```

- Finally, enter the following commands into your Executive to store the pattern:

```
(PUTCHARBITMAP (CHARCODE _) '(MODERN 10 MRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(MODERN 10 BRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(TERMINAL 10 MRR) IT)

(PUTCHARBITMAP (CHARCODE _) '(TERMINAL 10 BRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(TERMINAL 12 BRR) IT)
```

A

access-oriented programming 1-1,9
 accessing
 a class, errors 11-6
 an instance, errors 11-6
 class variable 5-16
 data in classes 5-13
 instance variable 5-19
 active value 1-9; 5-5; 8-1
 adding 8-17
 breaking and tracing 8-10
 bypassing 8-22
 creating 8-23
 deleting 8-17
 errors 11-9
 in class structures 8-27
 printing 17-11
 replacing 8-17
 ActiveValue 8-1
 methods 8-16
 shared 8-22
 specializations 8-2
Add (*Inspector Submenu Option*) 18-4,10
Add (*Method of Class*) 3-12
Add (**AddMethod**) (*Browser Menu Option*) 10-20,32
Add Category Menu (*Browser Menu Option*) 10-10,25
Add file to browser (*Browser Submenu Option*) 10-26
Add/Delete (*Inspector Menu Option*) 18-4,10
AddActiveValue (*Method of ActiveValue*) 8-17
AddCIV (*Function*) 3-14
AddCV (*Browser Submenu Option*) 10-20
AddCV (*Function*) 3-13
AddCV (*Method of Class*) 3-13
AddIV (*Browser Submenu Option*) 10-20
AddIV (*Function*) 2-10
AddIV (*Method of Class*) 3-14
AddIV (*Method of Object*) 2-11
AddMethod (*Browser Submenu Option*) 10-21
 adding an active value 8-17
AddRoot (*Browser Menu Option*) 10-10,25
AddRoot (*Method of LatticeBrowser*) 10-36
AddSubs (*Browser Menu Option*) 10-31
AddSubs! (*Browser Submenu Option*) 10-31
AddSuper (*Browser Submenu Option*) 10-21
AfterMove (*Method of Window*) 19-3
AfterReshape (*Method of Window*) 19-3
all (*Browser Submenu Option*) 10-26
All (*Inspector Menu Option*) 18-8
AllInstances (*Method of Class*) 3-24
AllInstances! (*Method of Class*) 3-25
AllMethodCategories (*Method of Class*) 6-2
AllSubClasses (*Function*) 3-28
AllValues (*Inspector Menu Option*) 18-3,10
AnalyzeFile (*Browser Submenu Option*) 10-28
 annotated value 8-24
 errors 11-9
 explicit control over 8-25
 restoring 8-26
 saving 8-26
AnnotatedValue (*Class*) 8-24
AnnotatedValue? (*Macro*) 9-2

Any (*Category*) 6-1
AppendSuperValue (*Class*) 8-11
ApplyMethod (*Function*) 6-7
ArgsOfMethodBeingCompiled (*Macro*) 6-10
associatedFiles (*Browser Submenu Option*) 10-26
AttachLispWindow (*Method of Window*) 19-22
AVPrintSource (*Method of ActiveValue*) 8-26; 17-11
AVPrintSource (*Specialization of ActiveValue*) 8-16

B

background menu 10-3
Blink (*Method of Window*) 19-3
Box/UnBoxNode (*Browser Menu Option*) 10-17
BoxNode (*Browser Menu Option*) 10-32
BoxNode (*Method of LatticeBrowser*) 10-37
Break on Access (*Inspector Submenu Option*) 18-6
Break on Put (*Inspector Submenu Option*) 18-6
BreakFunction (*Browser Submenu Option*) 10-29
 breaking 12-1
BreakIt (*Function*) 12-4
BreakIt (*Inspector Menu Option*) 18-6
BreakIt (*Method of Object*) 12-3
BreakMethod (*Browser Menu Option*) 10-20
BreakMethod (*Method of Class*) 12-1
BreakOnPut (*Class*) 8-10
BreakOnPutOrGet (*Class*) 8-10
BrokenVariables (*Global Variable*) 12-6
Browse (*Function*) 10-7
Browse (*Inspector Menu Option*) 18-8
Browse (*Method of LatticeBrowser*) 10-6,37
Browse Class (*Submenu Option*) 10-3,4
Browse File (*Submenu Option*) 10-3,5
Browse Supers (*Submenu Option*) 10-4
BrowseFile (*Method of FileBrowser*) 10-6
 browser 10-1
BrowserObjects (*Method of LatticeBrowser*) 10-38
BrowseSupers (*Inspector Submenu Option*) 18-8
Bury (*Method of Window*) 20-4
ButtonEventFn (*Method of Window*) 19-15

C

cache 15-2,3
CalledFns (*Function*) 6-9
CallsFunction (*Browser Submenu Option*) 10-28
 categories of a method 6-1
CategorizeMethods (*Browser Submenu Option*) 10-19
CategorizeMethods (*Method of Class*) 6-2
Change display mode (*Browser Menu Option*) 10-25
ChangeClass (*Method of Object*) 2-16
ChangeFontSize (*Browser Submenu Option*) 10-8
ChangeFontSize (*Method of LatticeBrowser*) 10-38
ChangeFormat (*Method of LatticeBrowser*) 10-38
ChangeMaxLabelSize (*Method of LatticeBrowser*) 10-39
ChangeMethodCategory (*Browser Submenu Option*) 10-19
ChangeMethodCategory (*Method of Class*) 6-3
 Changetran 5-13
Check (*Browser Submenu Option*) 10-27
CheckFile (*Browser Submenu Option*) 10-28
 class 1-3,6; 3-1
 contents 3-1

- copying 3-23
- creating 3-1
- destroying 3-5
- editing 3-10; 13-1
- enumerating instances 3-24
- manipulating 2-16; 3-16
- modifying 3-11
- printing 17-4
- querying the structure 3-17
- reading data in 5-13
- renaming 3-16
- storing data in 5-13
- Class** (Category) 6-1
- Class** (Inspector Menu Option) 18-3
- Class** (Macro) 2-17
- Class** (Method of Object) 2-17
- class browser 10-2
 - automatic update 10-52
 - menu interface 10-8
- class inspector 18-7
- class variable 1-4; 5-5
 - accessing 5-16
 - for class LatticeBrowser 10-34
- Class?** (Macro) 9-2
- ClassDoc** (Browser Submenu Option) 10-14
- CLASSES** (File Package Command) 14-4
- ClassIV inspector 18-9
- ClassName** (Function) 2-18; 3-17
- ClassName** (Method of Object) 2-18
- ClassNameOfMethodOwner** (Macro) 6-10
- CLEANUP** file (Browser Menu Option) 10-30
- CleanUp File** (Submenu Option) 10-5
- Clear** (Method of Window) 19-4
- ClearAllCaches** (Function) 15-3; 20-3
- ClearAllCaches** (Variable) 20-3
- ClearLabelCache** (Method of LatticeBrowser) 10-40
- ClearMenuCache** (Method of Window) 19-15
- ClearPromptWindow** (Method of Window) 19-10
- Close** (Method of Window) 19-4
- ClosePromptWindow** (Method of Window) 19-10
- compact accessing forms 5-10
- ConformToClass** (Method of Object) 2-12
- Copy** (Method of Class) 3-23
- Copy** (CopyMethodTo) (Browser Menu Option) 10-23,32
- CopyActiveValue** (Method of ActiveValue) 8-22
- CopyActiveValue** (Specialization of ActiveValue) 8-16
- CopyCV** (Method of Class) 3-23
- CopyCVTo** (Browser Submenu Option) 10-23
- CopyDeep** (Method of Object) 2-19
- copying
 - class 3-23
 - instance 2-19
- CopyIV** (Method of Class) 3-24
- CopyIVTo** (Browser Submenu Option) 10-23
- CopyMethodTo** (Browser Submenu Option) 10-23
- CopyShallow** (Method of Object) 2-20
- create annotatedValue** (Macro) 8-25
- CreateClass** (Method of Metaclass) 3-3
- CreateWindow** (Method of NonRectangular Window) 19-19
- CreateWindow** (Method of Window) 19-22
- creating
 - active value 8-23
 - class 3-1
 - instance 2-4

- method 6-4
- record 18-17
- CursorInside?** (Method of Window) 19-4
- CV, see class variable
- CVDoc** (Browser Submenu Option) 10-14
- CVMissing** (Method of Class) 11-2
- CVValueMissing** (Method of Class) 11-3
- D**
- data type predicate 9-1
- DefaultActiveValueClassName** (Variable) 17-12
- DEFCLASS** (NLambda NoSpread Function) 14-4
- DEFCLASSES** (NLambda NoSpread Function) 14-4
- DefineClass** (Function) 3-2
- DefineMethod** (Function) 6-5
- defining a metaclass 4-5
- DEFINST** (NLambda NoSpread Function) 14-6
- DEFINSTANCES** (NLambda NoSpread Function) 14-6
- Delete** (Inspector Submenu Option) 18-4,10
- Delete** (Method of Class) 3-12
- Delete** (DeleteMethod) (Browser Menu Option) 10-21,32
- DeleteActiveValue** (Method of ActiveValue) 8-18
- DeleteCIV** (Function) 3-15
- DeleteClass** (Browser Submenu Option) 10-22
- DeleteClassProp** (Function) 3-13
- DeleteCV** (Browser Submenu Option) 10-21
- DeleteCV** (Function) 3-14
- DeleteFromBrowser** (Browser Menu Option) 10-16,31
- DeleteFromBrowser** (Browser Submenu Option) 10-16
- DeleteFromBrowser** (Method of LatticeBrowser) 10-40
- DeleteIV** (Browser Submenu Option) 10-21
- DeleteIV** (Function) 2-11
- DeleteIV** (Method of Object) 2-12
- DeleteMethod** (Browser Submenu Option) 10-22
- DeleteMethod** (Function) 6-5
- DeleteSubtreeFromBrowser** (Browser Submenu Option) 10-16
- DeleteSubtreeFromBrowser** (Method of LatticeBrowser) 10-40
- deleting an active value 8-17
- DelFromFile** (Method of Object) 14-9
- Destroy** (Method of Class) 3-5
- Destroy** (Method of Object) 2-15
- Destroy** (Method of Window) 19-4
- Destroy!** (Method of Class) 3-6
- Destroy!** (Method of Object) 2-15
- DestroyClass** (Method of Class) 3-6
- destroying
 - class 3-5
 - instance 2-15
 - method 6-4
- DestroyInstance** (Method of Class) 2-15
- DetachLispWindow** (Method of Window) 19-22
- Doc** (ClassDoc) (Browser Menu Option) 10-13,30
- DoFringeMethods** (Function) 6-7
- DoMethod** (Function) 6-7
- DontSave** (Instance Variable Property Name) 14-10
- dynamic mixin 3-4

E

- Edit** (Browser Submenu Option) 10-27

Edit (*Inspector Menu Option*) 18-4,8
Edit (*Method of Class*) 3-10; 13-1
Edit (*Method of Object*) 13-5
Edit (EditClass) (*Browser Menu Option*) 10-24,32
Edit FileComs (*Browser Menu Option*) 10-28
Edit Filecoms (*Submenu Option*) 10-5
Edit Functions (*Browser Submenu Option*) 10-29
Edit! (*Method of Class*) 13-3
EditCategory (*Browser Submenu Option*) 10-19
EditClass (*Browser Submenu Option*) 10-24
EditClass! (*Browser Submenu Option*) 10-24
EditComs (*Browser Submenu Option*) 10-29
EditFns (*Browser Submenu Option*) 10-29
EditIcon (*Method of NonRectangular Window*) 19-19
 editing 13-1
 a class 13-1
 class 3-10
 description of window 13-2
 method 6-4
EditInstances (*Browser Submenu Option*) 10-29
EditMacros (*Browser Submenu Option*) 10-29
EditMask (*Method of NonRectangular Window*) 19-20
EditMethod (*Browser Submenu Option*) 10-18
EditMethod (*Method of Class*) 6-5
EditMethod! (*Browser Submenu Option*) 10-18
EditMethodObject (*Browser Submenu Option*) 10-18
EditRecords (*Browser Submenu Option*) 10-29
EditVars (*Browser Submenu Option*) 10-29
 enumerating instances of a class 3-24
 error handling 11-1
 error message 11-5
ErrorOnNameConflict (*Variable*) 2-4; 11-2
 escaping from message syntax 6-6
ExplicitFnActiveValue (*Class*) 8-8

F

fetch annotatedValue of (*Macro*) 8-25
FetchMethod (*Method of Class*) 7-5
 file
 storing 14-10
 file browser 10-2
 menu interface 10-24
 file manager 1-11; 14-1
 commands 14-3
FileBrowse (*Function*) 10-7
FileIn (*Method of Class*) 14-6
FileOut (*Method of Class*) 17-4
FileOut (*Method of Object*) 14-11; 17-8
FILES? (*Function*) 14-7
FirstFetchAV (*Class*) 8-12
FlashNode (*Method of LatticeBrowser*) 10-41
FlipNode (*Method of LatticeBrowser*) 10-41
Fringe (*Method of Class*) 3-27
 function calling 3-2

G

garbage collection 15-1
Get (*Method of Object*) 5-6
GetClass (*Function*) 5-14
GetClassHere (*Function*) 5-16
GetClassIV (*Function*) 5-20
GetClassIVHere (*Function*) 5-20
GetClassOnly (*Function*) 5-15
GetClassProp (*Method of Class*) 3-18

GetClassValue (*Function*) 5-8,16
GetClassValueOnly (*Function*) 5-9,18; 8-22
GetCVHere (*Function*) 5-18
GetDisplayLabel (*Method of LatticeBrowser*) 10-41
GetIt (*Function*) 5-1
GetItHere (*Function*) 5-3
GetItOnly (*Function*) 5-2
GetIVHere (*Function*) 5-10
GetLabel (*Method of LatticeBrowser*) 10-41
GetLispClass (*Function*) 4-3
GetObjectNames (*Function*) 2-4
GetObjFromUID (*Function*) 17-15
GetPromptWindow (*Method of Window*) 19-10
GetProp (*Method of Window*) 19-5
GetSubs (*Method of InstanceBrowser*) 10-50
GetSubs (*Method of LatticeBrowser*) 10-42
GettingWrappedValue (*Message*) 1-9
GetValue (*Function*) 4-3; 5-6
GetValue (*Macro*) 15-2
GetValueOnly (*Function*) 5-7; 8-22
GetWrappedValue (*Method of ActiveValue*) 8-20
GetWrappedValue (*Method of LispWindowAV*) 19-23
GetWrappedValue (*Specialization of ActiveValue*) 8-16
GetWrappedValueOnly (*Method of ActiveValue*) 8-20
 global cache 15-2,3
 Grapher 1-11; 10-1,33
GraphFits (*Method of LatticeBrowser*) 10-42

H

Hardcopy (*Method of Window*) 19-5
Hardcopy file (*Browser Submenu Option*) 10-30
HardcopyToFile (*Method of Window*) 19-5
HardcopyToPrinter (*Method of Window*) 19-5
HasAttribute (*Method of Class*) 3-18
HasAttribute! (*Method of Class*) 3-19
HasCV (*Method of Class*) 3-19
HasCV (*Method of Object*) 2-21
HasItem (*Method of Class*) 3-20
HasIV (*Method of Class*) 3-21
HasIV (*Method of Object*) 2-21
HasIV! (*Method of Class*) 3-21
HasLispWindow (*Method of Window*) 19-23
HasObject (*Method of LatticeBrowser*) 10-42
HasUID? (*Function*) 17-14
HELPCHECK (*Function*) 11-1
HighlightNode (*Method of LatticeBrowser*) 10-42

I

IconTitle (*Method of LatticeBrowser*) 10-43
IconWindow (*Class*) 19-20
ImplementsMethod (*Browser Submenu Option*) 10-28
in-supers-of (*Iterative Statement Operator*) 9-3
IndexedObject (*Class*) 3-25
IndirectVariable (*Class*) 8-3
 inheritance 1-6; 3-7,27
InheritedValue (*Method of InheritingAV*) 8-14
InheritingAV (*Class*) 8-14
InPlace (*Browser Submenu Option*) 10-8
Inspect (*Inspector Menu Option*) 18-7,10
Inspect (*Method of Object*) 2-22; 18-2
InspectClass (*Browser Submenu Option*) 10-24
InspectFetch (*Method of Object*) 18-12
 inspector 1-10; 18-1

customizing 18-15
InspectPropCommand (*Method of Object*) 18-13
InspectProperties (*Method of Object*) 18-13
InspectStore (*Method of Object*) 18-13
InspectTitle (*Method of Object*) 18-14
InspectValueCommand (*Method of Object*) 18-14
InstallEditSource (*Method of Class*) 13-3
InstallEditSource (*Method of Object*) 13-6
 instance 1-4
 accessing data in 5-4
 copying 2-19
 creating 2-4
 data storage at creation time 2-8
 destroying 2-15
 editing 13-5
 naming 2-1
 querying structure 2-21
 instance browser 10-3,50
 instance inspector 18-2
 instance variable 1-4; 5-5
 access 15-1
 accessing 5-19
 changing number of 2-10
 delimiters 2-11
 for class InstanceBrowser 10-50
 for class LatticeBrowser 10-33
Instance? (*Macro*) 9-2
INSTANCES (*File Package Command*) 14-5
InstOf (*Method of Object*) 2-18
InstOf! (*Method of Object*) 2-19
Internal (*Category*) 6-1
Invert (*Method of NonRectangular Window*) 19-20
Invert (*Method of Window*) 19-5
ItemMenu (*Method of Window*) 19-15
 iterative operator 9-3
 IV, see instance variable
IVDoc (*Browser Submenu Option*) 10-14
IVMissing (*Method of Object*) 2-12; 11-3
IVs (*Inspector Menu Option*) 18-4,10
IVValueMissing (*Method of Object*) 2-8; 11-4

L
 lattice 1-1
 lattice browser 10-2
Lattice/Tree (*Browser Submenu Option*) 10-9
 left column menu
 class inspector 18-8
 ClassIV inspector 18-10
 instance inspector 18-4
 left menu
 class, meta, and supers browsers 10-11
 file browser 10-30
 instance browser 10-51
LeftSelection (*Method of LatticeBrowser*) 10-43
LeftSelection (*Method of Window*) 19-16
LeftShiftSelect (*Method of LatticeBrowser*) 10-43
 Lisp window 19-22
LispClassTable (*Global Variable*) 4-4
LispUserFilesForLoops (*Variable*) 20-2
LispWindowAV (*Class*) 8-10
ListAttribute (*Method of Class*) 3-21
ListAttribute (*Method of Object*) 2-22
ListAttribute! (*Method of Class*) 3-22
ListAttribute! (*Method of Object*) 2-23
LOAD (*Function*) 14-2
Load PROP file (*Browser Submenu Option*) 10-30
LOADFNS (*Function*) 14-3
 loading a file 14-2

LoadLoopsForms (*Variable*) 20-2
Local (*Inspector Menu Option*) 18-8
 local cache 15-2,3
LocalStateActiveValue (*Class*) 8-6
LocalValues (*Inspector Menu Option*) 18-4,10
 LOOPS icon 10-4
Loops Icon (*Menu Option*) 10-3
LoopsDate (*Variable*) 20-2
LoopsDebugFlg (*Variable*) 11-2
LOOPSDIRECTORY (*Variable*) 20-2
LOOPFILES (*Variable*) 20-2
LoopsHelp (*NoSpread Function*) 11-2
LoopsIcon (*Class*) 19-21
LOOPSLIBRARYDIRECTORY (*Variable*) 20-2
LoopsPatchFiles (*Variable*) 20-2
LOOPUSERSDIRECTORY (*Variable*) 20-2
LOOPUSERSRULESDIRECTORY (*Variable*) 20-2
LoopsVersion (*Variable*) 20-1

M

MakeEditSource (*Method of Class*) 13-4
MakeEditSource (*Method of Object*) 13-5
MAKEFILE (*Function*) 14-11
MakeFileSource (*Method of Object*) 14-11
MakeFullEditSource (*Method of Class*) 13-4
MakeFunctionMenu (*Browser Submenu Option*) 10-29
 manipulating
 a file 14-1
 a class 2-16; 3-16
MapObjectUID (*Function*) 17-15
 Masterscope 1-10
Masterscope (*Category*) 6-2
MaxLatticeHeight (*Variable*) 10-48
MaxLatticeWidth (*Variable*) 10-48
 menu 19-14
 caching 19-18
 item structure 19-17
 message 1-3
 message sending 3-2
 message sending form 7-1; 18-16
 message syntax, escaping from 6-6
MessageNotUnderstood (*Method of AnnotatedValue*) 8-26
MessageNotUnderstood (*Method of Object*) 11-5
MessageNotUnderstood (*Method of Tofu*) 4-7
 metaclass 1-6; 5-13
 AbstractClass 4-2
 Class 4-1
 defining 4-5
 DestroyedClass 4-2
 MetaClass 4-2
 metaclass browser 10-2
 menu interface 10-8
METH (*NLambda NoSpread Function*) 14-5
 method 1-3; 6-1
 creating 6-4
 destroying 6-4
 editing 6-4
 for class InstanceBrowser 10-50
 for window 19-2
 printing 17-12
method (*Definer*)
 structure 6-3
 method lookup 15-3
MethodDoc (*Browser Submenu Option*) 10-14
MethodDoc (*Method of Class*) 17-13
MethodMenu (*Browser Submenu Option*) 10-19

MethodNotFound (*Method of Tofu*) 4-7
METHODS (*File Package Command*) 14-5
Methods (EditMethod) (*Browser Menu Option*) 10-18,32
MethodSummary (*Browser Submenu Option*) 10-13
MethodSummary (*Method of Class*) 17-13
 middle menu
 class, meta, and supers browsers 10-17
 file browser 10-31
 instance browser 10-52
MiddleSelection (*Method of Window*) 19-16
MiddleShiftSelect (*Method of LatticeBrowser*) 10-44
 modifying a class 3-11
MousePackage (*Method*) 19-6
MouseReadtable (*Method*) 19-6
Move (*Method of Window*) 19-5
Move (MoveMethodTo) (*Browser Menu Option*) 10-22,32
MoveClassVariable (*Function*) 2-14
MoveCVTo (*Browser Submenu Option*) 10-22
MoveIVTo (*Browser Submenu Option*) 10-22
MoveMethod (*Function*) 6-8
MoveMethod (*Method of Class*) 6-9
MoveMethodsToFile (*Function*) 6-9
MoveMethodTo (*Browser Submenu Option*) 10-22
MoveSuperTo (*Browser Submenu Option*) 10-22
MoveToFile (*Browser Submenu Option*) 10-22
MoveToFile (*Method of Class*) 14-9
MoveToFile! (*Browser Submenu Option*) 10-22
MoveToFile! (*Method of Class*) 14-9
MoveVariable (*Function*) 2-14
 moving a variable 2-13
 multiple references of objects 15-1

N

naming an instance 2-1
 naming an object, errors 11-8
NestedNotSetValue (*Class*) 8-16
New (*Method of Class*) 2-5
New (*Method of Metaclass*) 3-3; 4-5
NewClass (*Method of Class*) 3-3
NewInstance (*Browser Submenu Option*) 10-21,32
NewInstance (*Method of Object*) 2-6
NewItem (*Method of LatticeBrowser*) 10-44
NewPath (*Method of InstanceBrowser*) 10-51
NewWithValues (*Method of Class*) 2-7
NiceMenu (*Function*) 19-13
NodeRegion (*Method of LatticeBrowser*) 10-45
NonRectangularWindow (*Class*) 19-19
NotSetValue (*Class*) 8-15
NotSetValue (*Macro*) 2-9
NoUpdatePermittedAV (*Class*) 8-9
NoValueFound (*Macro*) 2-24
NoValueFound (*Variable*) 2-24

O

object 1-2
 multiple references 15-1
 printing 17-8
 saving on a file 14-6
 storing data in 1-4
Object (*Category*) 6-1
 object-oriented programming 1-1,3
Object? (*Macro*) 9-1
ObjectAlwaysPPFlag (*Variable*) 17-3

ObjectDontPPFlag (*Variable*) 17-3
ObjectFromLabel (*Method of LatticeBrowser*) 10-45
ObjectModified (*Method of Object*) 14-8
OldInstance (*Method of Object*) 14-10
OnFile (*Method of Class*) 14-8
Open (*Method of Window*) 19-6
 opening a browser 10-3
OptionalLispuserFiles (*Variable*) 9-2
OverridesMethod (*Browser Submenu Option*) 10-28

P

Paint (*Method of Window*) 19-6
PositionNode (*Method of LatticeBrowser*) 10-45
PP (*Browser Submenu Option*) 10-12
PP (*Method of Class*) 17-5
PP (*Method of Object*) 17-9
PP! (*Browser Submenu Option*) 10-12
PP! (*Method of Class*) 17-6
PP! (*Method of Object*) 17-9
PPDefault (*Variable*) 17-12
PPMethod (*Browser Submenu Option*) 10-13
PPMethod (*Method of Class*) 17-12
PPV! (*Browser Submenu Option*) 10-12
PPV! (*Method of Class*) 17-7
PPV! (*Method of Object*) 17-10
PrettyPrintClass (*Function*) 14-11
PrettyPrintInstance (*Function*) 14-11
PrintCategories (*Browser Submenu Option*) 10-12
 printing 17-1
 variables affecting 17-2
PrintOn (*Method of IndexedObject*) 3-25
PrintOn (*Method of Object*) 17-8
PrintSummary (*Browser Menu Option*) 10-12,30
PrintSummary (*Browser Submenu Option*) 10-13
 procedure-oriented programming 1-1,2,3
 programming paradigms 1-1
 prompt window 19-9
PromptEval (*Function*) 19-11
PromptForList (*Method of Window*) 19-11
PromptForString (*Method of Window*) 19-12
PromptForWord (*Method of Window*) 19-12
PromptPrint (*Method of Window*) 19-13
PromptRead (*Function*) 19-13
Properties (*Inspector Menu Option*) 18-5,11
Prototype (*Method of Class*) 3-26
 pseudoclass 8-24
 pseudoclasses 4-2
 pseudoinstances 4-2; 8-24
Public (*Category*) 6-1
Put (*Method of Object*) 5-7
PutClass (*Function*) 5-14
PutClassIV (*Function*) 5-21
PutClassOnly (*Function*) 5-15
PutClassValue (*Function*) 5-8,17
PutClassValueOnly (*Function*) 5-9,18; 8-22
PutCVHere (*Function*) 5-19
PutIt (*Function*) 5-3
PutItOnly (*Function*) 5-4
PutSavedValue (*Function*) 19-21
PuttingWrappedValue (*Message*) 1-9
PutValue (*Function*) 4-3; 5-6
PutValue (*Inspector Menu Option*) 18-5
PutValue (*Macro*) 15-2
PutValueOnly (*Function*) 5-7; 8-22
PutValueOnly (*Inspector Submenu Option*) 18-5
PutWrappedValue (*Method of ActiveValue*) 8-21

PutWrappedValue (Method of *LispWindowAV*) 19-23
PutWrappedValue (Specialization of *ActiveValue*) 8-16
PutWrappedValueOnly (Method of *ActiveValue*) 8-21

Q

querying
 structure of a class 3-17
 structure of an instance 2-21

R

reading 17-1
Recompute (Browser Menu Option) 10-8,25
Recompute (Browser Submenu Option) 10-8
Recompute (Method of *LatticeBrowser*) 10-46
RecomputeInPlace (Method of *LatticeBrowser*) 10-46
RecomputeLabels (Browser Submenu Option) 10-8
RecomputeLabels (Method of *LatticeBrowser*) 10-46
 record
 creating 18-17
Refetch (Inspector Menu Option) 18-4,8,10
RemoveFromBadList (Browser Submenu Option) 10-10
RemoveHighlights (Method of *LatticeBrowser*) 10-46
RemoveShading (Method of *LatticeBrowser*) 10-47
Rename (Method of Class) 3-16
Rename (Method of Object) 2-3
Rename (**RenameMethod**) (Browser Menu Option) 10-23,32
RenameClass (Browser Submenu Option) 10-23
RenameCV (Browser Submenu Option) 10-23
RenameIV (Browser Submenu Option) 10-23
RenameMethod (Browser Submenu Option) 10-23
RenameMethod (Function) 6-8
RenameVariable (Function) 2-14
replace annotatedValue of (Macro) 8-25
ReplaceActiveValue (Method of *ActiveValue*) 8-19
ReplaceMeAV (Class) 8-15
ReplaceSupers (Method of Class) 3-15
 replacing an active value 5-17
RetireMethod (Browser Submenu Option) 10-23
 right column menu
 class inspector 18-8
 ClassIV inspector 18-10
 instance inspector 18-6
RightSelection (Method of Window) 19-16
 rule-oriented programming 1-1

S

Save Value (Inspector Menu Option) 18-4,7,11
SavedValue (Function) 19-21
SaveInIT (Method of *LatticeBrowser*) 10-47
SaveInstance (Method of Object) 14-8
SaveInstance? (Method of Object) 14-9
SaveValue (Browser Submenu Option) 10-8
 saving
 an object on a file 14-6
ScrollWindow (Method of Window) 19-7
 SEdit 1-10
Select file (Browser Submenu Option) 10-26
selectedFile (Browser Submenu Option) 10-25

SelectFile (Lambda NoSpread Function) 19-14
 selector 1-3
SelectorOfMethodBeingCompiled (Macro) 6-10
SelectorsWithBreak (Method of Class) 12-3
SEND (Function) 7-2
SEND (Macro) 7-2
 sending
 a message, errors 11-7
SendsMessage (Browser Submenu Option) 10-28
SetName (Method of Class) 3-16
SetName (Method of Object) 2-2
SetProp (Method of Window) 19-7
ShadeNode (Method of *LatticeBrowser*) 10-47
Shape (Method of *NonRectangular Window*) 19-20
Shape (Method of Window) 19-7
Shape? (Method of Window) 19-8
ShapeToHold (Browser Submenu Option) 10-8
ShapeToHold (Method of *LatticeBrowser*) 10-48
Show (Method of *LatticeBrowser*) 10-48
Shrink (Method of *LatticeBrowser*) 10-48
Shrink (Method of Window) 19-8
Snap (Method of Window) 19-8
Specialize (Method of Class) 3-27
SpecializeClass (Browser Submenu Option) 10-21
SpecializedClass (Browser Submenu Option) 10-32
SpecializeMethod (Browser Submenu Option) 10-21
SpecializesMethod (Browser Submenu Option) 10-28
 stack method macros 6-10
 storing
 a file 14-10
 data in objects 1-4
SubBrowser (Browser Menu Option) 10-16,31
SubBrowser (Method of *LatticeBrowser*) 10-49
 subclass 1-6
Subclass (Method of Class) 3-28
SubClasses (Method of Class) 3-28
SubclassResponsibility (Macro) 6-6
Substitute (Browser Submenu Option) 10-27
SubsTree (Function) 3-29
 superclass 1-7,8
SuperMethodNotFound (Method of *Tofu*) 4-7
 supers browser 10-2
 menu interface 10-8
 system function 20-1
 system variable 20-1

T

THESE-INSTANCES (File Package Command) 14-5
 title
 class inspector 18-7
 ClassIV inspector 18-9
 instance inspector 18-2
 title bar menu
 class inspector 18-8
 class, meta, and supers browsers 10-8
 ClassIV inspector 18-9
 file browser 10-25
 instance browser 10-51
 instance inspector 18-3
TitleCommand (Method of Object) 18-15
TitleSelection (Method of *LatticeBrowser*) 10-49
TitleSelection (Method of Window) 19-16
 Tofu 4-6
 tools 1-10

ToTop (*Method of Window*) 19-8
Trace on Access (*Inspector Submenu Option*) 18-6
Trace on Put (*Inspector Submenu Option*) 18-6
TraceFunction (*Browser Submenu Option*) 10-29
TraceIt (*Function*) 12-6
TraceIt (*Inspector Menu Option*) 18-6
TraceIt (*Method of Object*) 12-5
TraceMethod (*Browser Submenu Option*) 10-20
TraceMethod (*Method of Class*) 12-2
TraceOnPut (*Class*) 8-10
TraceOnPutOrGet (*Class*) 8-10
tracing 12-1
type? annotatedValue (*Macro*) 8-25
TypeInName (*Browser Menu Option*) 10-16,31

U

UID, see Unique Identifier
UID (*Function*) 17-15
UnbreakFunction (*Browser Submenu Option*) 10-29
UnBreakIt (*Function*) 12-6
UnBreakIt (*Inspector Menu Option*) 18-6
UnbreakMethod (*Browser Submenu Option*) 10-20
UnbreakMethod (*Method of Class*) 12-2
Understands (*Method of Object*) 9-3
UNDO (*Program Assistant Command*) 14-3
Unique Identifier 15-1; 17-14
UnmarkNodes (*Method of LatticeBrowser*) 10-49
UnSetName (*Method of Class*) 3-17
UnSetName (*Method of Object*) 2-3
Update (*Method of Window*) 19-9
UpdateClassBrowsers (*Function*) 10-52
UpdateClassBrowsers? (*Variable*) 10-52
Use saved value (*Inspector Submenu Option*) 18-5
user interface to inspector 18-2
Uses IV? (*Browser Menu Option*) 10-26
UsesCV (*Browser Submenu Option*) 10-28
UsesIV (*Browser Menu Option*) 10-33
UsesIV (*Browser Submenu Option*) 10-27
UsesLispVar (*Browser Submenu Option*) 10-28
UsesObject (*Browser Submenu Option*) 10-28

V

ValueFound (*Macro*) 2-25
variable 1-4

W

WhenMenuItemHeld (*Method of Window*) 19-17
WhereIs (*Browser Menu Option*) 10-14
WhereIs (*Method of Object*) 2-24
WhereIs (WhereIsMethod) (*Browser Menu Option*) 10-30
WhereIsCV (*Browser Submenu Option*) 10-15
WhereIsIV (*Browser Submenu Option*) 10-15
WhereIsMethod (*Browser Submenu Option*) 10-15
WhoHas (*Function*) 3-22
Window (*Class*) 19-1
WindowAfterMoveFn (*Function*) 19-9
WindowButtonEventFn (*Function*) 19-17
WindowRightButtonFn (*Function*) 19-17
WindowShapeFn (*Function*) 19-9
wrapped value 8-19
WrappingPrecedence (*Method of ActiveValue*) 8-18
WrappingPrecedence (*Specialization of ActiveValue*) 8-16

←
← (*Function*) 7-1
← (*Macro*) 7-1
←! (*Function*) 7-2
←! (*Macro*) 7-2
←@ (*Macro*) 5-12
←AV (*Macro*) 8-26
←IV (*Macro*) 7-2
←New (*Macro*) 2-6
←New (*NLambda NoSpread Macro*) 7-5
←Process (*Macro*) 16-1
←Process! (*Macro*) 16-2
←Proto (*Macro*) 7-3
←Super (*Macro*) 7-3
←Super (*Function*) 7-3
←Super? (*Macro*) 7-5
←SuperFringe (*Macro*) 7-5
←SuperFringe (*Function*) 7-5
←Try (*Macro*) 7-3

#

#, 18-1

\$

\$ (*Macro*) 2-2
\$ (*NLambda Function*) 2-2; 17-1
\$! (*Function*) 2-2
\$! (*Lambda Function*) 17-2
\$AV (*NLambda NoSpread Function*) 8-27
\$C (*NLambda Function*) 17-2

*

any (*Browser Submenu Option*) 10-27
EditAll (*Browser Submenu Option*) 10-27; 27
FEATURES (*Variable*) 20-2
hiddenFile (*Submenu Option*) 10-5
loadFile (*Submenu Option*) 10-5
newFile (*Submenu Option*) 10-5
NewFunction (*Browser Submenu Option*) 10-29
other (*Browser Submenu Option*) 10-27
SubstituteAll (*Browser Submenu Option*) 10-27

:

:initForm (*Property*) 2-9

?

?= 18-16

@

@ (*Macro*) 5-10
@* (*Macro*) 5-12

ntionally left blank]

[This page inte

Writer's Notes -- Production Details

This file includes notes on the production of *Xerox LOOPS Reference Manual*, Koto Release. This manual is packaged with the *Xerox LOOPS Release Notes* and *Xerox LOOPS Library Packages Manual* to form one binder, part number 3103310.

Writer: Rosie (Raven) Kontur

Printing Date: <DD> <MM> 1987

Files Needed

To edit or print the manual, make sure you have the following files loaded:

IMTOOLS
SKETCH
GRAPHER

Fonts Used

{ERIS}<LISP>FONTS>

Modern font

18-point bold
14-point bold
12-point bold
10-point regular
10-point italic
10-point bold

Terminal font

12-point bold
10-point regular
10-point bold

Printing Information

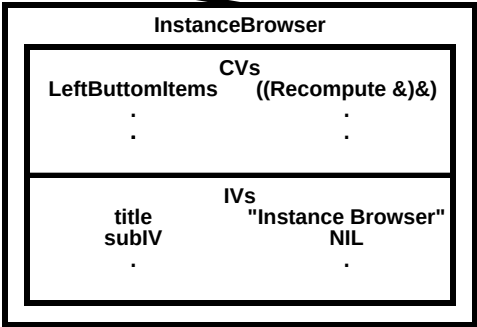
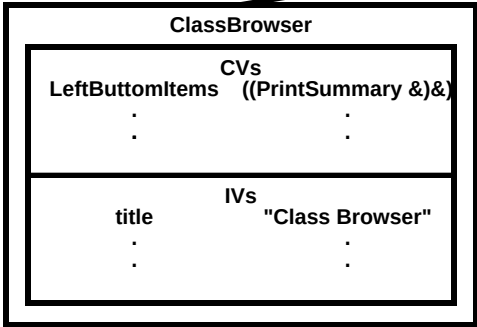
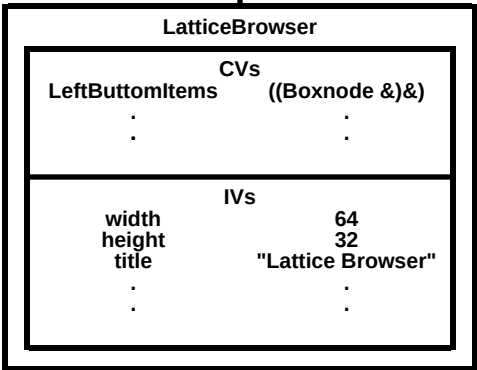
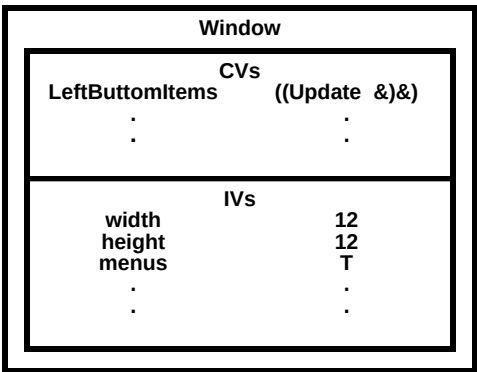
The manual was printed under a Lyric sysout on the Tsunami printer.

Artwork

- The art for the Beta Release was created in Pasadena, and is not in any files here.
- The file {ERIS}<Dc>LoopsBeta>BinderCover.tedit contains the cover page for the binder.
- The file {ERIS}<Doc>LoopsBeta>CommentForm.tedit contains the comment form for the entire documentation set. The prepaid mailer (see department file) must be added to complete this form.

WRITER'S NOTES

- The details for packaging are in the following files:
 {ERIS}<Doc>loops>ProductionSpecs>onfiguration-request.tedit
 {ERIS}<Doc>loops>ProductionSpecs>Packaging.tedit



**Replace this page with
Table of Contents
tab**

**Replace this page with
1. Introduction
tab**

**Replace this page with
2. Instances
tab**

**Replace this page with
3. Classes
tab**

**Replace this page with
4. Metaclasses
tab**

**Replace this page with
5. Accessing Data
tab**

**Replace this page with
6. Methods
tab**

**Replace this page with
7. Message Sending Forms
tab**

**Replace this page with
8. Active Values
tab**

**Replace this page with
9. Data Type Predicates
tab**

**Replace this page with
10. Browsers
tab**

**Replace this page with
11. Errors and Breaks
tab**

**Replace this page with
12. Breaking and Tracing
tab**

**Replace this page with
13. Editing
tab**

**Replace this page with
14. File Package
tab**

**Replace this page with
15. Performance Issues
tab**

**Replace this page with
16. Processes
tab**

**Replace this page with
17. Reading and Printing
tab**

**Replace this page with
18. User Input/Output Modules
tab**

**Replace this page with
19. Windows
tab**

**Replace this page with
20. System Variables and
Functions
tab**

Replace this page with
A. Active Values in Buttress
LOOPS
tab

**Replace this page with
Glossary
tab**

**Replace this page with
Index
tab**

abstract class	A class which cannot be instantiated, for example, ActiveValue .
active value	The mechanism that carries out access-oriented programming for variables in Xerox LOOPS. Active values send messages as a side effect of having an object's variable referenced.
activeValue	The previous implementation of the active value concept.
ActiveValue	An abstract class that defines the general protocol followed by all active value objects.
annotatedValue	A special Interlisp-D data type that wraps each ActiveValue instance.
AnnotatedValue	An abstract class that allows an annotatedValue to be treated as an object.
browser	A window that allows you to examine and change items in a data structure.
class	A description of one or more similar objects; that is, objects containing the same types of data fields and responding to the same messages.
class inheritance	The means by which a class inherits variables, values, and methods from its super class(es).
class lattice	A network showing the inheritance relationship among classes.
class variable (CV)	A variable that contains information shared by all instances of the class. A class variable is typically used for information about a class taken as a whole.
inheritance	The means by which you can organize information in objects, create objects that are similar to other objects, and update objects in a simplified way.
Inspector	A Xerox Lisp display program that has been modified to allow you to view classes, objects, and active values.
instance	An object described by a particular class. Every object within Xerox LOOPS is an instance of exactly one class.
instance variable (IV)	A variable that contains information specific to an instance.
instantiate	To make a new instance of a class.
lattice	An arrangement of nodes in a hierarchical network, which allows for multiple parents of each node.
Masterscope	A Xerox Lisp Library Module program analysis tool that has been modified to allow analysis of Xerox LOOPS files.
message	A command sent to an object that activates a method defined in the object's class. The object responds by computing a value that is returned to the sender of the message.
metaclass	Classes whose instances are classes or abstract classes.
method	What an object applies to the arguments of a message it receives. This is similar to a procedure in procedure-oriented programming, except that here, you determine the message to send and the object receiving the message.

	determines the method to apply, instead of the calling routine determining which procedure to apply.
mixin	A class that is used in conjunction with another class to create a subclass. Mixins never have instances, and hence have AbstractClass as their metaclass.
object	A data structure that contains data and a pointer to functionality that can manipulate the data.
property list	A place for storing additional information on classes, their variables, and their methods.
selector	Part of a message that is sent to an object. The object uses the selector to determine which method is appropriate to apply to the message arguments.
self	A method argument that represents the receiver of the message.
specialization	The process of creating a subclass from a class, or the result of that process.
subclass	A class that is a specialization of another class.
super class	A class from which a given class inherits variables, values, and methods.
Tofu	An acronym for Top of the universe, which is the highest class in the Xerox LOOPS hierarchy.
Unique Identifier (UID)	An alphanumeric identifier that Xerox LOOPS uses to store and retrieve objects. Objects do not have UIDs unless they are named, are instances of indexed objects, or are instances printed to a file.
wrap	Objects have fields that can contain data. Some ActiveValue can be added so this data is stored within it. When this occurs, the ActiveValue wraps the data.



Venue

LOOPS Release Notes

November, 1991

Address comments to:
Venue
User Documentation
1549 Industrial Road
San Carlos, CA 94070
415-508-9672

LOOPS RELEASE NOTES

November 1991

Copyright © 1988, 1991 by Venue.

All rights reserved.

LOOPS is a trademark of Venue.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

The information in this document is subject to change without notice and should not be construed as a commitment by Venue. While every effort has been made to ensure the accuracy of this document, Venue assumes no responsibility for any errors that may appear.

Text was written and produced with Venue text formatting tools; Xerox printers were used to produce text masters.

1. **RELEASE OVERVIEW**

LOOPS is a multiple-paradigm programming system designed to be used in a variety of artificial intelligence applications. LOOPS requires the Medley Release of Lisp.

1.1 System Configuration

You need the following minimum configuration to run Xerox LOOPS:

1186 Workstation	1108 Workstation
40 megabyte disk drive	40-42 megabyte disk drive
2 megabyte memory	2 megabyte memory

1.2 What's Included in This Release

Xerox LOOPS includes the following items:

- Software on floppy disks.

See Chapter 2, Changes from Koto LOOPS, for how the Lyric/Medley Release of LOOPS is different from the Koto Release.

- Documentation in binders.

See Chapter 3, Release Documentation, for details of release documentation.

[This page intentionally left blank]

2. CHANGES FROM KOTO LOOPS

This chapter describes how the current release of LOOPS, Lyric/Medley LOOPS, differs from the Koto Release. With the addition of a patch file, the current Loops release will also run under the Medley release. The inclusion of the patch file is automatically handled by the LOOPS installation process. To convert Koto LOOPS files into Lyric/Medley LOOPS files, see the LOOPS Users' Modules CONVERT-LOOPS-FILES.

A number of significant fixes to both the documentation and software have been made. In addition, LOOPS behaves differently due to new features of Lisp found beginning with the Lyric release:

- LOOPS vs. Packages

Lyric has Common Lisp functionality, including packages and new case-insensitive readtables. All LOOPS symbols are in the INTERLISP: or IL: package, and LOOPS is case-sensitive, so it is much easier to type LOOPS expressions into an Interlisp Exec (see the *Lisp Release Notes, Lyric Release*, for more on different Exec types).

- Editing

Lyric has a new default structure editor, SEdit. DEdit is still available as a Lyric Library Module, but most people find SEdit to be much faster and easier to use. SEdit has no specific LOOPS support features yet, but the features it has support LOOPS rather well. As an example, all Lyric LOOPS development was done using SEdit. See the *Lisp Release Notes, Lyric and Medley Releases*, for more information on SEdit.

- Source File Management

LOOPS source forms are no longer LAMBDATRAN forms; instead they use the definer system which also handles Common Lisp forms in Lyric and Medley (see the *Common Lisp Implementation Notes* for more on definers). This has several consequences:

- The source file format for Lyric/Medley LOOPS is different. A conversion utility is provided which takes Koto LOOPS source files and converts them to Lyric/Medley LOOPS format. The converter will not work correctly on source files from the prototype Buttress version of LOOPS. These files must first be converted using Koto LOOPS.
- LOOPS source forms are no longer of filetype FNS; they are instead METHOD-FNS.
- Comments may appear anywhere in a definer form, since the definer system removes them before making the form executable. This also means that various comments that the LOOPS system uses as help information (doc properties and first comments in method code) now need to be strings rather than comments, so the definer system will not remove them. The conversion utility handles this for most important comments; new code should use strings in these places.
- Method bodies can now be CL:LAMBDA or IL:LAMBDA, by using the **Method** function and specifying a keyword when creating the method. A

LOOPS CL:LAMBDA method can have Common Lisp features like &rest and :keywords.

- Masterscope

Masterscope is now a LOOPS Library Module, to match Masterscope becoming a Lisp Library Module in the Lyric release of XAIE, primarily because Masterscope does not support Common Lisp yet. Masterscope still supports LOOPS, however, and many bugs in that support from Buttress and Koto LOOPS have been fixed.

When using LOOPS Masterscope, quote all Method Names that you use. The Masterscope parser currently will not recognize Method Names unless they are quoted.

- ICONW

ICONW is now a part of Lisp and no longer needs to be loaded to run LOOPS.

- New Compiler

Lyric/Medley LOOPS uses the new Lyric compiler, which handles Common Lisp. This has several consequences:

- **_Super** and the other similar functions are now lexically scoped; that is, it is now illegal to call **_Super** anywhere but within a method body, and any selector given must be the same as the selector for that method.

- Files compiled by the new compiler have no FILECOMS. Use

```
(LOADFROM <FILE> NIL 'ALLPROP)
```

to load Source files so that LOOPS browsers can find them.

- The .DFASL output of the new compiler loads much faster than .DCOMs or the .LCOMs of Lyric.

The ByteCompiler is no longer supported for compilation of LOOPS files. With the new compiler and its macrolet facilities, a cleanup of LOOPS files requires that *DEFAULT-CLEANUP-COMPILER* be set to 'CL:COMPILE-FILE. The *Lisp Release Notes*, *Lyric Release*, contain more information on the new compiler and the cleanup flag. The ByteCompiler is no longer supported for compilation of LOOPS files.

- LOOPS Library Modules

SSDigiMeter will be removed from Gauges in the next release of LOOPS. DigiMeters are inherently self scaling because Meters are. Therefore, **SSDigiMeter** is redundant. Note that SSDigiMeters are also generally slower than DigiMeters.

[This page intentionally left blank]

Overview of the Manual

These Release Notes describe the Lyric/Medley Release of software for Xerox's Lisp Object-Oriented Programming System, LOOPS (TM). This document is directed to the people responsible for installing and testing LOOPS.

This manual describes the Lyric/Medley Release of LOOPS, which runs under the Lyric and Medley (with a small patch) Releases of Lisp.

Organization of the Manual and How to Use It

These Release Notes contain important information about the Lyric/Medley Release of LOOPS. The following chapters outline the major features of LOOPS, and highlight the principal differences between this and previous versions of LOOPS.

All readers should carefully read Chapter 1, Release Overview, Chapter 5, Reporting Procedure, and Chapter 6, Known Problems. Reading Chapter 2, Changes from Koto LOOPS, and Chapter 3, Release Documentation, is recommended for all readers. People responsible for installing LOOPS should read Chapter 4, Installation Procedures.

Conventions

These Release Notes use the following conventions:

- Case is significant in LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown.
- Arguments appear in italic type.
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a method appears as follows:

(← *self* **Selector** *Arg1 Arg2*)

- Examples appear in the following typeface:

89← (←LOGIN)

- All examples are typed into an Interlisp Exec. This is the recommended Exec for all LOOPS expressions.
- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

References

The following books and manuals augment this manual.

LOOPS Reference Manual

LOOPS Library Modules Manual

LOOPS Users' Modules Manual

Interlisp-D Reference Manual

Common Lisp: the Language by Guy Steele

Common Lisp Implementation Notes, Lyric Release

Lisp Release Notes, Lyric and Medley Releases

Lisp Library Modules Manual, Lyric and Medley Releases

XEROX LOOPS RELEASE NOTES

XEROX

XEROX LOOPS RELEASE NOTES

Lyric/Medley Release

July 1988

Copyright © 1988 by Xerox Corporation.

Xerox LOOPS is a trademark.

All rights reserved.

TABLE OF CONTENTS

PREFACE	v
1. RELEASE OVERVIEW	1
1.1 System Configuration	1
1.2 What's Included in This Release	1
2. CHANGES FROM KOTO LOOPS	3
3. RELEASE DOCUMENTATION	5
3.1 Xerox LOOPS Reference Manual	5
3.2 Xerox LOOPS Library Modules Manual	5
3.3 Xerox LOOPS Users' Modules Manual	5
4. INSTALLATION PROCEDURES	7
4.1 Overview of the Distribution Kits	7
4.1.1 1186 Distribution Kit	7
4.1.2 1108 Distribution Kit	7
4.2 Preparation	8
4.3 Installation	8
5. REPORTING PROCEDURE	11

TABLE OF CONTENTS

6. KNOWN PROBLEMS	13
6.1 Installation	13
6.2 Instances	14
6.3 Classes	14
6.4 MetaClasses	14
6.5 Active Values	14
6.6 Browsers	15
6.7 Breaking and Tracing	15
6.8 File Manager	16
6.9 Masterscope	16
6.10 Extensions to ?=	16
6.11 Windows	16
6.12 System Variables and Functions	17
6.13 Xerox LOOPS Library Modules	17
6.14 Conversion to Newer Releases	17

[This page intentionally left blank]

Overview of the Manual

These Release Notes describe the Lyric/Medley Release of software for Xerox's Lisp Object-Oriented Programming System, Xerox LOOPS (TM). This document is directed to the people responsible for installing and testing Xerox LOOPS.

This manual describes the Lyric/Medley Release of Xerox LOOPS, which runs under the Lyric and Medley (with a small patch) Releases of Xerox Lisp.

Organization of the Manual and How to Use It

These Release Notes contain important information about the Lyric/Medley Release of Xerox LOOPS. The following chapters outline the major features of Xerox LOOPS, and highlight the principal differences between this and previous versions of LOOPS.

All readers should carefully read Chapter 1, Release Overview, Chapter 5, Reporting Procedure, and Chapter 6, Known Problems. Reading Chapter 2, Changes from Koto LOOPS, and Chapter 3, Release Documentation, is recommended for all readers. People responsible for installing Xerox LOOPS should read Chapter 4, Installation Procedures.

Conventions

These Release Notes use the following conventions:

- Case is significant in Xerox LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown.
- Arguments appear in *italic type*.
- Selectors, methods, functions, objects, classes, and instances appear in **bold type**.

For example, a method appears as follows:

`(_ self Selector Arg1 Arg2)`

- Examples appear in the following typeface:

`89_ ( _ LOGIN)`

- All examples are typed into an Interlisp Exec. This is the recommended Exec for all Xerox LOOPS expressions.
- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

References

The following books and manuals augment this manual.

Xerox LOOPS Reference Manual

Xerox LOOPS Library Modules Manual

Xerox LOOPS Users' Modules Manual

Interlisp-D Reference Manual

Common Lisp: the Language by Guy Steele

Xerox Common Lisp Implementation Notes, Lyric Release

Xerox Lisp Release Notes, Lyric and Medley Releases

Xerox Lisp Library Modules Manual, Lyric and Medley Releases

1. **RELEASE OVERVIEW**

Xerox LOOPS is a multiple-paradigm programming system designed to be used in a variety of artificial intelligence applications. Xerox LOOPS runs on the Xerox 1100 series of Artificial Intelligence Workstations and requires the Lyric or Medley Release of Lisp running on the Xerox Artificial Intelligence Environment (XAIE).

1.1 System Configuration

You need the following minimum configuration to run Xerox LOOPS:

1186 Workstation	1108 Workstation
40 megabyte disk drive	40-42 megabyte disk drive
2 megabyte memory	2 megabyte memory

1.2 What's Included in This Release

Xerox LOOPS includes the following items:

- Software on floppy disks.

See Chapter 2, Changes from Koto LOOPS, for how the Lyric/Medley Release of Xerox LOOPS is different from the Koto Release.

- Documentation in binders.

See Chapter 3, Release Documentation, for details of release documentation.

[This page intentionally left blank]

2. CHANGES FROM KOTO LOOPS

This chapter describes how the current release of Xerox LOOPS, Lyric/Medley LOOPS, differs from the Koto Release. The current Xerox LOOPS release runs under the Lyric release of Xerox Lisp. With the addition of a patch file, the current Xerox Loops release will also run under the Xerox Lisp Medley release. The inclusion of the patch file is automatically handled by the LOOPS installation process. To convert Koto LOOPS files into Lyric/Medley LOOPS files, see the Xerox LOOPS Users' Modules CONVERT-LOOPS-FILES.

A number of significant fixes to both the documentation and software have been made. In addition, Xerox LOOPS behaves differently due to new features of Xerox Lisp found beginning with the Lyric release:

- Xerox LOOPS vs. Packages

Lyric has Common Lisp functionality, including packages and new case-insensitive readtables. All Xerox LOOPS symbols are in the INTERLISP: or IL: package, and Xerox LOOPS is case-sensitive, so it is much easier to type Xerox LOOPS expressions into an Interlisp Exec (see the *Xerox Lisp Release Notes, Lyric Release*, for more on different Exec types).

- Editing

Lyric has a new default structure editor, SEdit. DEdit is still available as a Lyric Library Module, but most people find SEdit to be much faster and easier to use. SEdit has no specific Xerox LOOPS support features yet, but the features it has support Xerox LOOPS rather well. As an example, all Lyric LOOPS development was done using SEdit. See the *Xerox Lisp Release Notes, Lyric and Medley Releases*, for more information on SEdit.

- Source File Management

Xerox LOOPS source forms are no longer LAMBDATRAN forms; instead they use the definer system which also handles Common Lisp forms in Lyric and Medley (see the *Xerox Common Lisp Implementation Notes* for more on definers). This has several consequences:

- The source file format for Lyric/Medley LOOPS is different. A conversion utility is provided which takes Koto LOOPS source files and converts them to Lyric/Medley LOOPS format. The converter will not work correctly on source files from the prototype Buttress version of LOOPS. These files must first be converted using Koto LOOPS.
- Xerox LOOPS source forms are no longer of filetype FNS; they are instead METHOD-FNS.
- Comments may appear anywhere in a definer form, since the definer system removes them before making the form executable. This also means that various comments that the Xerox LOOPS system uses as help information (doc properties and first comments in method code) now need to be strings rather than comments, so the definer system will not remove them. The conversion utility handles this for most important comments; new code should use strings in these places.
- Method bodies can now be CL:LAMBDA's or IL:LAMBDA's, by using the **Method** function and specifying a keyword when creating the method. A

Xerox LOOPS CL:LAMBDA method can have Common Lisp features like &rest and :keywords.

- Masterscope

Masterscope is now a Xerox LOOPS Library Module, to match Masterscope becoming a Xerox Lisp Library Module in the Lyric release of XAIE, primarily because Masterscope does not support Common Lisp yet. Masterscope still supports Xerox LOOPS, however, and many bugs in that support from Buttress and Koto LOOPS have been fixed.

When using LOOPS Masterscope, quote all Method Names that you use. The Masterscope parser currently will not recognize Method Names unless they are quoted.

- ICONW

ICONW is now a part of Xerox Lisp and no longer needs to be loaded to run LOOPS.

- New Compiler

Lyric/Medley LOOPS uses the new Lyric compiler, which handles Common Lisp. This has several consequences:

- **_Super** and the other similar functions are now lexically scoped; that is, it is now illegal to call **_Super** anywhere but within a method body, and any selector given must be the same as the selector for that method.

- Files compiled by the new compiler have no FILECOMS. Use

```
(LOADFROM <FILE> NIL 'ALLPROP)
```

to load Source files so that Xerox LOOPS browsers can find them.

- The .DFASL output of the new compiler loads much faster than .DCOMS or the .LCOMS of Lyric.

The ByteCompiler is no longer supported for compilation of LOOPS files. With the new compiler and its macrolet facilities, a cleanup of LOOPS files requires that *DEFAULT-CLEANUP-COMPILER* be set to 'CL:COMPILE-FILE. The *Xerox Lisp Release Notes*, *Lyric Release*, contain more information on the new compiler and the cleanup flag. The ByteCompiler is no longer supported for compilation of LOOPS files.

- LOOPS Library Modules

SSDigiMeter will be removed from Gauges in the next release of LOOPS. DigiMeters are inherently self scaling because Meters are. Therefore, **SSDigiMeter** is redundant. Note that SSDigiMeters are also generally slower than DigiMeters.

[This page intentionally left blank]

3. RELEASE DOCUMENTATION

This chapter describes the documentation that is part of Xerox LOOPS.

3.1 Xerox LOOPS Reference Manual

This manual provides a detailed description of all the methods, functions, classes, and other items available in Xerox LOOPS. This manual is for people who are already familiar with Xerox LOOPS programming principles.

3.2 Xerox LOOPS Library Modules Manual

This manual describes the Xerox LOOPS Library Modules: Gauges, Masterscope, and Virtual Copies. This manual is for people who want to use the additional features that these Library Modules provide.

3.3 Xerox LOOPS Users' Modules Manual

This manual describes the various Xerox LOOPS Users' Modules. This manual is for people who want to use the additional features that these Users' Modules provide.

[This page intentionally left blank]

4. INSTALLATION PROCEDURES

This chapter describes how to install the Xerox LOOPS program files, Library Modules program files, and Users' Modules program files on both the 1186 and 1108 workstations. The installation procedure for Xerox LOOPS is the same for both the Lyric and Medley releases.

4.1 Overview of the Distribution Kits

The distribution kits for Xerox LOOPS are different for the 1186 and the 1108.

4.1.1 1186 Distribution Kit

The distribution kit for Xerox LOOPS on the 1186 consists of four 5-1/4" diskettes:

- Lyric/Medley LOOPS System #1, which contains Xerox LOOPS program files.
- Lyric/Medley LOOPS System #2, which contains Xerox LOOPS program files.
- Lyric/Medley LOOPS Library, which contains the program files for the Xerox LOOPS Library Modules.
- Lyric/Medley LOOPS Users, which contains the program files for Xerox LOOPS Users' Modules.

4.1.2 1108 Distribution Kit

The distribution kit for Xerox LOOPS on the 1108 consists of three 8" diskettes:

- Lyric/Medley LOOPS System, which contains Xerox LOOPS program files.
- Lyric/Medley LOOPS Library, which contains the program files for the Xerox LOOPS Library Modules.
- Lyric/Medley LOOPS Users, which contains the program files for Xerox LOOPS Users' Modules.

4.2 Preparation

Before installing Xerox LOOPS, make sure that you have performed the following steps:

- Load the Lyric or Medley version of Xerox Lisp. Have your Lyric or Medley Release Kit handy if you are not on a network, as you may need to load parts of Lyric or Medley Xerox Lisp not previously loaded.

A standard partition of 8 MB is acceptable. A LOOPS sysout with an average number of LISP Library and LOOPS Users' Modules will require about 11,000 pages or 5 MB.

- Boot Xerox Lisp
- Open an Interlisp Executive Window.
- Make certain the time is set.

Machines on a Xerox network automatically get the current time of day from the net when they boot. If you are not on a network, make certain that the function (DATE) returns the current date and time. If it does not, then use the function (SETTIME) to correct it.

CAUTION

Rebooting a VMEM.PURE.STATE sysout without an Ethernet connection and without the time being set will erroneously allow you to create objects without informing you that the time is not set.

4.3 Installation

The Lyric/Medley LOOPS installation tool makes the installation of Xerox LOOPS, its Library Modules, and Users' Modules almost identical for 1186 and 1108 workstations. The differences are in the names of floppies. 1186 floppies are smaller, so there are more of them to hold the same data. The installation tool will determine what workstation you are using and prompt you with the appropriate floppy names.

1. Have your Lyric/Medley Release Kit handy, or, if you are connected to a network, set the **DIRECTORIES** and **DISPLAYFONTDIRECTORIES** variables appropriately so the sysout can find your Lyric library and font files.

2. Make the floppy drive your connected directory:

```
CONN {FLOPPY}
```

```
DIR {FLOPPY}
```

3. If you are using an 1108 workstation, insert the floppy labeled *Lyric LOOPS System*. If you are using an 1186 workstation, insert the floppy labeled *Lyric LOOPS System #1*. Enter the following into your Exec:

```
LOAD (LOOPS)
```


A menu will appear that looks like this:

```

Loops System
Install from floppies
Load into sysout

```

4. Select the menu option **Install from floppies**.

The following menu will appear:

Loops directories	Click here when done
LOOPSDIRECTORY	{DSK}<LISPFILS>LOOPS>
LOOPSLIBRARYDIRECTORY	{DSK}<LISPFILS>LOOPS>LIBRARY>
LOOPUSERSDIRECTORY	{DSK}<LISPFILS>LOOPS>USERS>
LOOPUSERSRULESDIRECTORY	{DSK}<LISPFILS>LOOPS>RULES>

This menu shows the current (or default, if unset) values of the variables LOOPS examines when it loads things.

- If you want LOOPS installed on your local disk under {DSK}<LISPFILS>LOOPS>, just select **Click here when done**. LOOPS requires approximately 2200 pages on the local disk.
- If you want LOOPS installed somewhere else, select the directory names using the left mouse button. Change the directory names by backspacing over them and typing new locations; select **Click here when done** when you are finished.

5. The installation tool prompts you for floppies with this menu:

```

Please insert floppy Lyric LOOPS System
Click here when done

```

It copies the LOOPS files from the floppies to the directories you just specified in the menu.

When the installation tool asks you for a floppy, insert the floppy in the drive, then select **Click here when done**.

As the last installation step, the installation tool modifies the file LOOPS, writes it out to LOOPSDIRECTORY, and compiles it.

When installation is finished, the first menu reappears:

```

Loops System
Install from floppies
Load into sysout

```

6. Select the menu option **Load into sysout** to load LOOPS into your system. The following menu appears:

```

Load Which?
Loops
Loops Masterscope
Gauges
LoopsBackwards
VirtualCopy

```

7. Select **LOOPS** from the menu to load LOOPS from the location where you installed it.

Once LOOPS is loaded the LOOPS System menu reappears. To load one of the other LOOPS library or User modules, select the appropriate name in the Load Which? menu.

8. Position your mouse cursor anywhere on the screen except for the Load Which? menu, then press the left mouse button to exit the installation procedure.

To load LOOPS in the future, after the installation procedure is finished, load the LOOPS.DFASL file which the installation procedure created:

```
(CNDIR LOOPS DIRECTORY)
```

```
(LOAD 'LOOPS.DFASL)
```

This will load the rest of LOOPS.

When you perform a CNDIR, LOOPS DIRECTORY in the example above is the directory you defined in Step 4; e.g.,

```
(CNDIR ' {DSK}<LISPFILES>LOOPS>)
```

LISPUSERSDIRECTORIES should point to a directory containing GRAPHER.LCOM, and DISPLAYFONTDIRECTORIES should point to a directory containing the Helvetica display font files from your Lyric or Medley XAIE distribution floppies.

CAUTION

LOOPS uses the new XAIE compiler and its macrolet facilities. When LOOPS is loaded, it sets your ***DEFAULT-CLEANUP-COMPILER*** to **'CL:COMPILE-FILE**. More information on this cleanup flag and the new compiler is available in the *Xerox Lisp Release Notes, Lyric Release*.

[This page intentionally left blank]

5. REPORTING PROCEDURE

If you have problems with this release, file an Action Request (AR) Bug Report.

[This page intentionally left blank]

6. KNOWN PROBLEMS

This chapter is a compilation of known problems in the Lyric/Medley Release of Xerox LOOPS. These problems are in the form of Action Requests (ARs) from the Xerox AR data base, followed by a brief description.

The ARs are divided into the following categories:

- Installation
- Instances
- Classes
- MetaClasses
- Active values
- Browsers
- Breaking and tracing
- File Manager
- Masterscope
- Extensions to ?=
- Windows
- System Variables and Functions
- Xerox LOOPS Library Packages
- Conversion to newer releases

6.1 Installation

AR	Description
10169	During the installation process, if fonts need to be loaded the LOOPS installer will try to write them to the directory on the front (CAR) of the list DISPLAYFONTDIRECTORIES. Workaround: Be sure that this is a directory you can write to, e.g. by pushing the name of a local directory onto the list like "{dsk}<lispfiles>fonts>." Grapher is loaded into the current sysout without being copied to a local directory; you may wish to do this and put that directory on your DIRECTORIES list.

6.2 Instances

AR	Description
9225	The implementation of multiple names for instances has NOT been changed in the Lyric/Medley release of LOOPS. However, some conditions which previously caused instances to appear twice on a file have been fixed.

6.3 Classes

AR	Description
8881	The Supers MACRO is not documented. It retrieves the supers of the given class. It accepts a class object as an argument and returns a list of class objects.
9134	Caution: ListAttribute (page 3-27) finds non-local IVPROPS on classes.
9881	To use the method CVMissing, it is necessary to define a class called MyClass which is a specialization of the class Class. Then the CVMissing method is specialized at this level (in MyClass). Finally, a class Foo is defined whose metaclass is MyClass. At this point the expression <pre>(GetClassValue (_ (\$ Foo) New) 'NewCV)</pre> can be evaluated and a new class variable will be created.
9884	To create a class / instance variable without a value the variable should be set to the value of NotSetValue .

6.4 MetaClasses

AR	Description
10049	A new low level accessor function has been defined for programmers who wish to implement their own inheritance schemes. (FetchMethodLocally classobj selector) If <i>classobj</i> has a method for <i>selector</i> returns its name, otherwise NIL.

6.5 Active Values

AR	Description
9158	If an IndirectVariable points at an ActiveValue in a remote object then the remote active value's Get or PutWrappedValue method is

triggered with the **containingObj** argument holding the object containing the **IndirectVariable**.

6.6 Browsers

AR	Description
9244	The behavior of LatticeBrowser and ClassBrowser are different for AddRoot. Workaround: The example for AddRoot in section 10.5.3 should be changed to: The following creates an instance of LatticeBrowser and adds Tofu as a root: <pre>54_ (_ (\$ LatticeBrowser) New 'LB1) 55_ (_ (\$ LB1) AddRoot 'Tofu)</pre>
9851	Section 10.3.2.7 "Extending Functionality with the Left Mouse Button:" should be modified as follows. On the 1186 the COPY key is similar to SHIFT, the MOVE and CTRL keys behave similarly, and META and SAME are similar as well.
9859	The specialization of LatticeBrowser methods into ClassBrowser methods is not enumerated. In section 10.5 "Programmer's Interface to Lattice Browsers," some methods are described which take different arguments when invoked on Class Browsers, e.g., BoxNode sent to a ClassBrowser does not allow the unboxPrevious flag.
10010	The documentation for NewItem in section 10.5.3 has the following additional information: Purpose: Gets an object, prompting the user if necessary. Behavior: ...to be added. NewItem accepts only names and maps them to objects using \$!. If no name is entered at the prompt (by pressing return), NewItem returns NIL.
10367	When specializing a method from the class browser, if there are no methods to inherit, other than generic methods from Object, the menu lists methods already defined in the current class. Be careful about specializing methods defined directly under Object.

6.7 Breaking and Tracing

AR	Description
10356	BT will not show the send frames for broken methods. Workaround: You can do BT! and grab the method name from the *APPLY* frame. Inspect the METHOD-FNS definition of it.

6.8 File Manager

AR	Description
9159	Renaming a method does not smash the associated old symbol's function definition.
9166	Renaming a method does not remove the associated old method-fns definition from the file manager's change list.
10484	CLASS coms compiled by Lyric/Medley LOOPS are combined into a single large form that is read all at once. This causes occasional "Class x not defined, defining one now." messages to appear. Aside from the inconvenience of not seeing the names of classes during compilation, this should not cause problems.

6.9 MasterScope

AR	Description
9911	When LOOPS MasterScope analyzes forms like <pre>(LET* ((A B) (C (F A))) --)</pre> dwim asks about what it thinks is an unbound use of A. The analysis is correct, however.

6.10 Extensions to ?=

AR	Description
9828	? = documentation for the ClassBrowser methods is not available.

6.11 Windows

AR	Description
9243	PutSavedValue and SavedValue have not been removed from this release of Xerox LOOPS. We no longer plan to remove them.
10040	There are two new methods on the Window class: <pre>(_ window Open?)</pre> returns non-NIL if window is a LOOPS Window and is open. <pre>(_ window Shade shade)</pre> shades a LOOPS Window if it's open. The <i>shade</i> argument defaults to the value of GRAYSHADE1 .

6.12 System Variables and Functions

AR	Description
9222	The variable LispUserFilesForLoops actually names Lisp Library modules.

6.13 Xerox LOOPS Library Modules

AR	Description
9797	After VirtualCopyMixin classes are "destroyed," instances of them can still be created without an error occurring.
9868	If a gauge is Attach 'ed to one value, and then Attach 'ed to another value the title of the gauge will not change even though the gauge will display the new value.
9871	Sending a Shape message to a Meter with the ExtraSpaceFlag set to T has no effect.

6.14 Conversion to Newer Releases

AR	Description
10404	There is currently no really convenient way of converting instance files from Buttriss into Koto and Lyric/Medley. However, one can load LOOPSBACKWARDS , which modifies the OLD-INTERLISP-FILE readtable to handle the older style instances. Then one can read instances individually from the file and write them into a new file. In the new file they will be correctly formatted in the new manner.

[This page intentionally left blank]

INDEX

A

Action Requests 13
active values, known problems with 14

B

breaking and tracing, known problems with 15
browsers, known problems with 15

C

classes, known problems with 14
conversion to newer releases, known problems with 17

D

distribution kits 7
documentation 5

E

Extensions to ?=, known problems with 16

F

file manager, known problems with 16

I

ICONW 4
installation
 preparation 8
installation, known problems with 13
instances, known problems with 14
items in release 1

K

known problems 13

L

LOOPS Library Modules 4
Lyric compiler, new 4

M

Masterscope 4
 known problems with 16
Medley patch file 3
metaclasses, known problems with 14

P

Packages 3
problem reporting procedure 11

R

release documentation 5
reporting procedure 11

S

SEdit 3
source file management 3
system configuration 1
system variables and functions, known problems with 17

W

Windows, known problems with 16

X

Xerox LOOPS Library Modules, known problems with 17
Xerox LOOPS Reference Manual 5

1

1108 distribution kit 7
1186 distribution kit 7

ntionally left blank]

[This page inte

This appendix describes how to install the LOOPS System files, Library Modules files, and Users' Modules files on the Sun Workstation.

Overview of the Distribution Kit

The distribution kit for LOOPS on the Sun consists of a single ¼-inch tape cartridge. It contains the complete release in "tar" format and creates appropriate directories when its contents are extracted.

Preparation

Preparing to install LOOPS requires that the Medley release of Lisp is already installed and that adequate file space is available.

Before installing LOOPS, remember that

- the Medley 1.0-S release of Lisp must already be installed on your Sun Workstation;
- the complete LOOPS distribution requires about 1.2 MBytes of file space.

Installation

The software installation procedure shows the steps required for installing the Lyric/Medley LOOPS software on a Sun Workstation with Medley 1.0-S already installed. Examples are given where appropriate. Only those users who are system administrators and have **root** privileges can install the LOOPS, Lyric/Medley release.

Before starting software installation, remember that the LOOPS software requires about 1.2 MBytes of file space.

1. Log in under your username.

```
login yourname
```

```
prompt%
```

where **yourname** is replaced by your username.

2. Check for available space with the **df** command:

```
prompt% df
```

Filesystem	kbytes	used	avail	capacity	Mounted on
/dev/xy0a	7437	5470	1223	82%	/
/dev/xy0h	148455	4900	128709	96%	/usr/misc

3. Determine if you need to run **su** to make a directory for the distribution. If so, type in **su**:

```
prompt% su
```

4. Make a directory for the distribution. This directory should be named **/usr/local/lde/loops**. If you have enough space on the file system containing **/usr/local/lde**, then

```
prompt# mkdir /usr/local/lde/loops/
```

If you don't have enough space on **/usr/local/lde**, go to step 6.

5. Make yourself owner of this directory:

```
prompt# /etc/chown yourname /usr/local/lde/loops/
```

where **yourname** is your username.

6. If you don't have space on the file system which contains **/usr/local/lde**, but do have space somewhere else, for instance on **/usr1**, then make the directory there and link **/usr/local/lde/loops** to it:

```
prompt# mkdir /usr1/loops
```

```
prompt# /etc/chown yourname /usr/usr1/loops
```

```
prompt# ln -s /usr1/loops /usr/local/lde/loops
```

7. If you ran **su**, leave the privileged shell by typing:

```
prompt% exit
```

8. Insert the ¼-inch cartridge tape, containing the LOOPS software, in the drive.

9. Connect to **/usr/local/lde/loops**:

```
prompt# cd /usr/local/lde/loops
```

10. Load the Medley software from tape. Indicate the appropriate device abbreviation for your tape by replacing **xx** in the example below with

ar for the Archive drive,

st for a SCSI tape drive.

This example shows the command entry sequence:

```
prompt# tar xvpf /dev/rxx0
```

As the software is loaded (a process that takes some time) the system prints a series of lines in the following form:

```
x ./system/LOOPS., 28552 bytes, 56 tape blocks
```

The x at the beginning of the line indicates that the file is being extracted from the tape.

This creates directories named:

```
/usr/local/lde/loops/system/
```

```
/usr/local/lde/loops/library/
```

```
/usr/local/lde/loops/users/
```

This is a good time to set the protection of the extracted directories and files so that the work group using LOOPS has at least read access to them.

11. Boot Medley Lisp.
12. Open an Interlisp Executive Window.
13. Make certain the time is set correctly.
14. Set the **DIRECTORIES** and **DISPLAYFONTDIRECTORIES** variables appropriately so the sysout can find your Lyric/Medley library and font files.
15. Make the LOOPS System directory your connected directory:

```
CONN {DSK}/usr/local/lde/loops/system/
```

16. Enter the following into your Exec:

```
LOAD (LOOPS)
```

A menu appears that looks like this:

```

Loops system
Install from distribution
Load into sysout

```

17. Select the menu option **Install from distribution**.

The following menu appears:

Loops directories	Click here when
LOOPSDIRECTORY	{DSK}<LISPFILS>LOOPS>
LOOPSLIBRARYDIRECTORY	{DSK}<LISPFILS>LOOPS>LIB
LOOPSUSERSDIRECTORY	{DSK}<LISPFILS>LOOPS>USE
LOOPSUSERSRULESDIRECTORY	{DSK}<LISPFILS>LOOPS>RUL

This menu shows the current (or default, if unset) values of the variables LOOPS examines when it loads things.

If you have installed LOOPS under /usr/local/lde/loops/ click the mouse on the menu items to set these directories to point where the tape was unloaded:

```

LOOPSDIRECTORY      {dsk}/usr/local/lde/loops/system/
LOOPSLIBRARYDIRECTORY {dsk}/usr/local/lde/loops/library/
LOOPSUSERSDIRECTORY  {dsk}/usr/local/lde/loops/users/
LOOPSUSERSRULESDIRECTORY {dsk}/usr/local/lde/loops/users/

```

As the last installation step, the installation tool automatically modifies the file LOOPSSITE, writes it out to **LOOPSDIRECTORY**, and compiles it.

When this setp is finished, the first menu reappears:


```
Loops system
Install from distribution
Load into sysout
```

18. Select the menu option **Load into sysout** to load LOOPS into your system. The following menu appears:

```
Load Which?
Loops
Loops Masterscope
Gauges
LoopsBackwards
VirtualCopy
```

19. Select **LOOPS** from the menu to load LOOPS from the location where you installed it.

Once LOOPS is loaded the LOOPS System menu reappears. To load one of the other LOOPS library or Users' modules, select the appropriate name in the Load Which? menu.

20. Position your mouse cursor anywhere on the screen except for the Load Which? menu, then press the left mouse button to exit the installation procedure.

Lyric/Medley LOOPS is now installed on your Sun Workstation.

Loading After Installation

This section describes how to reload LOOPS into a newly started Lisp sysout after LOOPS has been previously installed.

1. Start up Medley on your Sun Workstation.
2. Open an INTERLISP Exec window.
3. Make sure **DIRECTORIES** points to a directory containing GRAPHER.LCOM, and **DISPLAYFONTDIRECTORIES** points to a directory containing the Helvetica display font files from your Lisp distribution kit.

4. Connect to the directory containing the LOOPS system files:

```
(CNDIR ' {DSK} /USR/LOCAL/LDE/LOOPS/SYSTEM/)
```

5. Load LOOPS loader program:

```
(FILESLOAD LOADLOOPS)
```

6. Run the LOOPS loader program:

```
(LOADLOOPS)
```

This procedure loads only the LOOPS system files. Please see the manuals describing the LOOPS Library and Users' Modules for their loading procedures.

CAUTION

LOOPS uses the new compiler and its macrolet facilities. When LOOPS is loaded, it sets your ***DEFAULT-CLEANUP-COMPILER*** to **'CL:COMPILE-**

FILE. More information on this cleanup flag and the new compiler is available in the *Lisp Release Notes*, in your Lyric or Medley Lisp kit.

[This page intentionally left blank]

DOCUMENT UPDATE SHEET

Document Name: *LOOPS Manual*

Document Number: *310000*

DOC. VERSION	RELEASE DATE	REPLACE PAGES	INSERT PAGES	INSTRUCTIONS/ NOTES
Lyric/Medley	Oct., 1988	NA	NA	Please read the Errata Sheet, accompanying this release material, for last minute release notes.
Lyric/Medley	Oct., 1988	NA	NA	The Lyric/Medley LOOPS documentation contains numerous references to Xerox LOOPS. Xerox LOOPS is now known as Envos
LOOPS.				
Lyric/Medley	Oct., 1988	NA	A-1-A-4	Add <i>Appendix A, Sun Installation Procedure</i> , to your <i>LOOPS Release Notes</i> .



Venue

LOOPS

Reference Manual
Library Modules Manual
Users' Modules Manual

November, 1991

Address comments to:
Venue
User Documentation
1549 Industrial Road
San Carlos, CA 94070
415-508-9672

LOOPS

REFERENCE MANUAL
LIBRARY MODULES MANUAL
USERS' MODULES MANUAL

November, 1991

Copyright © 1988, 1991 by Venue.

All rights reserved.

LOOPS and Medley are trademarks of Venue.

UNIX® is a registered trademark of UNIX System Laboratories.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

The information in this document is subject to change without notice and should not be construed as a commitment by Venue. While every effort has been made to ensure the accuracy of this document, Venue assumes no responsibility for any errors that may appear.

Text was written and produced with Venue text formatting tools; Xerox printers were used to produce text masters. The typeface is Classic.



Venue

LOOPS Users' Modules Manual

November 1991

Address comments to:
Venue
User Documentation
1549 Industrial Road
San Carlos, CA 94070
415-508-9672

LOOPS USERS' MODULES MANUAL

November, 1991

Copyright © 1988, 1991 by Venue.

All rights reserved.

LOOPS is a trademark of Venue.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

The information in this document is subject to change without notice and should not be construed as a commitment by Venue. While every effort has been made to ensure the accuracy of this document, Venue assumes no responsibility for any errors that may appear.

Text was written and produced with Venue text formatting tools; Xerox printers were used to produce text masters. The typeface is Classic.

1. INTRODUCTION TO RULE-ORIENTED PROGRAMMING IN LOOPS

The core of decision-making expertise in many kinds of problem solving can be expressed succinctly in terms of rules. The following sections describe facilities in LOOPS for representing rules, and for organizing knowledge-based systems with rule-oriented programming. The LOOPS rule language provides an experimental framework for developing knowledge-based systems. The rule language and programming environment are integrated with the object-oriented, data-oriented, and procedure-oriented parts of LOOPS.

Rules in LOOPS are organized into production systems (called RuleSets) with specified control structures for selecting and executing the rules. The work space for RuleSets is an arbitrary LOOPS object.

Decision knowledge can be factored from control knowledge to enhance the perspicuity of rules. The rule language separates decision knowledge from meta-knowledge such as control information, rule descriptions, debugging instructions, and audit trail descriptions. An audit trail records inferential support in terms of the rules and data that were used. Such trails are important for knowledge-based systems that must be able to account for their results. They are also essential for guiding belief revision in programs that need to reason with incomplete information.

1.1 Introduction

Production rules have been used in expert systems to represent decision-making knowledge for many kinds of problem-solving. Such rules (also called *if-then* rules) specify actions to be taken when certain conditions are satisfied. Several rule languages have been developed in the past few years and used for building expert systems. The following sections describe the concepts and facilities for rule-oriented programming in LOOPS.

LOOPS has the following major features for rule-oriented programming:

- (1) Rules in LOOPS are organized into ordered sets of rules (called RuleSets) with specified control structures for selecting and executing the rules. Like subroutines, RuleSets are building blocks for organizing programs hierarchically.
- (2) The work space for rules in LOOPS is an arbitrary LOOPS object. The names of the instance variables provide a name space for variables in the rules.
- (3) Rule-oriented programming is integrated with object-oriented, data-oriented, and procedure-oriented programming in LOOPS.
- (4) RuleSets can be invoked in several ways: In the object-oriented paradigm, they can be invoked as methods by sending messages to objects. In the data-oriented paradigm, they can be invoked

as a side-effect of fetching or storing data in active values. They can also be invoked directly from Lisp programs. This integration makes it convenient to use the other paradigms to organize the interactions between RuleSets.

- (5) RuleSets can also be invoked from rules either as predicates on the LHS of rules, or as actions on the RHS of rules. This provides a way for RuleSets to control the execution of other RuleSets.
- (6) Rules can automatically leave an audit trail. An audit trail is a record of inferential support in terms of rules and data that were used. Such trails are important for programs that must be able to account for their results. They can also be used to guide belief revision in programs that must reason with incomplete information.
- (7) Decision knowledge can be separated from control knowledge to enhance the perspicuity of rules. The rule language separates decision knowledge from meta-knowledge such as control information, rule descriptions, debugging instructions, and audit trail descriptions.
- (8) The rule language provides a concise syntax for the most common operations.
- (9) There is a fast and efficient compiler for translating RuleSets into Interlisp functions.
- (10) LOOPS provides facilities for debugging rule-oriented programs.

The following sections are organized as follows: Section 1.2, "Basic Concepts," outlines the basic concepts of rule-oriented programming in LOOPS. It contains many examples that illustrate techniques of rule-oriented programming. Section 1.3, "Organizing a Rule-Oriented Program," describes the rule syntax, and the remaining sections in this chapter discuss the facilities for creating, editing, and debugging RuleSets in LOOPS.

1.2 Basic Concepts

Rules express the conditional execution of actions. They are important in programming because they can capture the core of decision-making for many kinds of problem-solving. Rule-oriented programming in LOOPS is intended for applications to expert and knowledge-based systems.

The following sections outline some of the main concepts of rule-oriented programming. LOOPS provides a special language for rules because of their central role, and because special facilities can be associated with rules that are impractical for procedural programming languages. For example, LOOPS can save specialized audit trails of rule execution. Audit trails are important in knowledge systems that need to explain their conclusions in terms of the knowledge used in solving a problem. This capability is essential in the development of large knowledge-intensive systems, where a long and sustained effort is required to create and validate knowledge bases. Audit trails are also important for programs that do non-monotonic reasoning. Such programs must work with incomplete information, and must be able to revise their conclusions in response to new information.

1.3 Organizing a Rule-Oriented Program

In any programming paradigm, it is important to have an organizational scheme for composing large systems from smaller ones. Stated differently, it is important to have a method for partitioning large programs into nearly-independent and manageably-sized pieces. In the procedure-oriented paradigm, programs are decomposed into procedures. In the object-oriented paradigm, programs are decomposed into objects. In the rule-oriented paradigm, programs are decomposed into *RuleSets*. A LOOPS program that uses more than one programming paradigm is factored across several of these dimensions.

There are three approaches to organizing the invocation of RuleSets in LOOPS:

Procedure-oriented Approach. This approach is analogous to the use of subroutines in procedure-oriented programming. Programs are decomposed into RuleSets that call each other and return values when they are finished. *SubRuleSets* can be invoked from multiple places. They are used to simplify the expression in rules of complex predicates, generators, and actions.

Object-oriented Approach. In this approach, RuleSets are installed as methods for objects. They are invoked as methods when messages are sent to the objects. The method RuleSets are viewed analogously to other procedures that implement object message protocols. The value computed by the RuleSet is returned as the value of the message sending operation.

Data-oriented Approach. In this approach, RuleSets are installed as access functions in active values. A RuleSet in an active value is invoked when a program gets or puts a value in the LOOPS object. As with active values with Lisp functions for the *getFn* or *putFn*, these RuleSet active values can be triggered by any LOOPS program, whether rule-oriented or not.

These approaches for organizing RuleSets can be combined to control the interactions between bodies of decision-making knowledge expressed in rules. For example, Figure 1 shows the RuleSet of consumer instructions for testing a washing machine. The work space for the ruleSet is a LOOPS object of the class **WashingMachine**. The control structure *While1* loops through the rules trying an escalating sequence of actions, starting again at the beginning of some rule is applied. Some rules, called one-shot rules, are executed at most once. These rules are indicated by preceding them with a one in braces ({1}).

```

RuleSet Name: CheckWashingMachine;
Workspace Class: WashingMachine;
Control Structure: while1;
While Condition: ruleApplied;

(* What a consumer should do when a washing machine failes.)

  IF .Operational THEN (STOP T);

  IF load>1.0 THEN .ReduceLoad;

  If ~pluggedInTo THEN .PlugIn;

{1} IF pluggedInTo:voltage=0 THEN breaker.Reset;
{1} IF pluggedInTo:voltage<110 THEN SPGE.Call;
{1} THEN dealer.RequestService;
{1} THEN manufacturer.Complain;
{1} THEN $ConsumerBoard.Complain;
{1} THEN (STOP T);

```

Figure 1. Basic RuleSet

1.4 Control Structures for Selecting Rules

RuleSets in LOOPS consist of an ordered list of rules and a control structure. Together with the contents of the rules and the data, a RuleSet control structure determines which rules are executed. Execution is determined by the contents of rules in that the conditions of a rule must be satisfied for it to be executed. Execution is also controlled by data in that different values in the data allow different rules to be satisfied. Criteria for iteration and rule selection are specified by a RuleSet control structure. There are two primitive control structures for RuleSets in LOOPS which operate as follows:

Do1	[RuleSet Control Structure]
The first rule in the RuleSet whose conditions are satisfied is executed. The value of the RuleSet is the value of the rule. If no rule is executed, the RuleSet returns NIL. The Do1 control structure is useful for specifying a set of mutually exclusive actions, since at most one rule in the RuleSet will be executed for a given invocation. When a RuleSet contains rules for specific and general situations, the specific rules should be placed before the general rules.	

DoAll

[RuleSet Control Structure]

Starting at the beginning of the RuleSet, every rule is executed whose conditions are satisfied. The value of the RuleSet is the value of the last rule executed. If no rule is executed, the RuleSet returns **NIL**.

The **DoAll** control structure is useful when a variable number of additive actions are to be carried out, depending on which conditions are satisfied. In a single invocation of the RuleSet, all of the applicable rules are invoked.

Figure 2 illustrates the use of a **Do1** control structure to select one of three mutually exclusive actions.

```
RuleSet Name: SimulateWashingMachine;
Workspace Class: WashingMachine;
Control Structure: Do1 ;

(* Rules for controlling the wash cycle of a washing machine.)

IF controlSetting = 'RegularFabric
THEN .Fill .Wash .Pause .SpinAndDrain
    .SprayAndRinse .SpinAndDrain
    .Fill. DeepRinse .Pause .DampDry;

IF controlSetting = 'PermanentPress
THEN .Fill .Wash .Pause .SpinAndPartialDrain
    .FillCold .SpinAndPartialDrain
    .FillCold .Pause .SpinAndDrain
    .FillCold. DeepRinse .Pause .DampDry;

IF controlSetting = 'DelicateFabric
THEN .FillSoak1 .Agitate .Soak4 .Agitate
    .Soak1 .SpinAndDrain .SprayAndRinse
    .SpinAndDrain .Fill .DeepRinse .Pause .DampDry;
```

Figure 2. RuleSet showing Do1

There are two control structures in LOOPS that specify iteration in the execution of a RuleSet. These control structures use an explicit while-condition associated with the RuleSet. They are direct extensions of the two primitive control structures above.

While1

[RuleSet Control Structure]

This is a cyclic version of **Do1**. If the while-condition is satisfied, the first rule is executed whose conditions are satisfied. This is repeated as long as the while condition is satisfied or until a **Stop** statement or transfer call is executed (see Section 2.14, "Stop Statements"). The value of the RuleSet is the value of the last rule that was executed, or **NIL** if no rule was executed.

WhileAll

[RuleSet Control Structure]

This is a cyclic version of **DoAll**. If the while-condition is satisfied, every rule is executed whose conditions are satisfied. This is repeated as long as the while condition is satisfied or until a **Stop** statement is executed. The value of the RuleSet is the value of the last rule that was executed, or **NIL** if no rule was executed.

The "while-condition" is specified in terms of the variables and constants accessible from the RuleSet. The constant **T** can be used to specify a RuleSet that iterates forever (or until a **Stop** statement or transfer is executed). The special variable **ruleApplied** is used to specify a RuleSet that continues as long as some rule was executed in the last iteration. Figure 3 illustrates a simple use of the **WhileAll** control structure to specify a sensing/acting feedback loop for controlling the filling of a washing machine tub with water.

```

RuleSet Name: FillTub;
Workspace Class: WashingMachine;
Control Structure: WhileAll ;
Temp Vars: waterLimit;
WhileCond: T;

(* Rules for controlling the filling of a washing tub with
water.)

{1!} IF loadSetting = 'Small THEN waterLimit_10;
{1!} IF loadSetting = 'Meduim THEN waterLimit_13.5;
{1!} IF loadSetting = 'Large THEN waterLimit_17;
{1!} IF loadSetting = 'ExtraLarge THEN waterLimit_20;

(* Respond to a change of temperature setting at any time.)

IF termperatureSetting = 'Hot
THEN HotWaterValve.Open ColdWaterValve.Close;

IF termperatureSetting = 'Warm
THEN HotWaterValve.Open ColdWaterValve.Open;

IF termperatureSetting = 'Cold
THEN HotWaterValve.Close ColdWaterValve.Open;

(* Stop when the water reaches its limit.)

IF waterLevelSensor.Test >= waterLimit
THEN HotWaterValve.Close ColdWaterValve.Close
(Stop T);

```

Figure 3. RuleSet with WhileAll

There are two control structures in LOOPS that specify iteration over a set of elements in the execution of a RuleSet. These control structures use an explicit while-condition associated with the RuleSet. They are direct extensions of the two primitive control structures above.

FOR1

[RuleSet Control Structure]

This is a cyclic version of **Do1**. If the iteration-condition (or while-condition) is satisfied, the first rule is executed whose conditions are satisfied or until a **Stop** statement is executed. This is repeated as long as the iteration condition is satisfied. The value of the RuleSet is the value of the last rule that was executed, or **NIL** if no rule was executed.

FORALL

[RuleSet Control Structure]

This is a cyclic version of **DoAll**. If the iteration-condition is satisfied, every rule is executed whose conditions are satisfied. This is repeated as long as the iteration condition is satisfied or until a **Stop** statement is executed. The value of the RuleSet is the value of the last rule that was executed, or **NIL** if no rule was executed.

The "iteration-condition" is specified in terms of the variables and constants accessible from the RuleSet. The simplest condition is

(FOR <iterVar> IN <setExpr> DO ruleSet) ;

The **setExpr** will be parsed with the RuleSet parser. The symbol **ruleSet** is a reserved word, and must be spelled as shown (no changes in capitalization).

Here is an example of iteration:

Control Structure: FORALL;

Iteration Condition: (FOR buyer IN (RoadStops (\$ Consumer)) DO ruleSet) ;

For each buyer in the list produced by RoadStops, the ruleSet will be run. In a **FOR1**, the iteration will go on to the next buyer as soon as one rule executes. In a **FORALL**, all rules in the RuleSet will be tried.

For nested iteration one can use a slightly more complicated form, as illustrated by the following example:

**Iteration Condition: (FOR buyer IN (RoadStops (\$ Consumer)) DO
(FOR seller in (RoadStops (\$ Producer)) DO ruleSet)) ;**

An experienced Lisp user can see that this resembles the CLISP iteration construct. In fact, except that you can (must) use the RuleSet syntax in the construct, it is the CLISP construct, and any such construct can be used. A DO1 or DOALL ruleSet will be substituted for the occurrence of the atom ruleSet, depending on whether the Control Structure is a FOR1 or FORALL.

As an abbreviation, if the construct does not contain the atom ruleSet, then (DO ruleSet) is appended to the Iteration Condition for a **FOR1** or **FORALL**. Thus one could write the first example as:

Iteration Condition: (FOR buyer IN (RoadStops (\$ Consumer)))

1.5 One-Shot Rules

One of the design objectives of LOOPS is to clarify the rules by factoring out control information whenever possible. This objective is met in part by the declaration of a control structure for RuleSets.

Another important case arises in cyclic control structures in which some of the rules should be executed only once. This was illustrated in the Washing Machine example in Figure 1 where we wanted to prevent the RuleSet from going into an infinite loop of resetting the breaker, when there was a short circuit in the Washing Machine. Such rules are also useful for initializing data for RuleSets as in the example in Figure 3.

In the absence of special syntax, it would be possible to encode the information that a rule is to be executed only once as follows:

Control Structure: While1
Temporary Vars: triedRule3;

...

IF ~triedRule3 condition condition THEN triedRule3_T action ;

In this example, the variable **triedRule3** is used to control the rule so that it will be executed at most once in an invocation of a RuleSet. However, the prolific use of rules with such control clauses in large systems has led to the common complaint that control clauses in rule languages defeat the expressiveness and conciseness of the rules. For the case above, LOOPS provides a shorthand notation as follows:

{1} IF condition condition THEN action ;

The brace notation means exactly the same thing in the example above, but it more concisely and clearly indicates that the rule executes only once. These rules are called "one shot" or "execute-once" rules.

In some cases, it is desired not only that a rule be executed at most once, but that it be tested at most once. This corresponds to the following:

Control Structure: While1
Temporary Vars: triedRule3;

...

IF ~triedRule3 triedRule3_T condition condition THEN action ;

In this case, the rule will not be tried more than once even if some of the conditions fail the first time that it is tested. The LOOPS shorthand for these rules (pronounced "one shot bang") is

{1!} **IF** *condition* *condition* **THEN** *action* ;

These rules are called "try-once" rules.

The two kinds of one-shot rules are our first examples of the use of meta-descriptions preceding the rule body in braces. See Section 1.7, "Saving an Audit Trail of Rule Invocation," for information on using meta-descriptions for describing the creation of audit trails.

1.6 First/Last Rules

It is sometimes useful to have rules which fire before or after the ordinary part of the RuleSet is invoked, independent of the form of the control structure. For example, in a DO1, such "FIRST " rules could be used for initialization. These now exist, and are notated by putting a {F} for a first rule in the MetaDescription field, and a {L} for a last rule. If a RuleSet has L rules which execute, the value of the RuleSet is the value of the last rule which executed.

1.7 Saving an Audit Trail of Rule Invocation

A basic property of knowledge-based systems is that they use knowledge to infer new facts from older ones. (Here we use the word "facts" as a neutral term, meaning any information derived or given, that is used by a reasoning system.) Over the past few years, it has become evident that reasoning systems need to keep track not only of their conclusions, but also of their reasoning steps. Consequently, the design of such systems has become an active research area in AI. The audit trail facilities of LOOPS support experimentation with systems that can not only use rules to make inferences, but also keep records of the inferential process itself.

1.7.1 Motivations and Applications

Debugging. In most expert systems, knowledge bases are developed over time and are the major investment. This places a premium on the use of tools and methods for identifying and correcting bugs in knowledge bases. By connecting a system's conclusions with the knowledge that it uses to derive them, audit trails can provide a substantial debugging aid. Audit trails provide a focused means of identifying potentially errorful knowledge in a problem solving context.

Explanation Facilities. Expert systems are often intended for use by people other than their creators, or by a group of people *pooling* their knowledge. An important consideration in validating expert systems is that reasoning should be *transparent*, that is, that a system should be able to give an account of its reasoning process. Facilities for doing this are sometimes called *explanation systems*

and the creation of powerful explanation systems is an active research area in AI and cognitive science. The audit trail mechanism provides an essential computational prerequisite for building such systems.

Belief Revision. Another active research area is the development of systems that can "change their minds". This characteristic is critical for systems that must reason from incomplete or errorful information. Such systems get leverage from their ability to make assumptions, and then to recover from bad assumptions by efficiently reorganizing their beliefs as new information is obtained. Research in this area ranges from work on non-monotonic logics, to a variety of approaches to belief revision. The facilities in the rule language make it convenient to use a user-defined calculus of belief revision, at whatever level of abstraction is appropriate for an application.

1.7.2 Overview of Audit Trail Implementation

When *audit mode* is specified for a RuleSet, the compilation of assignment statements on the right-hand sides of rules is altered so that audit records are created as a side-effect of the assignment of values to instance variables. Audit records are LOOPS objects, whose class is specified in RuleSet declarations. The audit records are connected with associated instance variables through the value of the **reason** properties of the variables.

Audit descriptions can be associated with a RuleSet as a whole, or with specific rules. Rule-specific audit information is specified in a property-list format in the meta-description associated with a rule. For example, this can include *certainty factor* information, categories of inference, or categories of support. Rule-specific information overrides RuleSet information.

During rule execution in audit mode, the audit information is evaluated after the rule's LHS has been satisfied and before the rule's RHS is applied. For each rule applied, a single audit record is created and then the audit information from the property list in the rule's meta-description is put into the corresponding instance variables of the audit record. The audit record is then linked to each of the instance variables that have been set on the RHS of the rule by way of the **reason** property of the instance variable.

Additional computations can be triggered by associating active values with either the audit record class or with the instance variables. For example, active values can be specified in the audit record classes in order to define a uniform set of side-effects for rules of the same category. In the following example, such an active value is used to carry out a "certainty factor" calculation.

1.7.3 An Example of Using Audit Trails

The following example illustrates one way to use the audit trail facilities. Figure 4 illustrates a RuleSet which is intended to capture the decisions for evaluating the potential purchase of a washing machine. As with any purchasing situation, this one includes the difficulty of incomplete information about the product. For example in this RuleSet, the reliability of the washing machine is estimated to be 0.5 in the absence of specific information from *Consumer Reports*. The meta-descriptions for the rules, which appear in braces, categorize them in terms of the *basis of belief* (the category *basis* is either a fact or estimate) and a *certainty factor* (*cf*) that is supposed to measure the "implication power" of the

rule. Within the braces, the variable on the left of the assignment statement is always interpreted as meaning a variable in the audit record, and the variables on the right are always interpreted as variables accessible within the RuleSet. This makes it straightforward to experiment with user-defined audit trails and experimental methods of belief revision. (Realistic belief revision systems are usually more sophisticated than this example.)

The result of running the RuleSet is an evaluation report for each candidate machine. Since the RuleSet was run in audit mode, each entry in the evaluation report is tagged with a reason that points to an audit record. Figure 5 illustrates the evaluation report for one machine and one of its audit records. In this example, each of the entries in the report has a reason and a cumulative certainty (cc) property in addition to the value. The value of the reason properties are audit records created as a side effect of running the RuleSet. The auditing process records the meta-description information of each rule in its audit record. This information can be used later for generating explanations or as a basis for belief revision. The auditing process can have side effects. For example, the active in the **cf** variable or the audit record performs a computation to maintain a calculated cumulative certainty in the reliability variable of the evaluation report.

The meta-descriptions for **basis** and **cf** are saved directly in the audit record. The *certainty factor* calculation in this combines information from the audit description with other information already associated with the object. To do this, the **cf** description triggers an active value inherited by the audit record from its class. This active value computes a *cumulative certainty* in the evaluation report. (Other variations on this idea would include certainty information descriptive of the premises of the rule.)

```
RuleSet Name: EvaluateWashingMachine;
Workspace Class: EvaluationReport;
Control Structure: doAll ;
Audit Class: CFAuditRecord ;
Compiler Options: A;

(* Rules for evaluating a potential washing machine for a
purchase.)

.
.
.
{ (basics_Fact cf_1) }
IF buyer:familySize>2 machine:capacity<20
THEN suitability_'Poor;

{ (basics_Fact cf_.8) }
reliability_(_($ ConsumerReports) GetFacts machine);

{ (basics_Estimate cf_.4) }
IF 'reliability THEN reliability_.5;
.
.
.
```

Figure 4. RuleSet Showing Evaluation

```
EvaluationReport "uid1"
expense: 510
```

```

suitability:  Poor cc 1 reason ...
reliability:  .5 cc .6 reason "uid2"
.
.
.
AuditRec "uid2"
rule:  "uid3"
basis: Estimate;
cf:  #(.4 NIL PutCumulativeCertainty)

```

Figure 5. Example of an Audit Trail

1.8 Comparison with Other Rule Languages

This section considers the rationale behind the design of the LOOPS rule language, focusing on ways that it diverges from other rule languages. In general, this divergence was driven by the following observation:

When a rule is heavy with control information, it obscures the domain knowledge that the rule is intended to convey.

Rules are harder to create, understand, and modify when they contain too much control information. This observation led us to find ways to factor control information out of the rules.

1.8.1 The Rationale for Factoring Meta-Level Syntax

One of the most striking features of the syntax of the LOOPS rule language is the factored syntax for meta-descriptions, which provides information about the rules themselves. Traditional rule languages only factor rules into conditions on the left hand side (LHS) and actions on the right hand side (RHS), without general provisions for meta-descriptions.

Decision knowledge expressed in rules is most perspicuous when it is not mixed with other kinds knowledge, such as control knowledge. For example, the following rule:

```

IF ~triedRule4 pluggedInTo:voltage=0
THEN triedRule4_T breaker.Reset;

```

is more obscure than the corresponding one-shot rule from Figure 1:

```

{1} IF pluggedInTo:voltage=0 THEN breaker.Reset;

```

which factors the control information (that the rule is to be applied at most once) from the domain knowledge (about voltages and breakers). In the LOOPS rule language, a meta-description (MD) is specified in braces in front of the LHS of a rule. For another example, the following rule from Figure 4:

```

{{(basis_Fact cf_.8)}}

```

IF buyer:familySize>2 machine:capacity<20
THEN suitability_ 'Poor;

uses an MD to indicate that the rule has a particular **cf** ("certainty factor") and **basis** category for belief support. The MD in this example factors the description of the inference category of the rule from the action knowledge in the rule.

In a large knowledge-based system, a substantial amount of control information must be specified in order to preclude combinatorial explosions. Since earlier rule languages fail to provide a means for factoring meta-information, they must either mix it with the domain knowledge or express it outside the rule language. In the first option, intelligibility is degraded. In the second option, the transparency of the system is degraded because the knowledge is hidden.

1.8.2 The Rationale for RuleSet Hierarchy

Some advocates of production systems have praised the flatness of traditional production systems, and have resisted the imposition of any organization to the rules. The flat organization is sometimes touted as making it *easy to add rules*. The argument is that other organizations diminish the power of pattern-directed invocation and make it more complicated to add a rule.

In designing LOOPS, we have tended to discount these arguments. We observe that there is no inherent property of production systems that can make rules additive. Rather, *additivity* is a consequence of the independence of particular sets of rules. Such independence is seldom achieved in large sets of rules. When rules are dependent, rule invocation needs to be carefully ordered.

Advocates of a flat organization tend to organize large programs as a single very large production system. In practice, most builders of production systems have found it essential to create groups of rules.

Grouping of rules in flat systems can be achieved in part by using *context* clauses in the rules. Context clauses are clauses inserted into the rules which are used to alter the flow of control by naming the context explicitly. Rules in the same "context" all contain an extra clause in their conditions that compares the context of the rules with a current context. Other rules redirect control by switching the current context. Unfortunately, this approach does not conveniently lend itself to the reuse of groups of rules by different parts of a program. Although context clauses admit the creation of "subroutine contexts", they require you to explicitly program a stack of return locations in cases where contexts are invoked from more than one place. The decision to use an implicit calling stack for RuleSet invocation in LOOPS is another example of the our desire to simplify the rules by factoring out control information.

1.8.3 The Rationale for RuleSet Control Structures

Production languages are sometimes described as having a *recognize-act cycle*, which specifies how rules are selected for execution. An important part of this cycle is the *conflict resolution strategy*, which specifies how to choose a production rule when several rules have conditions that are satisfied. For example, the **OPS5** production language has a conflict resolution strategy (**MEA**) which prevents rules

from being invoked more than once, prioritizes rules according to the recency of a change to the data, and gives preference to production rules with the most specific conditions.

In designing the rule language for LOOPS, we have favored the use of a small number of specialized control structures to the use of a single complex conflict resolution strategy. In so doing, we have drawn on some control structures in common use in familiar programming languages. For example, **Do1** is like Lisp's **COND**, **DoAll** is like Lisp's **PROG**, **WhileAll** is similar to **WHILE** statements in many programming languages.

The specialized control structures are intended for concisely representing programs with different control relationships among the rules. For example, the **DoAll** control structure is useful for rules whose effects are intended to be additive and the **Do1** control structure is appropriate for specifying mutually exclusive actions. Without some kind of iterative control structure that allows rules to be executed more than once, it would be impossible to write a simulation program such as the washing machine simulation in Figure 1.

We have resisted a reductionist argument for having only one control structure for all programming. For example, it could be argued that the control structure **Do1** is not strictly necessary because any RuleSet that uses **Do1** could be rewritten using **DoAll**. For example, the rules

Control Structure: Do1;

```
IF a b c THEN d e ;
IF a b c THEN d e ;
IF a b c THEN d e ;
```

could be written alternatively as

Control Structure: DoAll;
Task Vars: firedSomeRule;

```
IF a b c THEN firedSomeRule_T d e ;
IF ~firedSomeRule a b c THEN firedSomeRule_T d e ;
IF ~firedSomeRule a b c THEN firedSomeRule_T d e ;
```

However, the **Do1** control structure admits a much more concise expression of mutually exclusive actions. In the example above, the **Do1** control structure makes it possible to abbreviate the rule conditions to reflect the assumption that earlier rules in the RuleSet were not satisfied.

For some particular sets of rules the conditions are naturally mutually exclusive. Even for these rules **Do1** can yield additional conciseness. For example, the rules:

Control Structure: Do1;

```
IF a b c THEN d e ;
IF ~a b c THEN d e ;
IF ~a ~b c THEN d e ;
```

can be written as

Control Structure: Do1;

```
IF a b c THEN d e ;  
IF  b c THEN d e ;  
IF   c THEN d e ;
```

Similarly it could be argued that the **Do1** and **DoAll** control structures are not strictly necessary because such RuleSets can always be written in terms of **While1** and **WhileAll**. Following this reductionism to its end, we can observe that every RuleSet could be re-written in terms of **WhileAll**.

1.8.4 The Rationale for an Integrated Programming Environment

RuleSets in LOOPS are integrated with procedure-oriented, object-oriented, and data-oriented programming paradigms. In contrast to single-paradigm rule systems, this integration has two major benefits. It facilitates the construction of programs which don't entirely fit the rule-oriented paradigm. Rule-oriented programming can be used selectively for representing just the appropriate decision-making knowledge in a large program. Integration also makes it convenient to use the other paradigms to help organize the interactions between RuleSets.

Using the object-oriented paradigm, RuleSets can be invoked as methods for LOOPS objects. Figure 6 illustrates the installation of the RuleSet **SimulateWashingMachineRules** to carry out the **Simulate** method for instances of the class **WashingMachine**. This definition of the class **WashingMachine** specifies that Lisp functions are to be invoked for Fill and Wash messages. For example, the Lisp function **WashingMachine.Fill** is to be applied when a Fill message is received. When a Simulate message is received, the RuleSet **SimulateWashingMachineRules** is to be invoked with the washing machine as its work space. Simulate message to invoke the RuleSet may be sent by any LOOPS program, including other RuleSets.

The use of object-oriented paradigm is facilitated by special RuleSet syntax for sending messages to objects, and for manipulating the data in LOOPS objects. In addition, RuleSets, work spaces, and tasks are implemented as LOOPS objects.

```

[DEFCLASS WashingMachine
  (MetaClass Class Edited (* "rtk: 12-Jun-87 07:57")
    doc (* Home appliance for wachine clooths.))
  (Supers ElectricalDevice PlumbedDevice CleaningDevice)
  (ClassVariables)
  (InstanceVariables
    (controlSetting Meduim
      doc (* One of Small, Medium, Large, ExtraLarge))...)
  (Methods
    (Fill WashingMachine.Fill doc (* Fill the tub with water.))
    (Wash WashingMachine.Wash doc (* Perofrm the wash cycle.))
    (Simulate UseRuleSet RuleSet SimulateWashingMachineRules)
    .
    .
    .
  ]

```

Figure 6. RuleSet Invoked as a Method

Using the data-oriented paradigm, RuleSets can be installed in active values so that they are triggered by side-effect when LOOPS programs get or put data in objects. For example:

```

[DEFINST WashingMachine (StefiksMaytagWasher "uid2")
  (controlSetting RegularFabric)
  (loadSetting #(Medium NIL RSPut) RSPutFn CheckOverLoadRules)
  (waterLevelSensor "uid3")
]

```

The above code illustrates a RuleSet named **CheckOverLoadRules** which is triggered whenever a program changes the **loadSetting** variable in the **WashingMachine** instance in the figure. This data-oriented triggering can be caused by any LOOPS program when it changes the variable, whether or not that program is written in the rules language.

TABLE OF CONTENTS

PREFACE	v
CONVERT-LOOPS-FILES	1
LOOPSBACKWARDS	3
LOOPSMIXIN	7
RULES	9
INDEX	INDEX-1

[This page intentionally left blank]

2. THE RULE LANGUAGE

This chapter describes the syntax and semantics of the rule language.

2.1 Language Introduction

A rule in LOOPS describes actions to be taken when specified conditions are satisfied. A rule has three major parts called the *left hand side* (LHS) for describing the conditions, the *right hand side* (RHS) for describing the actions, and the *meta-description* (MD) for describing the rule itself. In the simplest case without a meta-description, there are two equivalent syntactic forms:

LHS -> *RHS*;

IF *LHS* **THEN** *RHS*;

The **If** and **Then** tokens are recognized in several combinations of upper and lower case letters. The syntax for LHSs and RHSs is given below. In addition, a rule can have no conditions (meaning always perform the actions) as follows:

-> *RHS*;

if T then *RHS*;

Rules can be preceded by a meta-description in braces as in:

{*MD*} *LHS* -> *RHS*;

{*MD*} **If** *LHS* **Then** *RHS*;

{*MD*} *RHS*;

Examples of meta-information include rule-specific control information, rule descriptions, audit instructions, and debugging instructions. For example, the syntax for one-shot rules shown in Section 1.5, "One-Shot Rules:"

{1} IF *condition* *condition* **THEN** *action* ;

is an example of a meta-description. Another example is the use of meta-assignment statements for describing audit trails and rules. These statements are discussed in Section 1.7, "Saving an Audit Trail of Rule Invocation."

LHS Syntax: The clauses on the LHS of a rule are evaluated in order from left to right to determine whether the LHS is satisfied. If they are all satisfied, then the rule is satisfied. For example:

A B C+D (Prime D) -> RHS;

In this rule, there are four clauses on the LHS. If the values of some of the clauses are **NIL** during evaluation, the remaining clauses are not evaluated. For example, if **A** is non-**NIL** but **B** is **NIL**, then the LHS is not satisfied and **C+D** will not be evaluated.

RHS Syntax: The RHS of a rule consists of actions to be performed if the LHS of the rule is satisfied. These actions are evaluated in order from left to right. Actions can be the invocation of RuleSets, the sending of LOOPS messages, Interlisp function calls, variables, or special termination actions.

RuleSets always return a value. The value returned by a RuleSet is the value of the last rule that was executed. Rules can have multiple actions on the right hand side. Unless there is a **Stop** statement or transfer call as described later, the value of a rule is the value of the last action. When a rule has no actions on its RHS, it returns **NIL** as its value.

Comments: Comments can be inserted between rules in the RuleSet. They are enclosed in parentheses with an asterisk for the first character as follows:

(* This is a comment)

2.2 Kinds of Variables

LOOPS distinguishes the following kinds of variables:

RuleSet arguments: All RuleSets have the variable **self** as their workspace. References to **self** can often be elided in the RuleSet syntax. For example, the expression **self.Print** means to send a **Print** message to **self**. This expression can be shortened to **.Print**. Other arguments can be defined for RuleSets. These are declared in an **Args:** declaration.

Instance variables: All RuleSets use a LOOPS object for their workSpace. In the LHS and RHS of a rule, the first interpretation tried for an undeclared literal is as an instance variable in the work space. Instance variables can be indicated unambiguously by preceding them with a colon, (e.g., **:varName** or **obj:varName**).

Class variables: Literals can be used to refer to class variables of LOOPS objects. These variables must be preceded by a double colon in the rule language, (e.g., **::classVarName** or **obj::classVarName**).

Temporary variables: Literals can also be used to refer to temporary variables allocated for a specific invocation of a RuleSet. These variables are initialized to **NIL** when a RuleSet is invoked. Temporary variables are declared in the **Temporary Vars** declaration in a RuleSet.

Audit record variables: Literals can also be used to refer to instance variables of audit records created by rules. These literals are used only in *meta-assignment* statements in the MD part of a rule. They are used to describe the information saved in audit records, which can be created as a side-effect of rule execution. These variables are ignored if a RuleSet is not compiled in *audit* mode. Undeclared variables appearing on the left side of assignment statements in the MD part of a rule are treated as

audit record variables by default. These variables are declared indirectly -- they are the instance variables of the class declared as the *Audit Class* of the RuleSet.

Interlisp variables: Literals can also be used to refer to Interlisp variables during the invocation of a RuleSet. These variables can be global to the Interlisp environment, or are bound in some calling function. Interlisp variables can be used when procedure-oriented and rule-oriented programs are intermixed. Interlisp variables must be preceded by a backSlash in the syntax of the rule language (e.g., *VispVarName*).

Reserved Words: The following literals are treated as *read-only* variables with special interpretations:

self	[Variable]
The current work space.	
rs	[Variable]
The current RuleSet.	
caller	[Variable]
The RuleSet that invoked the current RuleSet, or NIL if invoked otherwise.	
ruleApplied	[Variable]
Set to T if some rule was applied in this cycle. (For use only in while-conditions).	

The following reserved words are intended mainly for use in creating audit trails:

ruleObject	[Variable]
Variable bound to the object representing the rule itself.	
ruleNumber	[Variable]
Variable bound to the sequence number of the rule in a RuleSet.	
ruleLabel	[Variable]
Variable bound to the label of a rule or NIL .	
reasons	[Variable]
Variable bound a list of audit records supporting the instance variables mentioned on the LHS of the rule. (Computed at run time.)	

auditObject [Variable]

Variable bound to the object to which the reason record will be attached. (Computed at run time.)

auditVarName [Variable]

Variable bound to the name of the variable on which the reason will be attached as a property.

Other Literals: As described later, literals can also refer to Interlisp functions, LOOPS objects, and message selectors. They can also be used in strings and quoted constants.

The determination of the meaning of a literal is done at compile time using the declarations and syntax of RuleSets. The characters used in literals are limited to alphabetic characters and numbers. The first character of a literal must be alphabetic.

The syntax of literals also includes a compact notation for sending unary messages and for accessing instance variables of LOOPS objects. This notation uses *compound literals*. A compound literal is a literal composed of multiple parts separated by a periods, colons, and commas.

2.3 Rule Forms

Quoted Constants: The quote sign is used to indicate constant literals:

a b=3 c='open d=f e='(This is a quoted expression) -> ...

In this example, the LHS is satisfied if **a** is non-NIL, and the value of **b** is 3, and the value of **c** is exactly the atom **open**, the value of **d** is the same as the value of **f**, and the value of **e** is the list **(This is a quoted expression)**.

Strings: The double quote sign is used to indicate string constants:

IF a b=3 c='open d=f e=="This is a string"
THEN (WRITE "Begin configuration task") ... ;

In this example, the LHS is satisfied if **a** is non-NIL, and the value of **b** is 3, and the value of **c** is exactly the atom **open**, the value of **d** is the same as the value of **f**, and the value of **e** equal to the string **"This is a string"**.

Interlisp Constants: The literals **T** and **NIL** are interpreted as the Interlisp constants of the same name.

a (Foo x NIL b) -> x_T ...;

In this example, the function **Foo** is called with the arguments **x**, **NIL**, and **b**. Then the variable **x** is set to **T**.

2.4 Infix Operators and Brackets

To enhance the readability of rules, a few infix operators are provided. The following are infix binary operators in the rule syntax:

+	[Rule Infix Operator]
Addition.	
++	[Rule Infix Operator]
Addition modulo 4.	
-	[Rule Infix Operator]
Subtraction.	
--	[Rule Infix Operator]
Subtraction modulo 4.	
*	[Rule Infix Operator]
Multiplication.	
/	[Rule Infix Operator]
Division.	
>	[Rule Infix Operator]
Greater than.	
<	[Rule Infix Operator]
Less than.	
>=	[Rule Infix Operator]
Greater than or equal.	
<=	[Rule Infix Operator]
Less than or equal.	
=	[Rule Infix Operator]
EQ -- simple form of equals. Works for atoms, objects, and small integers.	
~=	[Rule Infix Operator]
NEQ. (Not EQ.)	

== [Rule Infix Operator]

EQUAL -- long form of equals.

<< [Rule Infix Operator]

Member of a list. (**FMEMB**)

In addition, the rule syntax provides two unary operators as follows:

- [Rule Unary Operator]

Minus.

~ [Rule Unary Operator]

Not.

The precedence of operators in rule syntax follows the usual convention of programming languages. For example

1+5*3 = 16

and

[3 < 2 + 4] = T

Brackets can be used to control the order of evaluation:

[1+5]*3 = 18

Ambiguity of the minus sign: Whenever there is an ambiguity about the interpretation of a minus sign as a unary or binary operator, the rule syntax interprets it as a binary minus. For example

a-b c d -e [-f] (g -h) (_ (\$ Foo) Move -j) -> ...

In this example, the first and second minus signs are both treated as binary subtraction statements. That is, the first three clauses are (1) **a-b**, (2) **c** and (3) **d-e**. Because the rule syntax allows arbitrary spacing between symbols and there is no syntax to separate clauses on the LHS of a rule, the interpretation of "**d -e**" is as a single clause (with the subtraction) instead of two clauses. To force the interpretation as a unary minus operator, one must use brackets as illustrated in the next clause. In this clause, the minus sign in the clause **[-f]** is treated as a unary minus because of the brackets. The minus sign in the function call **(g -h)** is treated as unary because there is no preceding argument. Similarly, the **-j** in the message expression is treated as unary because there is no preceding argument.

2.5 Interlisp Functions and Message Sending

Calls to Interlisp functions are parenthesized with the function name as the first literal after the left parenthesis. Each expression after the function name is treated as an argument to the function. For example:

a (Prime b) [a -b] -> c (Display b c+4 (Cursor x y) 2) ;

In this example, **Prime**, **Display**, and **Cursor** are interpreted as the names of Interlisp functions. Since the expression **[a -b]** is surrounded by brackets instead of parentheses, it is recognized as meaning **a** minus **b** as opposed to a call to the function **a** with the argument minus **b**. In the example above, the call to the Interlisp function **Display** has four arguments: **b**, **c+4**, the value of the function call **(Cursor x y)**, and **2**.

The use of Interlisp functions is usually outside the spirit of the rule language. However, it enables the use of Boolean expressions on the LHS beyond simple conjunctions. For example:

a (OR (NOT b) x y) z -> ... ;

LOOPS Objects and Message Sending: LOOPS classes and other named objects can be referenced by using the dollar notation. The sending of LOOPS messages is indicated by using a left arrow. For example:

**IF cell_ ((\$ LowCell) Occupied? 'Heavy)
THEN (_ cell Move 3 'North);**

In the LHS, an **Occupied?** message is sent to the object named **LowCell**. In the message expression on the RHS, there is no dollar sign preceding **cell**. Hence, the message is sent to the object that is the value of the variable **cell**.

For unary messages (i.e., messages with only the selector specified and the implicit argument **self**), a more compact notation is available as described below.

Unary Message Sending: When a period is used as the separator in a compound literal, it indicates that a unary message is to be sent to an object. (We will alternatively refer to a period as a *dot*.) For example:

tile.Type='BlueGreenCross command.Type='Slide4 -> ... ;

In this example, the object to receive the unary message **Type** is referenced indirectly through the **tile** instance variable in the work space. The left literal is the variable **tile** and its value must be a LOOPS object at execution time. The right literal must be a method selector for that object.

The dot notation can be combined with the dollar notation to send unary messages to named LOOPS objects. For example,

\$Tile.Type='BlueGreenCross ...

In this example, a unary **Type** message is sent to the LOOPS object whose name is **Tile**.

The dot notation can also be used to send a message to the work space of the RuleSet, that is, **self**. For example, the rule

IF scale>7 THEN .DisplayLarge;

would cause a **DisplayLarge** message to be sent to **self**. This is an abbreviation for

IF scale>7 THEN self.DisplayLarge;

2.6 Variables and Properties

When a single colon (:) is used in a literal, it indicates access to an instance variable of an object. For example:

tile:type='BlueGreenCross command:type=Slide4 -> ... ;

In this example, access to the LOOPS object is indirect in that it is referenced through an instance variable of the work space. The left literal is the variable **tile**, and its value must be a LOOPS object when the rule is executed. The right literal **type** must be the name of an instance variable of that object. The compound literal **tile:type** refers to the value of the **type** instance variable of the object in the instance variable **tile**.

The colon notation can be combined with the dollar notation to access a variable in a named LOOPS object. For example,

\$TopTile:type='BlueGreenCross ...

refers to the **type** variable of the object whose LOOPS name is **TopTile**.

A double colon notation (::) is provided for accessing class variables. For example

truck::MaxGas<45 ::ValueAdded>600 -> ... ;

In this example, **MaxGas** is a class variable of the object bound to **truck**. **ValueAdded** is a class variable of **self**.

A colon-comma notation (:,) is provided for accessing property values of class and instance variables. For example

wire:,capacitance>5 wire:voltage:,support='simulation -> ...

In the first clause, **wire** is an instance variable of the work space and **capacitance** is a property of that variable. The interpretation of the second clause is left to right as usual: (1) the object that is the value of the variable **wire** is retrieved, and (2) the **support** property of the **voltage** variable of that object is retrieved. For properties of class variables

::Wire:,capacitance>5 node::Voltage:,support='simulation -> ...

In the first clause, **wire** is a class variable of the work space and **capacitance** is a property of that variable. In the second clause, **node** is an instance variable bound to some object. **Voltage** is a class variable of that object, and **Support** is a property of that class variable.

The property notation is illegal for ruleVars and lispVars since those variables cannot have properties.

2.7 Computing Selectors and Variable Names

The short notations for instance variables, properties, and unary messages all show the selector and variable names *as they actually appear* in the object.

```
object.selector
object:ivName
object::cvName
object:varname:,propName
```

```
(_ object selector arg arg )
```

For example,

apple:flavor

refers to the **flavor** instance variable of the object bound to the variable **apple**. In Interlisp terminology, this implies implicit quoting of the name of the instance variable (**flavor**).

In some applications it is desired to be able to compute the names. For this, the LOOPS rule language provides analogous notations with an added exclamation sign (!). After the exclamation sign, the interpretation of the variable being evaluated starts over again. For example

apple:!x

refers to the same thing as **apple:flavor** if the Interlisp variable **x** is bound to **flavor**. The fact that **x** is a Lisp variable is indicated by the backslash. If **x** is an instance variable of **self** or a temporary variable, we could use the notation:

apple:!x

If **x** is a class variable of **self**, we could use the notation:

apple!:::x

All combinations are possible, including:

```
object.!selector
object.!\\selector
object!:::selector
object:!ivName
object!:!cvName
object:!varname:,propName
```

```
(_! object selector arg arg )
```

2.8 Recursive Compound Literals

Multiple colons or periods can be used in a literal, For example:

a:b:c

means to (1) get the object that is the value of **a**, (2) get the object that is the value of the **b** instance variable of **a**, and finally (3) get the value of the **c** instance variable of that object.

Similarly, the notation

a.b:c

means to get the **c** variable of the object returned after sending a **b** message to the object that is the value of the variable **a**. Again, the operations are carried out left to right: (1) the object that is the value of the variable **a** is retrieved, (2) it is sent a **b** message which must return an object, and then (3) the value of the **c** variable of that object is retrieved.

Compound literal notation can be nested arbitrarily deeply.

2.9 Assignment Statements

An assignment statement using a left arrow can be used for setting all kinds of variables. For example,

x_a;

sets the value of the variable **x** to the value of **a**. The same notation works if **x** is a task variable, rule variable, class variable, temporary variable, or work space variable. The right side of an assignment statement can be an expression as in:

x_a*b + 17*(LOG d);

The assignment statement can also be used with the colon notation to set values of instance variables of objects. For example:

y:b_0 ;

In this example, first the object that is the value of **y** is computed, then the value of its instance variable **b** is set to **0**.

Properties: Assignment statements can also be used to set property values as in:

box:x:,origin_47 fact:,reason_currentSupport;

Nesting: Assignment statements can be nested as in

a_b_c:d_3;

This statement sets the values of **a**, **b**, and the **d** instance variable of **c** to **3**. The value of an assignment statement itself is the new assigned value.

2.10 Meta-Assignment Statements

Meta-assignment statements are assignment statements used for specifying rule descriptions and audit trails. These statements appear in the MD part of rules.

Audit Trails: The default interpretation of meta-assignment statements for undeclared variables is as audit trail specifications. Each meta-assignment statement specifies information to be saved in audit records when a rule is applied. In the following example from Figure 4, the audit record must have variables named **basis** and **cf**:

```
{{(basis_Fact cf_1.)}
IF buyer:familySize>2 machine:capacity<20
THEN suitability_'Poor';
```

In this example, the RHS of the rule assigns the value of the work space instance variable **suitability** to **'Poor'** if the conditions of the rule are satisfied. In addition, if the RuleSet was compiled in *audit* mode, then during RuleSet execution an audit record is created as a side-effect of the assignment. The audit record is attached to the **reason** property of the suitability variable. It has instance variables **basis** and **cf**.

In general, an audit description consists of a sequence of meta-assignment statements. The assignment variable on the left must be an instance variable of the audit record. The class of the audit record is declared in the *Audit Class* declaration of the RuleSet. The expression on the right is in terms of the variables accessible by the RuleSet. If the conditions of a rule are satisfied, an audit record is instantiated. Then the meta-assignment statements are evaluated in the execution context of the RuleSet and their values are put into the audit record. A separate audit record is created for each of the object variables that are set by the rule.

2.11 Push and Pop Statements

A compact notation is provided for pushing and popping values from lists. To push a new value onto a list, the notation **_+** is used:

```
myList_+newItem;
```

```
focus:goals_+newGoal;
```

To pop an item from a list, the **_-** notation is used:

```
item_-myList;
```

nextGoal_-focus:goals;

As with the assignment operator, the push and pop notation works for all kinds of variables and properties. They can be used in conjunction with infix operator << for membership testing.

2.12 Invoking RuleSets

One of the ways to cause RuleSets to be executed is to invoke them from rules. This is used on the LHS of rules to express predicates in terms of RuleSets, and on the RHS of rules to express actions in terms of RuleSets. A short double-dot syntax(..) for this is provided that invokes a RuleSet on a work space:

Rs1..ws1

In this example, the RuleSet bound to the variable **Rs1** is invoked with the value of the variable **ws1** as its work space. The value of the invocation expression is the value returned by the RuleSet. The double-dot syntax can be combined with the dollar notation (\$) to invoke a RuleSet by its LOOPS name, as in

\$MyRules..ws1

which invokes the RuleSet object that has the LOOPS name **MyRules**.

This form of RuleSet invocation is like subroutine calling, in that it creates an implicit stack of arguments and return addresses. This feature can be used as a mechanism for *meta-control* of RuleSets as in:

IF breaker:status='Open
THEN source_\$OverLoadRules..washingMachine;

IF source='NotFound
THEN \$ShortCircuitRules..washingMachine;

In this example, two "meta-rules" are used to control the invocation of specialized RuleSets for diagnosing overloads or short circuits.

2.13 Transfer Calls

An important optimization in many recursive programs is the elimination of tail recursion. For example, suppose that the RuleSet A calls B, B calls C, and C calls A recursively. If the first invocation of A must do some more work after returning from B, then it is useful to save the intermediate states of each of the procedures in frames on the calling stack. For such programs, the space allocation for the stack must be enough to accommodate the maximum depth of the calls.

There is a common and special case, however, in which it is unnecessary to save more than one frame on the stack. In this case each RuleSet has no more work to do after invoking the other RuleSets, and the value of each RuleSet is the value returned by the RuleSet that it invokes. RuleSet invocation in this case amounts to the evaluation of arguments followed by a direct transfer of control. We call such invocations transfer calls.

The LOOPS rule language extends the syntax for RuleSet invocation and message sending to provide this as follows:

RS..*ws

The RuleSet **RS** is invoked on the work space **ws**. With transfer calls, RuleSet invocations can be arbitrarily deep without using proportional stack space.

2.14 Stop Statements

To provide premature terminations in the execution of a RuleSet, the Stop statement is provided.

<u>(Stop <i>value</i>)</u>	<u>[RuleSet Statement]</u>

value is the value to be returned by the RuleSet.

[This page intentionally left blank]

Overview of the Manual

This manual describes the Users' Modules for Xerox's Lisp Object-Oriented Programming System, LOOPS (TM), to developers.

Note: Venue does not support LOOPS Users' Modules. However, each Users' Module contains the name and network mailing address of the person who wrote or last modified that module, and the date it was written or last modified.

This manual describes the Lyric/Medley Release of the LOOPS Users' Modules, which run under the Lyric and Medley Releases of Lisp.

Organization of the Manual and How to Use It

This manual is divided into chapters, with each chapter describing a separate Users' Module.

To use the manual, read the chapter that corresponds to the Users' Module you want to use. A general Table of Contents is provided to help you locate specific information.

Conventions

This manual uses the following conventions:

- Case is significant in LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown. Typically, this means that method names are capitalized and variables are not.
- Arguments appear in italic type.
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a method appears as follows:

`(← self Selector Arg1 Arg2)`

- Examples appear in the following typeface:

`89← (←LOGIN)`

- All examples are typed into an Interlisp Exec. This is the recommended Exec for all LOOPS expressions.
- Methods with an exclamation mark (!) suffix usually perform operations deeply into class structure instead of only on a given object.

- Methods with a question mark (?) suffix usually are predicates; that is, truth functions.
- Methods often appear in the form **ClassName.SelectorName**.
- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

References

The following books and manuals augment this manual.

LOOPS Reference Manual

LOOPS Release Notes

LOOPS Library Modules Manual

Interlisp-D Reference Manual

Common Lisp: the Language by Guy Steele

Common Lisp Implementation Notes, Lyric Release

Lisp Release Notes, Lyric Release and Medley Release

Lisp Library Modules Manual, Lyric Release and Medley Release

3. USING RULES IN LOOPS

The LOOPS rules language is supported by an integrated programming environment for creating, editing, compiling, and debugging RuleSets. This section describes how to use that environment.

3.1 Creating RuleSets

RuleSets are named LOOPS objects and are created by sending the class **RuleSet** a **New** message as follows:

```
(_ ($ RuleSet) New)
```

After entering this form, the user will be prompted for a LOOPS name as

RuleSet name: *RuleSetName*

Afterwards, the RuleSet can be referenced using LOOPS dollar sign notation as usual. It is also possible to include the RuleSet name in the **New** message as follows:

```
(_ ($ RuleSet) New NIL RuleSetName)
```

3.2 Editing RuleSets

A RuleSet is created empty of rules. The RuleSet editor is used to enter and modify rules. The editor can be invoked with an **EditRules** message (or **ER** shorthand message) as follows:

```
(_ RuleSet EditRules)
```

```
(_ RuleSet ER)
```

If a RuleSet is installed as a method of a class, it can be edited conveniently by selecting the **EditMethod** option from a browser containing the class. Alternatively, the **EditMethod** message can be used:

```
(_ ClassName EditMethod selector) [Message]
```

Both approaches to editing retrieve the source of the RuleSet and put the user into the TTYIN or TEdit editor, treating the rule source as text.

Initially, the source is a template for RuleSets as shown in Figure 7. The rules are entered after the comment at the bottom. The declarations at the beginning are filled in as needed and superfluous declarations can be discarded.

```

RuleSet Name: RuleSetName;
Workspace Class:  ClassName;
Control Structure: doAll;
While Condition:  ;
Audit Class:  StandardAuditRecord;
Rule Class:  Rule;
Task Class:  ;
Meta Assignments:  ;
Temporary Vars:  ;
Lisp Vars:  ;
Debug Vars:  ;
Compiler Options:  ;

(* Rules for whatever.  Comment goes here.)

```

Figure 7. Initial Template for a RuleSet

You can then edit this template to enter rules and set the declarations at the beginning. In the current version of the rule editor, most of these declarations are left out. If you choose the **EditAllDecls** option in the RuleSet editor menu, the declarations and default values will be printed in full.

The template is only a guide. Declarations that are not needed can be deleted. For example, if there are no temporary variables for this RuleSet, the **Temporary Vars** declaration can be deleted. If the control structure is not one of the **while** control structures, then the **While Condition** declaration can be deleted. If the compiler option **A** is not chosen, then the **Audit Class** declaration can be deleted.

When you leave the editor, the RuleSet is compiled automatically into a Lisp function.

If a syntax error is detected during compilation, an error message is printed and you are given another opportunity to edit the RuleSet.

3.3 Copying RuleSets

Sometimes it is convenient to create new RuleSets by editing a copy of an existing RuleSet. For this purpose, the method **CopyRules** is provided as follows:

```

(_ oldRuleSet CopyRules newRuleSetName) [Message]

```

This creates a new RuleSet by some of the information from the perspectives of the old RuleSet. It also updates the source text of the new RuleSet to contain the new name.

3.4 Saving RuleSets on Lisp Files

RuleSets can be saved on Lisp files just like other LOOPS objects. In addition, it is usually useful to save the Lisp functions that result from RuleSet compilation. In the current implementation, these functions have the same names as the RuleSets themselves. To save RuleSets on a file, it is necessary to add two statements to the file commands for the file as follows:

```
(FNS * MyRuleSetNames)
(INSTANCES * MyRuleSetNames)
```

where **MyRuleSetNames** is a Lisp variable whose value is a list of the names of the RuleSets to be saved.

If RuleSets are methods associated with a class, and they are saved by using (FILES?), then the file package saves the appropriate entries. The user does not have to be concerned with editing the filecoms of the file being made.

3.5 Printing RuleSets

To print a RuleSet without editing it, one can send a **PPRules** or **PPR** message as follows:

```
( _ RuleSet PPRules) [Message]
( _ RuleSet PPR) [Message]
```

A convenient way to make hardcopy listings of RuleSets is to use the function **ListRuleSets**. The files will be printed on the **DEFAULTPRINTINGHOST** as is standard in Interlisp-D. **ListRuleSets** can be given four kinds of arguments as follows:

```
(ListRuleSets RuleSetName)
(ListRuleSets ListOfRuleSetNames)
(ListRuleSets ClassName)
(ListRuleSets FileName)
```

In the *ClassName* case, all of the RuleSets that have been installed as methods of the class will be printed. In the last case, all of the RuleSets stored in the file will be printed.

3.6 Running RuleSets from LOOPS

RuleSets can be invoked from LOOPS using any of the usual protocols.

Procedure-oriented Protocol: The way to invoke a RuleSet from LOOPS is to use the **RunRS** function:

(RunRS RuleSet workSpace arg2 ... argN) [Function]

workSpace is the LOOPS object to be used as the work space.
This is "procedural" in the sense that the RuleSet is invoked by its name. *RuleSet* can be either a RuleSet object or its name.

Object-oriented Protocol: When RuleSets are installed as methods in LOOPS classes, they can be invoked in the usual way by sending a message to an instance of the class. For example, if **WashingMachine** is a class with a RuleSet installed for its **Simulate** method, the RuleSet is invoked as follows:

(_ washingMachineInstance Simulate)

Data-oriented Protocol: When RuleSets are installed in active values, they are invoked by side-effect as a result of accessing the variable on which they are installed.

3.7 Installing RuleSets as Methods

RuleSets can also be used as methods for classes. This is done by installing automatically-generated invocation functions that invoke the RuleSets. For example:

```
[DEFCLASS WashingMachine
  (MetaClass Class doc (* comment) ...)
  ...
  (InstanceVariables (owner ...))
  (Methods
    (Simulate RunSimulateWMRules)
    (Check RunCheckWMRules
      doc (* Rules to Check a washing machine.))
  )
...]
```

When an instance of the class **WashingMachine** receives a **Simulate** message, the RuleSet **SimulateWMRules** will be invoked with the instance as its work space.

To simplify the definition of RuleSets intended to be used as Methods, the function **DefRSM** (for "Define Rule Set as a Method") is provided:

(DefRSM ClassName Selector RuleSetName) [Function]

If the optional argument *RuleSetName* is given, **DefRSM** installs that RuleSet as a method using the *ClassName* and *Selector*. It does this by automatically generating an installation function as a method to invoke the RuleSet. **DefRSM** automatically documents the installation function and the method.
If the argument *RuleSetName* is **NIL**, then **DefRSM** creates the RuleSet object, puts the user into an Editor to enter the rules,

compiles the rules into a Lisp function, and installs the RuleSet as before.

DefRSM can be invoked with the browser as follows:

- Position the cursor over a class in a browser.
- Press the middle mouse button. A menu pops up.
- Select the Add option in this menu, and drag the mouse to the right to display the submenu that includes the "DefRSM" option. You are prompted to enter a selector name.

After a RuleSet has been installed as a method by using **DefRSM**, you can then edit that RuleSet by selecting the "EditMethod" option from the browser editing menu.

3.8 Installing RuleSets in Active Values

Note: The following section and any other references to active values within the rule documentation refer to active values as they were implemented in the Buttriss release. The functionality of triggering rules from active values has not been tested using the current implementation of active values. It should work to use the **ExplicitFnActiveValue** class to implement this behavior.

RuleSets can also be used in data-oriented programming so that they are invoked when data is accessed. To use a RuleSet as a *getFn*, the function **RSGetFn** is used with the property **RSGet** as follows:

```
...
(InstanceVariables
  (myVar #(myVal RSGetFn NIL) RSGet RuleSetName))
...
```

RSGetFn is a LOOPS system function that can be used in an active value to invoke a RuleSet in response to a LOOPS get operation (e.g., **GetValue**) is performed. It requires that the name of the RuleSet be found on the **RSGet** property of the item. **RSGetFn** activates the RuleSet using the local state as the work space. The value returned by the RuleSet is returned as the value of the get operation.

To use a RuleSet as a *putFn*, the function **RSPutFn** is used with the property **RSPut** as follows:

```
...
(InstanceVariables
  (myVar #(myVal NIL RSPutFn) RSPut RuleSetName))
...
```

RSPutFn is a function that can be used in an active value to invoke a RuleSet in response to a LOOPS put operation (e.g., **PutValue**). It requires that the name of the RuleSet be found on the **RSPut** property of the item. **RSGetFn** activates the RuleSet using the *newValue* from the put

operation as the work space. The value returned by the RuleSet is put into the local state of the active value.

3.9 Tracing and Breaking RuleSets

LOOPS provides breaking and tracing facilities to aid in debugging RuleSets. These can be used in conjunction with the auditing facilities and the rule executive for debugging RuleSets. The following summarizes the compiler options for breaking and tracing:

T	Trace if rule is satisfied. Useful for creating a running display of executed rules.
TT	Trace if rule is tested.
B	Break if rule is satisfied.
BT	Break if rule is tested. Useful for stepping through the execution of a RuleSet.

Specifying the declaration **Compiler Options: T**; in a RuleSet indicates that tracing information should be displayed when a rule is satisfied. To specify the tracing of just an individual rule in the RuleSet, the **T** meta-descriptions should be used as follows:

{T} IF *cond* THEN *action*;

This tracing specification causes LOOPS to print a message whenever the LHS of the rule is tested, or the RHS of the rule is executed. It is also possible to specify that the values of some variables (and compound literals) are to be printed when a rule is traced. This is done by listing the variables in the **Debug Vars** declaration in the RuleSet:

Debug Vars: a a:b a:b.c;

This will print the values of **a**, **a:b**, and **a:b.c** when any rule is traced or broken.

Analogous specifications are provided for breaking rules. For example, the declaration **Compiler Options: B**; indicates that LOOPS is to enter the rule executive (see Section 3.10, "The Rule Exec") after the LHS is satisfied and before the RHS is executed. The rule-specific form:

{B} IF *cond* THEN *action*;

indicates that LOOPS is to break before the execution of a particular rule.

Sometimes it is convenient in debugging to display the source code of a rule when it is traced or broken. This can be effected by using the **PR** compiler option as in

Compiler Options: T PR;

which prints out the source of a rule when the LHS of the rule is tested and

Compiler Options: B PR;

which prints out the source of a rule when the LHS of a rule is satisfied, and before entering the break.

3.10 The Rule Exec

A Read-Compile-Evaluate-Print loop, called the rule Executive, is provided for the rule language. The rule Executive can be entered during a break by invoking the Lisp function **RE**. During RuleSet execution, the rule executive can be entered by typing **^f** (<control>-f) on the keyboard.

On the first invocation, **RE** prompts the user for a window. It then displays a stack of RuleSet invocations in a menu to the left of this window in a manner similar to the Interlisp-D Break Package. Using the left mouse button in this window creates an Inspector window for the work space for the RuleSet. Using the middle mouse button pretty prints the RuleSet in the default prettyprint window.

In the main rule Executive window, **RE** prompts the user with "**re:**". Anything in the rule language (other than declarations) that is typed to this Executive will be compiled and executed immediately and its value printed out. For example, you may type rules to see whether they execute or variable names to determine their values. For example:

re: trafficLight:color

Red

re:

this example shows how to get the value of the **color** variable of the **trafficLight** object. If the value of a variable was set by a RuleSet running with auditing, then a **why** question can be typed to the rule executive as follows:

re: why trafficLight:color

IF highLight:color = 'Green farmRoadSensor:cars timer.TL
THEN highLight:color _ 'Yellow timer.Start;

Rule 3 of RuleSet LightRules

Edited: Conway "13-Oct-82"

re:

The rule executive may be exited by typing **OK**.

3.11 Auditing RuleSets

Two declarations at the beginning of a RuleSet affect the auditing. Auditing is turned on by the compiler option **A**. The simplest form of this is

Compiler Options: A;

The **Audit Class** declaration indicates the class of the audit record to be used with this RuleSet if it is compiled in *audit* mode.

Audit Class: StandardAuditRecord;

A **Meta Assignments** declaration can be used to indicate the audit description to be used for the rules unless overridden by a rule-specific meta-assignment statement in a meta-descriptor.

Meta Assignments: cf_ .5 support_ 'GroundWff;

3.12 Loading Rules

Set the variable LOOPSUSERSDIRECTORIES to include the directory where the Rules files are stored.

Load the file LOOPSRULES-ROOT.LCOM, which will load the following files from LOOPSUSERSDIRECTORIES:

- LOOPSBACKWARDS.LCOM
- LOOPSMIXIN
- LOOPSRULES.LCOM
- LOOPSRULESP.LCOM
- LOOPSRULESC.LCOM
- LOOPSRULESD.LCOM, which will load the file TTY.LCOM from LISPUSERSDIRECTORIES.

Editing rules will be easier if TEdit is loaded. Loading the Rules does not automatically load TEdit.

3.13 Known Problems

In a rule, the expression \$pipe.ri..\$p compiles to (RunRS (QUOTE (\$ pipe)) (\$ p)), which fails.

Meta-assignment statements cannot handle expressions. This means that statements like {cf _ .5} work fine, but {validity _ 'fact} fails.

A value of 1 in a meta-descriptor statement is always taken to be a one-shot designator. You cannot have a meta-descriptor statement like {cf_1}. However, the number 1.0 can be used; the meta-descriptor statement, {cf_1.0}, works.

Rules have not been tested without loading TEdit in order to edit RuleSets.

[This page intentionally left blank.]

CONVERT-LOOPS-FILES

By: Bob Bane (Bane.pa@Xerox.com)

5-Feb-88

INTRODUCTION

CONVERT-LOOPS-FILES allows you to convert files from the Koto release of LOOPS to the Medley release. The changes from the Koto release are described in detail in the *LOOPS Release Notes*.

PROCEDURE

To convert Koto LOOPS Files to Lyric/Medley Loops:

1. Your files must be from Koto LOOPS. Pre-Koto LOOPS files must first be run through the Buttrech->Koto converter in a Koto LOOPS sysout.
2. Install and load your Lyric LOOPS sysout. You need to have TEdit loaded, as the converter uses it.
3. Load the file CONVERSION-AIDS.DFASL, which defines the function

```
(CONVERT-LOOPS-FILES <list-of-files> <dump-files-p>)
```

where:

<list-of-files> A filename or a list of filenames to be converted.

<dump-files-p> Determines disposition of files.

- If non-NIL, converted files will be dumped back out immediately.
- If :COMPILE, they will be compiled.
- If :COMPILE/LOAD, they will be compiled and the compiled code loaded.

4. Call CONVERT-LOOPS-FILES with the names of the files you want converted and an appropriate option.

Note the following:

- CONVERT-LOOPS-FILES makes more than one pass over the files being converted; the first pass is a TEdit textual change to make the files loadable into Medley LOOPS. If you specify version numbers in your <list-of-files>, these changes will be made in place on your original files.
- The converter doesn't work completely automatically on systems consisting of several files that automatically load themselves with FILES coms; the converter may try to load subfiles before they are converted. It may be necessary to RETURN NIL from the LOAD calls in the break windows that will occur when this problem comes up. Aside from that, the converter works reasonably well, and has been used to convert large LOOPS systems with almost no source code changes from Koto Loops.

[This page intentionally left blank.]

A

assignment statements 40
 audit trail of rule invocation 23
 auditing RuleSets 52
auditObject (*Variable*) 34
auditVarName (*Variable*) 34

B

breaking and tracing RuleSets 50

C

caller (*Variable*) 33
 colon-comma in a literal 38
 comments 32
 comparison with other rule languages 26
 compiler options for breaking and tracing 50
 computing selectors 39
 control structures for selecting rules 18
 converting from Buttriss Rules 55
 copying RuleSets 46
CopyRules (*Message*) 46
 creating RuleSets 45

D

DefAVP (*Function*) 5.5
DefRSM (*Function*) 48
Do1 (*RuleSet Control Structure*) 18
DoAll (*RuleSet Control Structure*) 19
 dollar notation to invoke RuleSets 42
 double colon in a literal 38
 double-dot syntax to invoke RuleSets 42
 double-dot-star syntax to invoke RuleSets 43

E

EditAllDecls 46
 editing RuleSets 45
EditMethod (*Message*) 45
EditRules (*Message*) 45
ER (*Message*) 45
 exclamation sign to compute names 39
 ExplicitFnActiveValue 4

F

factoring meta-level syntax 26
 first/last rules 23
FOR1 (*RuleSet Control Structure*) 21
FORALL (*RuleSet Control Structure*) 21

I

if-then rules 15
 infix operators 35
 installing RuleSets
 as methods 48
 in active values 49
 integrated programming environment 29
 Interlisp
 constants 34
 functions 37
 invoking RuleSets 42
 items in release 1
 iteration-condition in RuleSets 21

L

LHS syntax 31
ListRuleSets (*Function*) 47
 literal 38
 loading rules 52

M

message sending 37
 meta-assignment statements 41
 meta-control of RuleSets 42
 multiple colons in a literal 40

O

one-shot rules 22

P

pop statement 42
PPR (*Message*) 47
PPRules (*Message*) 47
 printing RuleSets 47
 production rules 15
 properties 38
 push statement 42

Q

quoted constants 34

R

RE 51
reasons (*Variable*) 33
 recursive compound literals 40
 RHS syntax 32
rs (*Variable*) 33
RSGGet (*Property*) 49
RSGGetFn (*Function*) 49
RSPut (*Property*) 49
RSPutFn (*Function*) 49
 rule Exec 51
 rule-oriented programming 15
ruleApplied (*Variable*) 33
ruleLabel (*Variable*) 33
ruleNumber (*Variable*) 33
ruleObject (*Variable*) 33
 rules 15

 basic concepts 16
 forms 34
 language 31
 loading 52
 major features 15
 using 45
 work space 15
 RuleSets 15
 approaches to organizing 17
 auditing 52
 breaking and tracing 50
 control structures 28
 copying 46
 creating 45
 editing 45
 hierarchy 27
 installing as methods 48
 installing in active values 49
 invoking 15,42
 iteration condition 21
 meta-control 42

printing 47
 protocols 47
 running from LOOPS 47
 saving on Lisp files 47
 running RuleSets from LOOPS 47
RunRS (*Function*) 48

S

saving RuleSets on Lisp files 47
self (*Variable*) 33
 single colon in a literal 38
Stop (*RuleSet Statement*) 43
 strings 34
 system configuration 1

T

transfer calls 43

U

unary message sending 37
 using rules 45

V

variable names 39
 variables 32,38

W

While1 (*RuleSet Control Structure*) 19
WhileAll (*RuleSet Control Structure*) 20
 work space for rules 15

←
 ←+ push statement 42
 ←- pop statement 42

~
 ~ (*Rule Unary Operator*) 36
 ~= (*Rule Infix Operator*) 36

!
 ! to compute names 39

\$
 \$ to invoke RuleSets 42

*
 * (*Rule Infix Operator*) 35

+
 + (*Rule Infix Operator*) 35
 ++ (*Rule Infix Operator*) 35

-
 - (*Rule Infix Operator*) 35
 - (*Rule Unary Operator*) 36
 -- (*Rule Infix Operator*) 35

.
 .. to invoke RuleSets 42
 ..* to invoke RuleSets 43

/
 / (*Rule Infix Operator*) 35

:
 : in a literal 38
 :, in a literal 38
 :: in a literal 38

<
 < (*Rule Infix Operator*) 35
 << (*Rule Infix Operator*) 36
 <= (*Rule Infix Operator*) 35

=
 = (*Rule Infix Operator*) 36
 == (*Rule Infix Operator*) 36

>
 > (*Rule Infix Operator*) 35
 >= (*Rule Infix Operator*) 35

ntionally left blank]

[This page inte

LOOPSBACKWARDS

By: Bob Bane (Bane.pa@Xerox.com)

15-Dec-87

INTRODUCTION

LOOPSBACKWARDS allows you to run files which were previously converted from the Buttriss release of LOOPS to the Koto release. Unlike the Koto version of LOOPSBACKWARDS conversion methods for moving files from Buttriss LOOPS to Koto LOOPS are included but **not supported**. *We strong recommend that you convert LOOPS source code from Buttriss to Koto using the Koto release of LOOPS.* Conversion of Koto LOOPS source code to Lyric LOOPS is done using the CONVERSION-AIDS users' module.

The changes between the Buttriss and Koto releases are described in detail in the *LOOPS Release Notes* for the **Koto** release. Old features that are included in LOOPSBACKWARDS are summarized here:

- Many functions that were removed from the Buttriss release are defined.
- The messages **List** and **List!** are available.
- The operation of old style active values, from the Buttriss release, are provided.
- Support for reading old style macros, such as `$(localState getFn putFn)` or `#$Mumble`, is available.

LOOPSBACKWARDS is an unsupported LOOPS users module. It is strongly recommended that it only be used as part of an effort to upgrade very old LOOPS code to newer releases. It will allow very old LOOPS code to run well enough that it can be rewritten for a newer release.

INSTALLATION

LOOPSVCOPY will be automatically loaded by LOOPSBACKWARDS.

FUNCTIONS

LOOPSBACKWARDS includes **ExplicitFnActiveValue** and **DefAVP**. **ExplicitFnActiveValue** allows the user code triggered by Get- or Put- accesses to be stored within functions which are pointed to by instance variables rather than requiring the redefinition of **GetWrappedValue** or **PutWrappedValue**. These functions must have the form specified in the **DefAVP** function.

ExplicitFnActiveValue

[Class]

Purpose: Mimics the behavior of the Buttruss style of active values.

Behavior: Get- accesses to the wrapped variable cause the **getFn** to be called, Put- accesses cause **putFn** to be called. Enables the old style activeValue to look like the new style without changing any functionality.

The **getFn** is called by the **ExplicitFnActiveValue** **GetWrappedValue** method. This method passes to the **getFn** the arguments defined by **DefAVP** as described in the *LOOPS Users' Modules*.

The **putFn** is called by the **ExplicitFnActiveValue** **PutWrappedValue** method. This method passes to the **putFn** the arguments defined by **DefAVP** as described in the *LOOPS Users' Modules*.

Instance Variables: **localState** A place for data storage.

getFn The name of a function applied when the active variable is read.

putFn The name of a function applied when the active variable is changed.

Example:

```
32← (← ($ Bin) New 'bin4)
#, ($& Bin (|DAW0.1Y:.H53.]99| . 521))

33← (← ($ ExplicitFnActiveValue) New 'EFAV1)
#, ($& ExplicitFnActiveValue (|DAW0.1Y:.H53.]99| . 522))

34← (DEFINEQ (PrintOnGet
  (self varName localSt propName activeVal type)
  (PRINTOUT T "I am:" , activeVal T) localSt))
(PrintOnGet)

35← (←@ ($ EFAV1) getFn 'PrintOnGet)
PrintOnGet

36← (← ($ EFAV1) AddActiveValue ($ bin4) 'height)
#, ($A #,NestedNotSetValue PrintOnGet NIL)

37← (←@ ($ bin4) height 123)
123

38← (@ ($ bin4) height)
```

```
I am: #, ($ EFAV1)
123
```

(DefAVP *fnName putFlg*) [Function]

Purpose: Creates a template for defining an active value function.

Behavior: Creates a template and leaves you in the Interlisp function display editor.

Arguments: *fnName* Name of the function.
 putFlg T indicates function is a **putFn**, NIL indicates a **getFn**.

Returns: The function name on exit from the editor .

Example: In each of the following cases the template only is shown. User code is to be added immediately after the comment by using the display editor.

```
66← (DefAVP 'AGetFn)
AGetFn

67←PP* AGetFn
(AGetFn
  [LAMBDA (self varName localSt propName activeVal type)
    (* This is a getFn. The value of this getFn is
    returned as the value of the enclosing GetValue.)
    localSt])
(AGetFn)

68← (DefAVP 'APutFn T)
APutFn

69←PP* APutFn
(APutFn
  [LAMBDA (self varName newValue propName activeVal type)
    (* This is a putFn. ***NOTE*** The value of this
    function will be returned as the value of any enclosing
    PutValue. This usually means that you want to return the value
    returned by PutLocalState.)
    (PutLocalState activeVal newValue self varName propName
    type)])
(APutFn)
```

[This page intentionally left blank.]

LOOPSMIXIN

By: Bob Bane (Bane.pa@Xerox.com)

15-Dec-87

INTRODUCTION

LOOPSMIXIN defines several small classes that can be mixed into your application classes. It also defines the class **Perspective** and its support classes, which are used in the implementation of LOOPS Rules. Perspectives allow you to view one object as having more than one class at a time; they have not been tested extensively outside of Rules and are known to have major bugs which don't affect Rules, so they are not documented here.

LOOPSMIXIN Classes

DatedObject - Defines the instance variables **created** and **creator** which are set at object creation time to the values of (DATE) and (USERNAME).

NamedObject - Defines the instance variable **name** and initializes it to an ActiveValue that insures that the LOOPS system name for the containing object is uniquely name; i.e. storing a name for the object in **name** causes the previous name for the object to be removed with **DeleteObjectName** and the new name for the object to be asserted with **NameEntity**.

GlobalNamedObject - A subclass of **NamedObject** that works the same way as **NamedObject**.

ListMetaClass - Specializes the **New** and **DestroyInstance** methods to keep a list of all instances of that class in the class property **AllInstances** for that class.

StrucMeta - A MetaClass useful for creating new classes. Specializes the **New** method to create a Class object by copying the instance variable and class variable descriptions of the current class. Class variables with a non-NIL Local property will not be copied.

TempClass - Specializes the **New** method to always create objects of this class using the **NewTemp** method, insuring that the objects will be temporary objects.

Perspective, Node, Template, TextItem - These classes are used in LOOPS Rules.

[This page intentionally left blank.]

[This page intentionally left blank.]

[This page intentionally left blank.]

RULES

Modified by: Rick Martin (Martin.pasa@Xerox.com)

14-Apr-86

[This page intentionally left blank.]

TABLE OF CONTENTS

1. INTRODUCTION TO RULE-ORIENTED PROGRAMMING IN LOOPS	15
1.1 Introduction.....	15
1.2 Basic Concepts	16
1.3 Organizing a Rule-Oriented Program.....	17
1.4 Control Structures for Selecting Rules	18
1.5 One-Shot Rules.....	22
1.6 First-Last Rules	23
1.7 Saving an Audit Trail of Rule Invocation	23
1.7.1 Motivations and Applications.....	23
1.7.2 Overview of Audit Trail Implementation.....	24
1.7.3 An Example of Using Audit Trails.....	24
1.8 Comparison with Other Rule Languages	26
1.8.1 The Rationale for Factoring Meta-Level Syntax	26
1.8.2 The Rationale for RuleSet Hierarchy.....	27
1.8.3 The Rationale for RuleSet Control Structures	28
1.8.4 The Rationale for an Integrated Programming Environment.....	29
2. THE RULE LANGUAGE	31
2.1 Language Introduction.....	31
2.2 Kinds of Variables	32
2.3 Rule Forms.....	34
2.4 Infix Operators and Brackets.....	35
2.5 Interlisp Functions and Message Sending	37

2.6 Variables and Properties	38
2.7 Computing Selectors and Variable Names	39
2.8 Recursive Compound Literals	40
2.9 Assignment Statements	41
2.10 Meta-Assignment Statements	41
2.11 Push and Pop Statements.....	42
2.12 Invoking RuleSets	42
2.13 Transfer Calls	43
2.14 Stop Statements.....	43
 3. USING RULES IN LOOPS	 45
3.1 Creating RuleSets	45
3.2 Editing RuleSets.....	45
3.3 Copying RuleSets.....	46
3.4 Saving RuleSets on Lisp Files.....	47
3.5 Printing RuleSets.....	47
3.6 Running RuleSets from LOOPS.....	47
3.7 Installing RuleSets as Methods	48
3.8 Installing RuleSets in ActiveValues	49
3.9 Tracing and Breaking RuleSets.....	50
3.10 The Rule Exec.....	51
3.11 Auditing RuleSets.....	52
3.12 Loading Rules	52
3.13 Known Problems	52

LIST OF FIGURES

1. Basic RuleSet.....	18
2. RuleSet Showing Do1	19
3. RuleSet with WhileAll	20
4. RuleSet Showing Evaluation.....	25
5. Example of an Audit Trail	26
6. RuleSet Invoked as a Method	30
7. Initial Template for a RuleSet.....	46

[This page intentionally left blank.]

TABLE OF CONTENTS

PREFACE	v
CONVERT-LOOPS-FILES	1
LOOPSBACKWARDS	3
LOOPSMIXIN	7
RULES	9
INDEX	INDEX-1

[This page intentionally left blank]

Overview of the Manual

This manual describes the Users' Modules for Xerox's Lisp Object-Oriented Programming System, Xerox LOOPS (TM), to developers.

Note: Xerox does not support Xerox LOOPS Users' Modules. However, each Users' Module contains the name and network mailing address of the person who wrote or last modified that module, and the date it was written or last modified.

This manual describes the Lyric/Medley Release of the Xerox LOOPS Users' Modules, which run under the Lyric and Medley Releases of Xerox Lisp.

Organization of the Manual and How to Use It

This manual is divided into chapters, with each chapter describing a separate Users' Module.

To use the manual, read the chapter that corresponds to the Users' Module you want to use. A general Table of Contents is provided to help you locate specific information.

Conventions

This manual uses the following conventions:

- Case is significant in Xerox LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown. Typically, this means that method names are capitalized and variables are not.
- Arguments appear in italic type.
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a method appears as follows:

`(_ self Selector Arg1 Arg2)`

- Examples appear in the following typeface:

`89_ (_ LOGIN)`

- All examples are typed into an Interlisp Exec. This is the recommended Exec for all Xerox LOOPS expressions.
- Methods with an exclamation mark (!) suffix usually perform operations deeply into class structure instead of only on a given object.

- Methods with a question mark (?) suffix usually are predicates; that is, truth functions.
- Methods often appear in the form **ClassName.SelectorName**.
- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

References

The following books and manuals augment this manual.

Xerox LOOPS Reference Manual

Xerox LOOPS Release Notes

Xerox LOOPS Library Modules Manual

Interlisp-D Reference Manual

Common Lisp: the Language by Guy Steele

Xerox Common Lisp Implementation Notes, Lyric Release

Xerox Lisp Release Notes, Lyric Release and Medley Release

Xerox Lisp Library Modules Manual, Lyric Release and Medley Release

Writer's Notes -- Conventions

This file includes notes on conventions for *Xerox LOOPS Users' Modules Manual*, Lyric Beta Release. This manual is packaged in one binder.

Writer: Raven Kontur Brewster

Printing Date: 22 February 1988

Directories and Files

The directory {ERIS}<Doc>Loops>Lyric>Beta>UserMods> contains the files for the manual. This directory has the following subdirectories:

- {ERIS}<Doc>Loops>Lyric>Beta>UserMods>Z-ReleaseInfo> contains this file on writing conventions and a file on production details.

Filenames describe the contents of the file. For example, the filename

{ERIS}<Doc>Loops>Lyric>Beta>UserMods>LoopsMixin.tedit

contains the chapter on LoopsMixin.

Assemble the files in the following order for the manual:

{ERIS}<Doc>Loops>Lyric>Beta>UserMods>A1-TitlePage.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>A2-TOC.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>A3-Preface.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>Converter.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>LoopsBackwards.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>LoopsMixin.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>Rules-A1-TitlePage.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>Rules-A2-TOC.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>Rules-A3-LOF.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>Rules1-Intro.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>Rules2-Language.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>Rules3-Use.tedit
{ERIS}<Doc>Loops>Lyric>Beta>UserMods>RulesA-Convert.tedit

Conventions

This manual uses the following conventions:

- Case is significant in Xerox LOOPS and Lisp. All selectors, methods, arguments, etc., must be typed as shown. Typically, this means that method names are capitalized and variables are not.
- Arguments appear in italic type.
- Selectors, methods, functions, objects, classes, and instances appear in bold type.

For example, a method appears as follows:

(_ self **Selector** *Arg1 Arg2*)

- Examples appear in the following typeface:

89_ (_ LOGIN)

- Methods with an exclamation mark (!) suffix usually perform operations deeply into class structure instead of only on a given object.
- Methods with a question mark (?) suffix usually are predicates; that is, truth functions.
- Methods often appear in the form **ClassName.SelectorName**.
- Cautions describe possible dangers to hardware or software.
- Notes describe related text.

Style Sheet Addenda

Here are some guidelines I used when writing the LOOPS manuals. Items appear in rather random order.

- Avoid contractions.
- Avoid subscripts. Use WORD1 rather than WORD to avoid inconsistent line leading.
- Avoid wording that starts "Note that..." or "Notice that...". Either make it a note with correct format or eliminate the "Note that".
- Use semicolons rather than m-dashes.
- Each item in the template starts with an initial capital letter; e.g., "Describes..."
- The arguments are identical in the call and in the argument description.
- Parentheses appear around expressions and square brackets appear around the name of the functionality.
- The arrow in the expression is the NS character $\leftarrow$, not $\rightarrow$. These characters appear similarly when printed, but differently on the screen. See the section, "Special Notes and Cautions," for details.
- A period appears after the word None, after argument descriptions, and Returns: item.
- Items are set to or return T (instead of true).
- Menus contain options, not items or selections.
- You drag (not roll) the mouse to the right of a menu option to see its submenu.
- Use "above" and "below" when referring to things in the same section, section numbers and names when referring to things in the same chapter, and chapter numbers and names when referring to things in another chapter.
- Please study the following style sheet carefully before you start to edit. The various appearances of active value and annotated values are especially crazy making.

These things appear in **bold**:

class variables
functions
instance variables
messages
methods
variables

ActiveValue - specific class/instance
 active value - general information
 activeValue - previous implementation of **ActiveValue**

annotatedValue - data type
AnnotatedValue - specific class
 annotated values - general information

bitmap

data type

file package
 filecoms

inspector

Lisp Library package
localState - instance variable

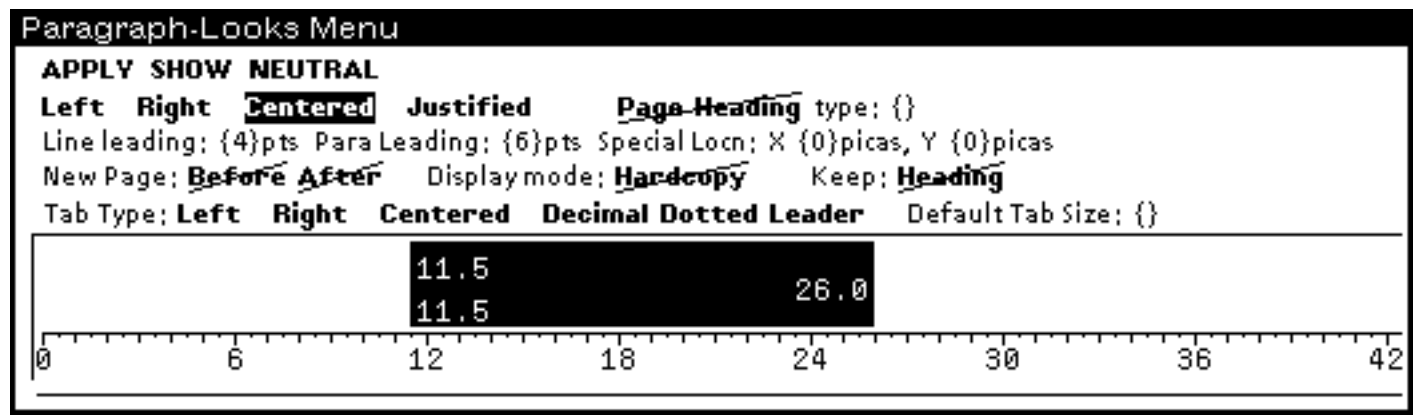
non-NIL

prettyprints

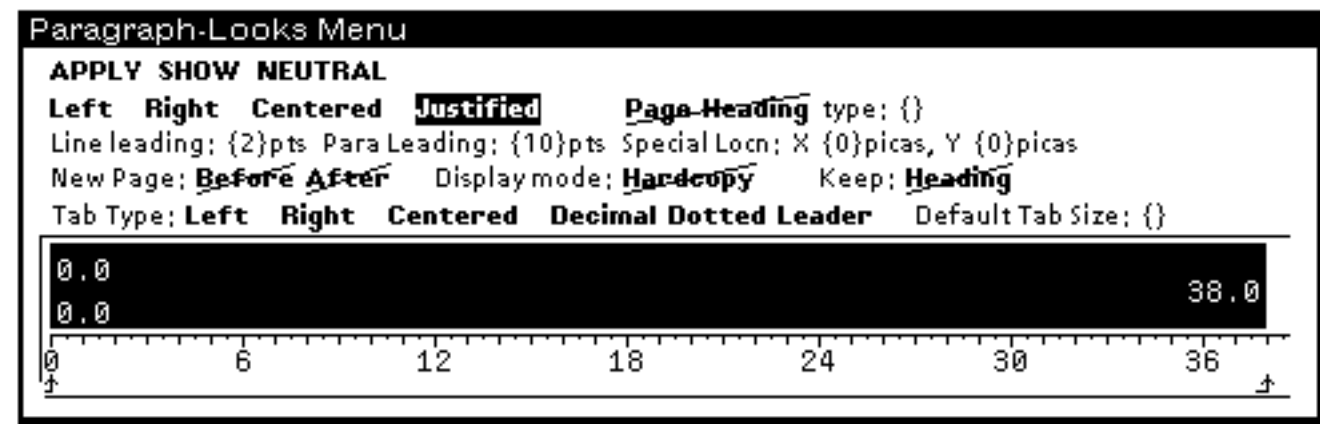
supers list

Paragraph Formatting

The heading has the following format:



The text the following format:



Page Layout

The starting page number varies with the package.

Page Layout Menu

APPLY SHOW

For page: **First(&Default)** **Other Left** **Other Right**

Starting Page #: {11} Paper Size: **Letter** **Legal** **A4 Landscape**

Page numbers: **No** **Yes** X: {26,5} Y: {3,0} Format: **123** **xiv** **XIV**

Alignment: **Left** **Centered** **Right**

Textbefore number: {} Textafter number: {}

Margins: Left {7,0} Right {6,0} Top {8,0} Bottom {8,0}

Columns: {1} Col Width: {38,0} Space between cols: {0,0}

Page Layout Menu

APPLY SHOW

For page: **First(&Default)** **Other Left** **Other Right**

Starting Page #: {} Paper Size: **Letter** **Legal** **A4 Landscape**

Page numbers: **No** **Yes** X: {26,5} Y: {3,0} Format: **123** **xiv** **XIV**

Alignment: **Left** **Centered** **Right**

Textbefore number: {} Textafter number: {}

Margins: Left {7,0} Right {6,0} Top {8,0} Bottom {8,0}

Columns: {1} Col Width: {38,0} Space between cols: {0,0}

Page Layout Menu

APPLY SHOW

For page: **First(&Default)** **Other Left** **Other Right**

Starting Page #: {} Paper Size: **Letter** **Legal** **A4 Landscape**

Page numbers: **No** **Yes** X: {26,5} Y: {3,0} Format: **123** **xiv** **XIV**

Alignment: **Left** **Centered** **Right**

Textbefore number: {} Textafter number: {}

Margins: Left {7,0} Right {6,0} Top {8,0} Bottom {8,0}

Columns: {1} Col Width: {38,0} Space between cols: {0,0}

Bitmaps, Graphs, and Sketches

Scale for bitmaps is 0.75.

Special Notes and Cautions

Make sure you have changed the underscore to be a left arrow before loading and printing any files. To do this,

- Enter the following commands into your Executive:

```
(GETCHARBITMAP (CHARCODE _) '(MODERN 10 MRR))
(EDITBM IT)
```

- When the bitmap editor appears, delete the underscore and insert the following left arrow:

```
.....
.....
.....
.....
.....
...X....
...XX...
..XXXXX.
...XX...
...X....
.....
.....
.....
.....
```

- Finally, enter the following commands into your Executive to store the pattern:

```
(PUTCHARBITMAP (CHARCODE _) '(MODERN 10 MRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(MODERN 10 BRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(TERMINAL 10 MRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(TERMINAL 10 BRR) IT)
(PUTCHARBITMAP (CHARCODE _) '(TERMINAL 12 BRR) IT)
```

CONVERT-LOOPS-FILES

By: Bob Bane (Bane.pa@Xerox.com)

5-Feb-88

INTRODUCTION

CONVERT-LOOPS-FILES allows you to convert files from the Koto release of Xerox LOOPS to the Lyric/Medley release. The changes from the Koto release are described in detail in the *Xerox LOOPS Release Notes*.

PROCEDURE

To Convert Koto LOOPS Files to Lyric/Medley Loops:

1. Your files must be from Koto LOOPS. Pre-Koto LOOPS files must first be run through the Buttress->Koto converter in a Koto LOOPS sysout.
2. Install and load your Lyric LOOPS sysout. You need to have TEdit loaded, as the converter uses it.
3. Load the file CONVERSION-AIDS.DFASL, which defines the function

```
(CONVERT-LOOPS-FILES <list-of-files> <dump-files-p>)
```

where:

<list-of-files> A filename or a list of filenames to be converted.

<dump-files-p> Determines disposition of files.

- If non-NIL, converted files will be dumped back out immediately.
- If :COMPILE, they will be compiled.
- If :COMPILE/LOAD, they will be compiled and the compiled code loaded.

4. Call CONVERT-LOOPS-FILES with the names of the files you want converted and an appropriate option.

Note the following:

- CONVERT-LOOPS-FILES makes more than one pass over the files being converted; the first pass is a TEdit textual change to make the files loadable into Lyric/Medley LOOPS. If you specify version numbers in your <list-of-files>, these changes will be made in place on your original files.
- The converter doesn't work completely automatically on systems consisting of several files that automatically load themselves with FILES coms; the converter may try to load subfiles before they are converted. It may be necessary to RETURN NIL from the LOAD calls in the break windows that will occur when this problem comes up. Aside from that, the converter works reasonably well, and has been used to convert large LOOPS systems with almost no source code changes from Koto Loops.

[This page intentionally left blank.]

LOOPSBACKWARDS

By: Bob Bane (Bane.pa@Xerox.com)

15-Dec-87

INTRODUCTION

LOOPSBACKWARDS allows you to run files which were previously converted from the Buttriss release of Xerox LOOPS to the Koto release. Unlike the Koto version of LOOPSBACKWARDS conversion methods for moving files from Buttriss LOOPS to Koto LOOPS are included but **not supported**. *We strong recommend that you convert LOOPS source code from Buttriss to Koto using the Koto release of LOOPS.* Conversion of Koto LOOPS source code to Lyric LOOPS is done using the CONVERSION-AIDS users' module.

The changes between the Buttriss and Koto releases are described in detail in the *Xerox LOOPS Release Notes* for the **Koto** release. Old features that are included in LOOPSBACKWARDS are summarized here:

- Many functions that were removed from the Buttriss release are defined.
- The messages **List** and **List!** are available.
- The operation of old style active values, from the Buttriss release, are provided.
- Support for reading old style macros, such as `$(localState getFn putFn)` or `#$Mumble`, is available.

LOOPSBACKWARDS is an unsupported LOOPS users module. It is strongly recommended that it only be used as part of an effort to upgrade very old LOOPS code to newer releases. It will allow very old LOOPS code to run well enough that it can be rewritten for a newer release.

INSTALLATION

LOOPSVCOPY will be automatically loaded by LOOPSBACKWARDS.

FUNCTIONS

LOOPSBACKWARDS includes **ExplicitFnActiveValue** and **DefAVP**. **ExplicitFnActiveValue** allows the user code triggered by Get- or Put- accesses to be stored within functions which are pointed to by instance variables rather than requiring the redefinition of **GetWrappedValue** or **PutWrappedValue**. These functions must have the form specified in the **DefAVP** function.

ExplicitFnActiveValue

[Class]

Purpose: Mimics the behavior of the Buttruss style of active values.

Behavior: Get- accesses to the wrapped variable cause the **getFn** to be called, Put- accesses cause **putFn** to be called. Enables the old style activeValue to look like the new style without changing any functionality.

The **getFn** is called by the **ExplicitFnActiveValue GetWrappedValue** method. This method passes to the **getFn** the arguments defined by **DefAVP** as described in the *Xerox LOOPS Users' Modules*.

The **putFn** is called by the **ExplicitFnActiveValue PutWrappedValue** method. This method passes to the **putFn** the arguments defined by **DefAVP** as described in the *Xerox LOOPS Users' Modules*.

Instance Variables: **localState** A place for data storage.

getFn The name of a function applied when the active variable is read.

putFn The name of a function applied when the active variable is changed.

Example:

```

32_(_ ($ Bin) New 'bin4)
#,($& Bin (|DAW0.1Y:.H53.]99| . 521))

33_(_ ($ ExplicitFnActiveValue) New 'EFAV1)
#,($& ExplicitFnActiveValue (|DAW0.1Y:.H53.]99| . 522))

34_(DEFINEQ (PrintOnGet
  (self varName localSt propName activeVal type)
  (PRINTOUT T "I am:" , activeVal T) localSt))
(PrintOnGet)

35_(@ ($ EFAV1) getFn 'PrintOnGet)
PrintOnGet

36_(_ ($ EFAV1) AddActiveValue ($ bin4) 'height)
#,($A #,NestedNotSetValue PrintOnGet NIL)

37_(@ ($ bin4) height 123)
123

38_(@ ($ bin4) height)

```

```
I am: #, ($ EFAV1)
123
```

(DefAVP *fnName putFlg*)

[Function]

Purpose: Creates a template for defining an active value function.

Behavior: Creates a template and leaves you in the Interlisp function display editor.

Arguments: *fnName* Name of the function.
putFlg T indicates function is a **putFn**, NIL indicates a **getFn**.

Returns: The function name on exit from the editor .

Example: In each of the following cases the template only is shown. User code is to be added immediately after the comment by using the display editor.

```
66_ (DefAVP 'AGetFn)
AGetFn
```

```
67_PP* AGetFn
(AGetFn
 [LAMBDA (self varName localSt propName activeVal type)
  (* This is a getFn. The value of this getFn is
 returned as the value of the enclosing GetValue.)
  localSt])
(AGetFn)
```

```
68_ (DefAVP 'APutFn T)
APutFn
```

```
69_PP* APutFn
(APutFn
 [LAMBDA (self varName newValue propName activeVal type)
  (* This is a putFn. ***NOTE*** The value of this
 function will be returned as the value of any enclosing
 PutValue. This usually means that you want to return the value
 returned by PutLocalState.)
  (PutLocalState activeVal newValue self varName propName
 type)])
(APutFn)
```

[This page intentionally left blank.]

LOOPSMIXIN

By: Bob Bane (Bane.pa@Xerox.com)

15-Dec-87

INTRODUCTION

LOOPSMIXIN defines several small classes that can be mixed into your application classes. It also defines the class **Perspective** and its support classes, which are used in the implementation of Xerox LOOPS Rules. Perspectives allow you to view one object as having more than one class at a time; they have not been tested extensively outside of Rules and are known to have major bugs which don't affect Rules, so they are not documented here.

LOOPSMIXIN Classes

DatedObject - Defines the instance variables **created** and **creator** which are set at object creation time to the values of (DATE) and (USERNAME).

NamedObject - Defines the instance variable **name** and initializes it to an ActiveValue that insures that the Xerox LOOPS system name for the containing object is uniquely name; i.e. storing a name for the object in **name** causes the previous name for the object to be removed with **DeleteObjectName** and the new name for the object to be asserted with **NameEntity**.

GlobalNamedObject - A subclass of **NamedObject** that works the same way as **NamedObject**.

ListMetaClass - Specializes the **New** and **DestroyInstance** methods to keep a list of all instances of that class in the class property **AllInstances** for that class.

StrucMeta - A MetaClass useful for creating new classes. Specializes the **New** method to create a Class object by copying the instance variable and class variable descriptions of the current class. Class variables with a non-NIL Local property will not be copied.

TempClass - Specializes the **New** method to always create objects of this class using the **NewTemp** method, insuring that the objects will be temporary objects.

Perspective, Node, Template, TextItem - These classes are used in Xerox LOOPS Rules.

[This page intentionally left blank.]

[This page intentionally left blank.]

[This page intentionally left blank.]

Writer's Notes -- Production Details

This file includes notes on the production of *Xerox LOOPS Users' Modules Manual*, Lyric Beta Release. This manual is packaged in one binder.

Writer: Raven Kontur Brewster

Printing Date: 22 February 1988

Files Needed

To edit or print the manual, make sure you have the following files loaded:

IMTOOLS
SKETCH
GRAPHER

Fonts Used

{ERIS}<LISP>FONTS>

Modern font
18-point bold
14-point bold
12-point bold
10-point regular
10-point italic
10-point bold

Terminal font
10-point regular
10-point italic
10-point bold

Printing Information

The manual was printed under a Lyric sysout on the Tsunami printer.

1. INTRODUCTION TO RULE-ORIENTED PROGRAMMING IN XEROX LOOPS

The core of decision-making expertise in many kinds of problem solving can be expressed succinctly in terms of rules. The following sections describe facilities in Xerox LOOPS for representing rules, and for organizing knowledge-based systems with rule-oriented programming. The Xerox LOOPS rule language provides an experimental framework for developing knowledge-based systems. The rule language and programming environment are integrated with the object-oriented, data-oriented, and procedure-oriented parts of Xerox LOOPS.

Rules in Xerox LOOPS are organized into production systems (called RuleSets) with specified control structures for selecting and executing the rules. The work space for RuleSets is an arbitrary Xerox LOOPS object.

Decision knowledge can be factored from control knowledge to enhance the perspicuity of rules. The rule language separates decision knowledge from meta-knowledge such as control information, rule descriptions, debugging instructions, and audit trail descriptions. An audit trail records inferential support in terms of the rules and data that were used. Such trails are important for knowledge-based systems that must be able to account for their results. They are also essential for guiding belief revision in programs that need to reason with incomplete information.

1.1 Introduction

Production rules have been used in expert systems to represent decision-making knowledge for many kinds of problem-solving. Such rules (also called *if-then* rules) specify actions to be taken when certain conditions are satisfied. Several rule languages have been developed in the past few years and used for building expert systems. The following sections describe the concepts and facilities for rule-oriented programming in Xerox LOOPS.

Xerox LOOPS has the following major features for rule-oriented programming:

- (1) Rules in Xerox LOOPS are organized into ordered sets of rules (called RuleSets) with specified control structures for selecting and executing the rules. Like subroutines, RuleSets are building blocks for organizing programs hierarchically.
- (2) The work space for rules in Xerox LOOPS is an arbitrary Xerox LOOPS object. The names of the instance variables provide a name space for variables in the rules.
- (3) Rule-oriented programming is integrated with object-oriented, data-oriented, and procedure-oriented programming in Xerox LOOPS.
- (4) RuleSets can be invoked in several ways: In the object-oriented paradigm, they can be invoked as methods by sending messages to objects. In the data-oriented paradigm, they can be invoked

as a side-effect of fetching or storing data in active values. They can also be invoked directly from Lisp programs. This integration makes it convenient to use the other paradigms to organize the interactions between RuleSets.

- (5) RuleSets can also be invoked from rules either as predicates on the LHS of rules, or as actions on the RHS of rules. This provides a way for RuleSets to control the execution of other RuleSets.
- (6) Rules can automatically leave an audit trail. An audit trail is a record of inferential support in terms of rules and data that were used. Such trails are important for programs that must be able to account for their results. They can also be used to guide belief revision in programs that must reason with incomplete information.
- (7) Decision knowledge can be separated from control knowledge to enhance the perspicuity of rules. The rule language separates decision knowledge from meta-knowledge such as control information, rule descriptions, debugging instructions, and audit trail descriptions.
- (8) The rule language provides a concise syntax for the most common operations.
- (9) There is a fast and efficient compiler for translating RuleSets into Interlisp functions.
- (10) Xerox LOOPS provides facilities for debugging rule-oriented programs.

The following sections are organized as follows: Section 1.2, "Basic Concepts," outlines the basic concepts of rule-oriented programming in Xerox LOOPS. It contains many examples that illustrate techniques of rule-oriented programming. Section 1.3, "Organizing a Rule-Oriented Program," describes the rule syntax, and the remaining sections in this chapter discuss the facilities for creating, editing, and debugging RuleSets in Xerox LOOPS.

1.2 Basic Concepts

Rules express the conditional execution of actions. They are important in programming because they can capture the core of decision-making for many kinds of problem-solving. Rule-oriented programming in Xerox LOOPS is intended for applications to expert and knowledge-based systems.

The following sections outline some of the main concepts of rule-oriented programming. Xerox LOOPS provides a special language for rules because of their central role, and because special facilities can be associated with rules that are impractical for procedural programming languages. For example, Xerox LOOPS can save specialized audit trails of rule execution. Audit trails are important in knowledge systems that need to explain their conclusions in terms of the knowledge used in solving a problem. This capability is essential in the development of large knowledge-intensive systems, where a long and sustained effort is required to create and validate knowledge bases. Audit trails are also important for programs that do non-monotonic reasoning. Such programs must work with incomplete information, and must be able to revise their conclusions in response to new information.

1.3 Organizing a Rule-Oriented Program

In any programming paradigm, it is important to have an organizational scheme for composing large systems from smaller ones. Stated differently, it is important to have a method for partitioning large programs into nearly-independent and manageably-sized pieces. In the procedure-oriented paradigm, programs are decomposed into procedures. In the object-oriented paradigm, programs are decomposed into objects. In the rule-oriented paradigm, programs are decomposed into *RuleSets*. A Xerox LOOPS program that uses more than one programming paradigm is factored across several of these dimensions.

There are three approaches to organizing the invocation of RuleSets in Xerox LOOPS:

Procedure-oriented Approach. This approach is analogous to the use of subroutines in procedure-oriented programming. Programs are decomposed into RuleSets that call each other and return values when they are finished. *SubRuleSets* can be invoked from multiple places. They are used to simplify the expression in rules of complex predicates, generators, and actions.

Object-oriented Approach. In this approach, RuleSets are installed as methods for objects. They are invoked as methods when messages are sent to the objects. The method RuleSets are viewed analogously to other procedures that implement object message protocols. The value computed by the RuleSet is returned as the value of the message sending operation.

Data-oriented Approach. In this approach, RuleSets are installed as access functions in active values. A RuleSet in an active value is invoked when a program gets or puts a value in the Xerox LOOPS object. As with active values with Lisp functions for the *getFn* or *putFn*, these RuleSet active values can be triggered by any Xerox LOOPS program, whether rule-oriented or not.

These approaches for organizing RuleSets can be combined to control the interactions between bodies of decision-making knowledge expressed in rules. For example, Figure 1 shows the RuleSet of consumer instructions for testing a washing machine. The work space for the ruleSet is a Xerox LOOPS object of the class **WashingMachine**. The control structure *While1* loops through the rules trying an escalating sequence of actions, starting again at the beginning of some rule is applied. Some rules, called one-shot rules, are executed at most once. These rules are indicated by preceding them with a one in braces ({1}).


```

RuleSet Name: CheckWashingMachine;
Workspace Class: WashingMachine;
Control Structure: while1;
While Condition: ruleApplied;

(* What a consumer should do when a washing machine failes.)

  IF .Operational THEN (STOP T);

  IF load>1.0 THEN .ReduceLoad;

  If ~pluggedInTo THEN .PlugIn;

{1} IF pluggedInTo:voltage=0 THEN breaker.Reset;
{1} IF pluggedInTo:voltage<110 THEN SPGE.Call;
{1} THEN dealer.RequestService;
{1} THEN manufacturer.Complain;
{1} THEN $ConsumerBoard.Complain;
{1} THEN (STOP T);

```

Figure 1. Basic RuleSet

1.4 Control Structures for Selecting Rules

RuleSets in Xerox LOOPS consist of an ordered list of rules and a control structure. Together with the contents of the rules and the data, a RuleSet control structure determines which rules are executed. Execution is determined by the contents of rules in that the conditions of a rule must be satisfied for it to be executed. Execution is also controlled by data in that different values in the data allow different rules to be satisfied. Criteria for iteration and rule selection are specified by a RuleSet control structure. There are two primitive control structures for RuleSets in Xerox LOOPS which operate as follows:

Do1

[RuleSet Control Structure]

The first rule in the RuleSet whose conditions are satisfied is executed. The value of the RuleSet is the value of the rule. If no rule is executed, the RuleSet returns **NIL**.

The **Do1** control structure is useful for specifying a set of mutually exclusive actions, since at most one rule in the RuleSet will be executed for a given invocation. When a RuleSet contains rules for specific and general situations, the specific rules should be placed before the general rules.

DoAll

[RuleSet Control Structure]

Starting at the beginning of the RuleSet, every rule is executed whose conditions are satisfied. The value of the RuleSet is the value of the last rule executed. If no rule is executed, the RuleSet returns **NIL**.

The **DoAll** control structure is useful when a variable number of additive actions are to be carried out, depending on which conditions are satisfied. In a single invocation of the RuleSet, all of the applicable rules are invoked.

Figure 2 illustrates the use of a **Do1** control structure to select one of three mutually exclusive actions.

```
RuleSet Name: SimulateWashingMachine;
Workspace Class: WashingMachine;
Control Structure: Do1 ;

(* Rules for controlling the wash cycle of a washing machine.)

IF controlSetting = 'RegularFabric
THEN .Fill .Wash .Pause .SpinAndDrain
    .SprayAndRinse .SpinAndDrain
    .Fill. DeepRinse .Pause .DampDry;

IF controlSetting = 'PermanentPress
THEN .Fill .Wash .Pause .SpinAndPartialDrain
    .FillCold .SpinAndPartialDrain
    .FillCold .Pause .SpinAndDrain
    .FillCold. DeepRinse .Pause .DampDry;

IF controlSetting = 'DelicateFabric
THEN .FillSoak1 .Agitate .Soak4 .Agitate
    .Soak1 .SpinAndDrain .SprayAndRinse
    .SpinAndDrain .Fill .DeepRinse .Pause .DampDry;
```

Figure 2. RuleSet showing Do1

There are two control structures in Xerox LOOPS that specify iteration in the execution of a RuleSet. These control structures use an explicit while-condition associated with the RuleSet. They are direct extensions of the two primitive control structures above.

While1

[RuleSet Control Structure]

This is a cyclic version of **Do1**. If the while-condition is satisfied, the first rule is executed whose conditions are satisfied. This is repeated as long as the while condition is satisfied or until a **Stop** statement or transfer call is executed (see Section 2.14, "Stop Statements"). The value of the RuleSet is the value of the last rule that was executed, or **NIL** if no rule was executed.

WhileAll

[RuleSet Control Structure]

This is a cyclic version of **DoAll**. If the while-condition is satisfied, every rule is executed whose conditions are satisfied. This is repeated as long as the while condition is satisfied or until a **Stop** statement is executed. The value of the RuleSet is the value of the last rule that was executed, or **NIL** if no rule was executed.

The "while-condition" is specified in terms of the variables and constants accessible from the RuleSet. The constant **T** can be used to specify a RuleSet that iterates forever (or until a **Stop** statement or transfer is executed). The special variable **ruleApplied** is used to specify a RuleSet that continues as long as some rule was executed in the last iteration. Figure 3 illustrates a simple use of the **WhileAll** control structure to specify a sensing/acting feedback loop for controlling the filling of a washing machine tub with water.

```

RuleSet Name: FillTub;
Workspace Class: WashingMachine;
Control Structure: WhileAll ;
Temp Vars: waterLimit;
WhileCond: T;

(* Rules for controlling the filling of a washing tub with
water.)

{1!} IF loadSetting = 'Small THEN waterLimit_10;
{1!} IF loadSetting = 'Meduim THEN waterLimit_13.5;
{1!} IF loadSetting = 'Large THEN waterLimit_17;
{1!} IF loadSetting = 'ExtraLarge THEN waterLimit_20;

(* Respond to a change of temperature setting at any time.)

IF termperatureSetting = 'Hot
THEN HotWaterValve.Open ColdWaterValve.Close;

IF termperatureSetting = 'Warm
THEN HotWaterValve.Open ColdWaterValve.Open;

IF termperatureSetting = 'Cold
THEN HotWaterValve.Close ColdWaterValve.Open;

(* Stop when the water reaches its limit.)

IF waterLevelSensor.Test >= waterLimit
THEN HotWaterValve.Close ColdWaterValve.Close
(Stop T);

```

Figure 3. RuleSet with WhileAll

There are two control structures in Xerox LOOPS that specify iteration over a set of elements in the execution of a RuleSet. These control structures use an explicit while-condition associated with the RuleSet. They are direct extensions of the two primitive control structures above.

FOR1

[RuleSet Control Structure]

This is a cyclic version of **Do1**. If the iteration-condition (or while-condition) is satisfied, the first rule is executed whose conditions are satisfied or until a **Stop** statement is executed. This is repeated as long as the iteration condition is satisfied. The value of the RuleSet is the value of the last rule that was executed, or **NIL** if no rule was executed.

FORALL

[RuleSet Control Structure]

This is a cyclic version of **DoAll**. If the iteration-condition is satisfied, every rule is executed whose conditions are satisfied. This is repeated as long as the iteration condition is satisfied or until a **Stop** statement is executed. The value of the RuleSet is the value of the last rule that was executed, or **NIL** if no rule was executed.

The "iteration-condition" is specified in terms of the variables and constants accessible from the RuleSet. The simplest condition is

(FOR <iterVar> IN <setExpr> DO ruleSet) ;

The **setExpr** will be parsed with the RuleSet parser. The symbol **ruleSet** is a reserved word, and must be spelled as shown (no changes in capitalization).

Here is an example of iteration:

Control Structure: FORALL;

Iteration Condition: (FOR buyer IN (RoadStops (\$ Consumer)) DO ruleSet) ;

For each buyer in the list produced by RoadStops, the ruleSet will be run. In a **FOR1**, the iteration will go on to the next buyer as soon as one rule executes. In a **FORALL**, all rules in the RuleSet will be tried.

For nested iteration one can use a slightly more complicated form, as illustrated by the following example:

**Iteration Condition: (FOR buyer IN (RoadStops (\$ Consumer)) DO
(FOR seller in (RoadStops (\$ Producer)) DO ruleSet)) ;**

An experienced Lisp user can see that this resembles the CLISP iteration construct. In fact, except that you can (must) use the RuleSet syntax in the construct, it is the CLISP construct, and any such construct can be used. A DO1 or DOALL ruleSet will be substituted for the occurrence of the atom ruleSet, depending on whether the Control Structure is a FOR1 or FORALL.

As an abbreviation, if the construct does not contain the atom ruleSet, then (DO ruleSet) is appended to the Iteration Condition for a **FOR1** or **FORALL**. Thus one could write the first example as:

Iteration Condition: (FOR buyer IN (RoadStops (\$ Consumer)))

1.5 One-Shot Rules

One of the design objectives of Xerox LOOPS is to clarify the rules by factoring out control information whenever possible. This objective is met in part by the declaration of a control structure for RuleSets.

Another important case arises in cyclic control structures in which some of the rules should be executed only once. This was illustrated in the Washing Machine example in Figure 1 where we wanted to prevent the RuleSet from going into an infinite loop of resetting the breaker, when there was a short circuit in the Washing Machine. Such rules are also useful for initializing data for RuleSets as in the example in Figure 3.

In the absence of special syntax, it would be possible to encode the information that a rule is to be executed only once as follows:

Control Structure: While1
Temporary Vars: triedRule3;

...

IF ~triedRule3 condition condition THEN triedRule3_T action ;

In this example, the variable **triedRule3** is used to control the rule so that it will be executed at most once in an invocation of a RuleSet. However, the prolific use of rules with such control clauses in large systems has led to the common complaint that control clauses in rule languages defeat the expressiveness and conciseness of the rules. For the case above, Xerox LOOPS provides a shorthand notation as follows:

{1} IF condition condition THEN action ;

The brace notation means exactly the same thing in the example above, but it more concisely and clearly indicates that the rule executes only once. These rules are called "one shot" or "execute-once" rules.

In some cases, it is desired not only that a rule be executed at most once, but that it be tested at most once. This corresponds to the following:

Control Structure: While1
Temporary Vars: triedRule3;

...

IF ~triedRule3 triedRule3_T condition condition THEN action ;

In this case, the rule will not be tried more than once even if some of the conditions fail the first time that it is tested. The Xerox LOOPS shorthand for these rules (pronounced "one shot bang") is

{1!} **IF** *condition* *condition* **THEN** *action* ;

These rules are called "try-once" rules.

The two kinds of one-shot rules are our first examples of the use of meta-descriptions preceding the rule body in braces. See Section 1.7, "Saving an Audit Trail of Rule Invocation," for information on using meta-descriptions for describing the creation of audit trails.

1.6 First/Last Rules

It is sometimes useful to have rules which fire before or after the ordinary part of the RuleSet is invoked, independent of the form of the control structure. For example, in a DO1, such "FIRST " rules could be used for initialization. These now exist, and are notated by putting a {F} for a first rule in the MetaDescription field, and a {L} for a last rule. If a RuleSet has L rules which execute, the value of the RuleSet is the value of the last rule which executed.

1.7 Saving an Audit Trail of Rule Invocation

A basic property of knowledge-based systems is that they use knowledge to infer new facts from older ones. (Here we use the word "facts" as a neutral term, meaning any information derived or given, that is used by a reasoning system.) Over the past few years, it has become evident that reasoning systems need to keep track not only of their conclusions, but also of their reasoning steps. Consequently, the design of such systems has become an active research area in AI. The audit trail facilities of Xerox LOOPS support experimentation with systems that can not only use rules to make inferences, but also keep records of the inferential process itself.

1.7.1 Motivations and Applications

Debugging. In most expert systems, knowledge bases are developed over time and are the major investment. This places a premium on the use of tools and methods for identifying and correcting bugs in knowledge bases. By connecting a system's conclusions with the knowledge that it uses to derive them, audit trails can provide a substantial debugging aid. Audit trails provide a focused means of identifying potentially errorful knowledge in a problem solving context.

Explanation Facilities. Expert systems are often intended for use by people other than their creators, or by a group of people *pooling* their knowledge. An important consideration in validating expert systems is that reasoning should be *transparent*, that is, that a system should be able to give an account of its reasoning process. Facilities for doing this are sometimes called *explanation systems*.

and the creation of powerful explanation systems is an active research area in AI and cognitive science. The audit trail mechanism provides an essential computational prerequisite for building such systems.

Belief Revision. Another active research area is the development of systems that can "change their minds". This characteristic is critical for systems that must reason from incomplete or errorful information. Such systems get leverage from their ability to make assumptions, and then to recover from bad assumptions by efficiently reorganizing their beliefs as new information is obtained. Research in this area ranges from work on non-monotonic logics, to a variety of approaches to belief revision. The facilities in the rule language make it convenient to use a user-defined calculus of belief revision, at whatever level of abstraction is appropriate for an application.

1.7.2 Overview of Audit Trail Implementation

When *audit mode* is specified for a RuleSet, the compilation of assignment statements on the right-hand sides of rules is altered so that audit records are created as a side-effect of the assignment of values to instance variables. Audit records are Xerox LOOPS objects, whose class is specified in RuleSet declarations. The audit records are connected with associated instance variables through the value of the **reason** properties of the variables.

Audit descriptions can be associated with a RuleSet as a whole, or with specific rules. Rule-specific audit information is specified in a property-list format in the meta-description associated with a rule. For example, this can include *certainty factor* information, categories of inference, or categories of support. Rule-specific information overrides RuleSet information.

During rule execution in audit mode, the audit information is evaluated after the rule's LHS has been satisfied and before the rule's RHS is applied. For each rule applied, a single audit record is created and then the audit information from the property list in the rule's meta-description is put into the corresponding instance variables of the audit record. The audit record is then linked to each of the instance variables that have been set on the RHS of the rule by way of the **reason** property of the instance variable.

Additional computations can be triggered by associating active values with either the audit record class or with the instance variables. For example, active values can be specified in the audit record classes in order to define a uniform set of side-effects for rules of the same category. In the following example, such an active value is used to carry out a "certainty factor" calculation.

1.7.3 An Example of Using Audit Trails

The following example illustrates one way to use the audit trail facilities. Figure 4 illustrates a RuleSet which is intended to capture the decisions for evaluating the potential purchase of a washing machine. As with any purchasing situation, this one includes the difficulty of incomplete information about the product. For example in this RuleSet, the reliability of the washing machine is estimated to be 0.5 in the absence of specific information from *Consumer Reports*. The meta-descriptions for the rules, which appear in braces, categorize them in terms of the *basis of belief* (the category *basis* is either a fact or estimate) and a *certainty factor* (*cf*) that is supposed to measure the "implication power" of the

rule. Within the braces, the variable on the left of the assignment statement is always interpreted as meaning a variable in the audit record, and the variables on the right are always interpreted as variables accessible within the RuleSet. This makes it straightforward to experiment with user-defined audit trails and experimental methods of belief revision. (Realistic belief revision systems are usually more sophisticated than this example.)

The result of running the RuleSet is an evaluation report for each candidate machine. Since the RuleSet was run in audit mode, each entry in the evaluation report is tagged with a reason that points to an audit record. Figure 5 illustrates the evaluation report for one machine and one of its audit records. In this example, each of the entries in the report has a reason and a cumulative certainty (cc) property in addition to the value. The value of the reason properties are audit records created as a side effect of running the RuleSet. The auditing process records the meta-description information of each rule in its audit record. This information can be used later for generating explanations or as a basis for belief revision. The auditing process can have side effects. For example, the active in the **cf** variable or the audit record performs a computation to maintain a calculated cumulative certainty in the reliability variable of the evaluation report.

The meta-descriptions for **basis** and **cf** are saved directly in the audit record. The *certainty factor* calculation in this combines information from the audit description with other information already associated with the object. To do this, the **cf** description triggers an active value inherited by the audit record from its class. This active value computes a *cumulative certainty* in the evaluation report. (Other variations on this idea would include certainty information descriptive of the premises of the rule.)

```
RuleSet Name: EvaluateWashingMachine;
Workspace Class: EvaluationReport;
Control Structure: doAll ;
Audit Class: CFAuditRecord ;
Compiler Options: A;

(* Rules for evaluating a potential washing machine for a
purchase.)

.
.
.
{ (basics_Fact cf_1) }
IF buyer:familySize>2 machine:capacity<20
THEN suitability_'Poor;

{ (basics_Fact cf_.8) }
reliability_(_($ ConsumerReports) GetFacts machine);

{ (basics_Estimate cf_.4) }
IF 'reliability THEN reliability_.5;
.
.
.
```

Figure 4. RuleSet Showing Evaluation

```
EvaluationReport "uid1"
expense: 510
```



```

suitability:  Poor cc 1 reason ...
reliability:  .5 cc .6 reason "uid2"
.
.
.
AuditRec "uid2"
rule:  "uid3"
basis:  Estimate;
cf:  #(.4 NIL PutCumulativeCertainty)

```

Figure 5. Example of an Audit Trail

1.8 Comparison with Other Rule Languages

This section considers the rationale behind the design of the Xerox LOOPS rule language, focusing on ways that it diverges from other rule languages. In general, this divergence was driven by the following observation:

When a rule is heavy with control information, it obscures the domain knowledge that the rule is intended to convey.

Rules are harder to create, understand, and modify when they contain too much control information. This observation led us to find ways to factor control information out of the rules.

1.8.1 The Rationale for Factoring Meta-Level Syntax

One of the most striking features of the syntax of the Xerox LOOPS rule language is the factored syntax for meta-descriptions, which provides information about the rules themselves. Traditional rule languages only factor rules into conditions on the left hand side (LHS) and actions on the right hand side (RHS), without general provisions for meta-descriptions.

Decision knowledge expressed in rules is most perspicuous when it is not mixed with other kinds knowledge, such as control knowledge. For example, the following rule:

```

IF ~triedRule4 pluggedInTo:voltage=0
THEN triedRule4_T breaker.Reset;

```

is more obscure than the corresponding one-shot rule from Figure 1:

```

{1} IF pluggedInTo:voltage=0 THEN breaker.Reset;

```

which factors the control information (that the rule is to be applied at most once) from the domain knowledge (about voltages and breakers). In the Xerox LOOPS rule language, a meta-description (MD) is specified in braces in front of the LHS of a rule. For another example, the following rule from Figure 4:

```
{{(basis_Fact cf_8)}  
IF buyer:familySize>2 machine:capacity<20  
THEN suitability_ 'Poor;
```

uses an MD to indicate that the rule has a particular **cf** ("certainty factor") and **basis** category for belief support. The MD in this example factors the description of the inference category of the rule from the action knowledge in the rule.

In a large knowledge-based system, a substantial amount of control information must be specified in order to preclude combinatorial explosions. Since earlier rule languages fail to provide a means for factoring meta-information, they must either mix it with the domain knowledge or express it outside the rule language. In the first option, intelligibility is degraded. In the second option, the transparency of the system is degraded because the knowledge is hidden.

1.8.2 The Rationale for RuleSet Hierarchy

Some advocates of production systems have praised the flatness of traditional production systems, and have resisted the imposition of any organization to the rules. The flat organization is sometimes touted as making it *easy to add rules*. The argument is that other organizations diminish the power of pattern-directed invocation and make it more complicated to add a rule.

In designing Xerox LOOPS, we have tended to discount these arguments. We observe that there is no inherent property of production systems that can make rules additive. Rather, *additivity* is a consequence of the independence of particular sets of rules. Such independence is seldom achieved in large sets of rules. When rules are dependent, rule invocation needs to be carefully ordered.

Advocates of a flat organization tend to organize large programs as a single very large production system. In practice, most builders of production systems have found it essential to create groups of rules.

Grouping of rules in flat systems can be achieved in part by using *context* clauses in the rules. Context clauses are clauses inserted into the rules which are used to alter the flow of control by naming the context explicitly. Rules in the same "context" all contain an extra clause in their conditions that compares the context of the rules with a current context. Other rules redirect control by switching the current context. Unfortunately, this approach does not conveniently lend itself to the reuse of groups of rules by different parts of a program. Although context clauses admit the creation of "subroutine contexts", they require you to explicitly program a stack of return locations in cases where contexts are invoked from more than one place. The decision to use an implicit calling stack for RuleSet invocation in Xerox LOOPS is another example of the our desire to simplify the rules by factoring out control information.

1.8.3 The Rationale for RuleSet Control Structures

Production languages are sometimes described as having a *recognize-act cycle*, which specifies how rules are selected for execution. An important part of this cycle is the *conflict resolution strategy*, which specifies how to choose a production rule when several rules have conditions that are satisfied. For

example, the **OPS5** production language has a conflict resolution strategy (**MEA**) which prevents rules from being invoked more than once, prioritizes rules according to the recency of a change to the data, and gives preference to production rules with the most specific conditions.

In designing the rule language for Xerox LOOPS, we have favored the use of a small number of specialized control structures to the use of a single complex conflict resolution strategy. In so doing, we have drawn on some control structures in common use in familiar programming languages. For example, **Do1** is like Lisp's **COND**, **DoAll** is like Lisp's **PROG**, **WhileAll** is similar to **WHILE** statements in many programming languages.

The specialized control structures are intended for concisely representing programs with different control relationships among the rules. For example, the **DoAll** control structure is useful for rules whose effects are intended to be additive and the **Do1** control structure is appropriate for specifying mutually exclusive actions. Without some kind of iterative control structure that allows rules to be executed more than once, it would be impossible to write a simulation program such as the washing machine simulation in Figure 1.

We have resisted a reductionist argument for having only one control structure for all programming. For example, it could be argued that the control structure **Do1** is not strictly necessary because any RuleSet that uses **Do1** could be rewritten using **DoAll**. For example, the rules

Control Structure: Do1;

```
IF a b c THEN d e ;
IF a b c THEN d e ;
IF a b c THEN d e ;
```

could be written alternatively as

Control Structure: DoAll;
Task Vars: firedSomeRule;

```
IF a b c THEN firedSomeRule_T d e ;
IF ~firedSomeRule a b c THEN firedSomeRule_T d e ;
IF ~firedSomeRule a b c THEN firedSomeRule_T d e ;
```

However, the **Do1** control structure admits a much more concise expression of mutually exclusive actions. In the example above, the **Do1** control structure makes it possible to abbreviate the rule conditions to reflect the assumption that earlier rules in the RuleSet were not satisfied.

For some particular sets of rules the conditions are naturally mutually exclusive. Even for these rules **Do1** can yield additional conciseness. For example, the rules:

Control Structure: Do1;

```
IF a b c THEN d e ;
IF ~a b c THEN d e ;
IF ~a ~b c THEN d e ;
```

can be written as

Control Structure: Do1;

```
IF a b c THEN d e ;  
IF  b c THEN d e ;  
IF   c THEN d e ;
```

Similarly it could be argued that the **Do1** and **DoAll** control structures are not strictly necessary because such RuleSets can always be written in terms of **While1** and **WhileAll**. Following this reductionism to its end, we can observe that every RuleSet could be re-written in terms of **WhileAll**.

1.8.4 The Rationale for an Integrated Programming Environment

RuleSets in Xerox LOOPS are integrated with procedure-oriented, object-oriented, and data-oriented programming paradigms. In contrast to single-paradigm rule systems, this integration has two major benefits. It facilitates the construction of programs which don't entirely fit the rule-oriented paradigm. Rule-oriented programming can be used selectively for representing just the appropriate decision-making knowledge in a large program. Integration also makes it convenient to use the other paradigms to help organize the interactions between RuleSets.

Using the object-oriented paradigm, RuleSets can be invoked as methods for Xerox LOOPS objects. Figure 6 illustrates the installation of the RuleSet **SimulateWashingMachineRules** to carry out the **Simulate** method for instances of the class **WashingMachine**. This definition of the class **WashingMachine** specifies that Lisp functions are to be invoked for Fill and Wash messages. For example, the Lisp function **WashingMachine.Fill** is to be applied when a Fill message is received. When a Simulate message is received, the RuleSet **SimulateWashingMachineRules** is to be invoked with the washing machine as its work space. Simulate message to invoke the RuleSet may be sent by any Xerox LOOPS program, including other RuleSets.

The use of object-oriented paradigm is facilitated by special RuleSet syntax for sending messages to objects, and for manipulating the data in Xerox LOOPS objects. In addition, RuleSets, work spaces, and tasks are implemented as Xerox LOOPS objects.

```

[DEFCLASS WashingMachine
  (MetaClass Class Edited (* "rtk: 12-Jun-87 07:57")
    doc (* Home appliance for wachine clooths.))
  (Supers ElectricalDevice PlumbedDevice CleaningDevice)
  (ClassVariables)
  (InstanceVariables
    (controlSetting Meduim
      doc (* One of Small, Medium, Large, ExtraLarge))...)
  (Methods
    (Fill WashingMachine.Fill doc (* Fill the tub with water.))
    (Wash WashingMachine.Wash doc (* Perofrm the wash cycle.))
    (Simulate UseRuleSet RuleSet SimulateWashingMachineRules)
    .
    .
    .
  ]

```

Figure 6. RuleSet Invoked as a Method

Using the data-oriented paradigm, RuleSets can be installed in active values so that they are triggered by side-effect when Xerox LOOPS programs get or put data in objects. For example:

```

[DEFINST WashingMachine (StefiksMaytagWasher "uid2")
  (controlSetting RegularFabric)
  (loadSetting #(Medium NIL RSPut) RSPutFn CheckOverLoadRules)
  (waterLevelSensor "uid3")
]

```

The above code illustrates a RuleSet named **CheckOverLoadRules** which is triggered whenever a program changes the **loadSetting** variable in the **WashingMachine** instance in the figure. This data-oriented triggering can be caused by any Xerox LOOPS program when it changes the variable, whether or not that program is written in the rules language.

2. THE RULE LANGUAGE

This chapter describes the syntax and semantics of the rule language.

2.1 Language Introduction

A rule in Xerox LOOPS describes actions to be taken when specified conditions are satisfied. A rule has three major parts called the *left hand side* (LHS) for describing the conditions, the *right hand side* (RHS) for describing the actions, and the *meta-description* (MD) for describing the rule itself. In the simplest case without a meta-description, there are two equivalent syntactic forms:

LHS -> *RHS*;

IF *LHS* **THEN** *RHS*;

The **If** and **Then** tokens are recognized in several combinations of upper and lower case letters. The syntax for LHSs and RHSs is given below. In addition, a rule can have no conditions (meaning always perform the actions) as follows:

-> *RHS*;

if *T* **then** *RHS*;

Rules can be preceded by a meta-description in braces as in:

{*MD*} *LHS* -> *RHS*;

{*MD*} **If** *LHS* **Then** *RHS*;

{*MD*} *RHS*;

Examples of meta-information include rule-specific control information, rule descriptions, audit instructions, and debugging instructions. For example, the syntax for one-shot rules shown in Section 1.5, "One-Shot Rules:"

{**1**} **IF** *condition* *condition* **THEN** *action* ;

is an example of a meta-description. Another example is the use of meta-assignment statements for describing audit trails and rules. These statements are discussed in Section 1.7, "Saving an Audit Trail of Rule Invocation."

LHS Syntax: The clauses on the LHS of a rule are evaluated in order from left to right to determine whether the LHS is satisfied. If they are all satisfied, then the rule is satisfied. For example:

A B C+D (Prime D) -> RHS;

In this rule, there are four clauses on the LHS. If the values of some of the clauses are **NIL** during evaluation, the remaining clauses are not evaluated. For example, if **A** is non-**NIL** but **B** is **NIL**, then the LHS is not satisfied and **C+D** will not be evaluated.

RHS Syntax: The RHS of a rule consists of actions to be performed if the LHS of the rule is satisfied. These actions are evaluated in order from left to right. Actions can be the invocation of RuleSets, the sending of Xerox LOOPS messages, Interlisp function calls, variables, or special termination actions.

RuleSets always return a value. The value returned by a RuleSet is the value of the last rule that was executed. Rules can have multiple actions on the right hand side. Unless there is a **Stop** statement or transfer call as described later, the value of a rule is the value of the last action. When a rule has no actions on its RHS, it returns **NIL** as its value.

Comments: Comments can be inserted between rules in the RuleSet. They are enclosed in parentheses with an asterisk for the first character as follows:

(* This is a comment)

2.2 Kinds of Variables

Xerox LOOPS distinguishes the following kinds of variables:

RuleSet arguments: All RuleSets have the variable **self** as their workspace. References to **self** can often be elided in the RuleSet syntax. For example, the expression **self.Print** means to send a **Print** message to **self**. This expression can be shortened to **.Print**. Other arguments can be defined for RuleSets. These are declared in an **Args:** declaration.

Instance variables: All RuleSets use a Xerox LOOPS object for their workSpace. In the LHS and RHS of a rule, the first interpretation tried for an undeclared literal is as an instance variable in the work space. Instance variables can be indicated unambiguously by preceding them with a colon, (e.g., **:varName** or **obj:varName**).

Class variables: Literals can be used to refer to class variables of Xerox LOOPS objects. These variables must be preceded by a double colon in the rule language, (e.g., **::classVarName** or **obj::classVarName**).

Temporary variables: Literals can also be used to refer to temporary variables allocated for a specific invocation of a RuleSet. These variables are initialized to **NIL** when a RuleSet is invoked. Temporary variables are declared in the **Temporary Vars** declaration in a RuleSet.

Audit record variables: Literals can also be used to refer to instance variables of audit records created by rules. These literals are used only in *meta-assignment* statements in the MD part of a rule. They are used to describe the information saved in audit records, which can be created as a side-effect of rule execution. These variables are ignored if a RuleSet is not compiled in *audit* mode. Undeclared variables appearing on the left side of assignment statements in the MD part of a rule are treated as

audit record variables by default. These variables are declared indirectly -- they are the instance variables of the class declared as the *Audit Class* of the RuleSet.

Interlisp variables: Literals can also be used to refer to Interlisp variables during the invocation of a RuleSet. These variables can be global to the Interlisp environment, or are bound in some calling function. Interlisp variables can be used when procedure-oriented and rule-oriented programs are intermixed. Interlisp variables must be preceded by a backSlash in the syntax of the rule language (e.g., *VispVarName*).

Reserved Words: The following literals are treated as *read-only* variables with special interpretations:

self	[Variable]
The current work space.	
rs	[Variable]
The current RuleSet.	
caller	[Variable]
The RuleSet that invoked the current RuleSet, or NIL if invoked otherwise.	
ruleApplied	[Variable]
Set to T if some rule was applied in this cycle. (For use only in while-conditions).	

The following reserved words are intended mainly for use in creating audit trails:

ruleObject	[Variable]
Variable bound to the object representing the rule itself.	
ruleNumber	[Variable]
Variable bound to the sequence number of the rule in a RuleSet.	
ruleLabel	[Variable]
Variable bound to the label of a rule or NIL .	
reasons	[Variable]
Variable bound a list of audit records supporting the instance variables mentioned on the LHS of the rule. (Computed at run time.)	

auditObject [Variable]

Variable bound to the object to which the reason record will be attached. (Computed at run time.)

auditVarName [Variable]

Variable bound to the name of the variable on which the reason will be attached as a property.

Other Literals: As described later, literals can also refer to Interlisp functions, Xerox LOOPS objects, and message selectors. They can also be used in strings and quoted constants.

The determination of the meaning of a literal is done at compile time using the declarations and syntax of RuleSets. The characters used in literals are limited to alphabetic characters and numbers. The first character of a literal must be alphabetic.

The syntax of literals also includes a compact notation for sending unary messages and for accessing instance variables of Xerox LOOPS objects. This notation uses *compound literals*. A compound literal is a literal composed of multiple parts separated by a periods, colons, and commas.

2.3 Rule Forms

Quoted Constants: The quote sign is used to indicate constant literals:

a b=3 c='open d=f e='(This is a quoted expression) -> ...

In this example, the LHS is satisfied if **a** is non-NIL, and the value of **b** is 3, and the value of **c** is exactly the atom **open**, the value of **d** is the same as the value of **f**, and the value of **e** is the list **(This is a quoted expression)**.

Strings: The double quote sign is used to indicate string constants:

IF a b=3 c='open d=f e=="This is a string"
THEN (WRITE "Begin configuration task") ... ;

In this example, the LHS is satisfied if **a** is non-NIL, and the value of **b** is 3, and the value of **c** is exactly the atom **open**, the value of **d** is the same as the value of **f**, and the value of **e** equal to the string **"This is a string"**.

Interlisp Constants: The literals **T** and **NIL** are interpreted as the Interlisp constants of the same name.

a (Foo x NIL b) -> x_T ...;

In this example, the function **Foo** is called with the arguments **x**, **NIL**, and **b**. Then the variable **x** is set to **T**.

2.4 Infix Operators and Brackets

To enhance the readability of rules, a few infix operators are provided. The following are infix binary operators in the rule syntax:

+	[Rule Infix Operator]
Addition.	
++	[Rule Infix Operator]
Addition modulo 4.	
-	[Rule Infix Operator]
Subtraction.	
--	[Rule Infix Operator]
Subtraction modulo 4.	
*	[Rule Infix Operator]
Multiplication.	
/	[Rule Infix Operator]
Division.	
>	[Rule Infix Operator]
Greater than.	
<	[Rule Infix Operator]
Less than.	
>=	[Rule Infix Operator]
Greater than or equal.	
<=	[Rule Infix Operator]
Less than or equal.	
=	[Rule Infix Operator]
EQ -- simple form of equals. Works for atoms, objects, and small integers.	
~=	[Rule Infix Operator]
NEQ . (Not EQ .)	

== [Rule Infix Operator]

EQUAL -- long form of equals.

<< [Rule Infix Operator]

Member of a list. (**FMEMB**)

In addition, the rule syntax provides two unary operators as follows:

- [Rule Unary Operator]

Minus.

~ [Rule Unary Operator]

Not.

The precedence of operators in rule syntax follows the usual convention of programming languages. For example

1+5*3 = 16

and

[3 < 2 + 4] = T

Brackets can be used to control the order of evaluation:

[1+5]*3 = 18

Ambiguity of the minus sign: Whenever there is an ambiguity about the interpretation of a minus sign as a unary or binary operator, the rule syntax interprets it as a binary minus. For example

a-b c d -e [-f] (g -h) (_ (\$ Foo) Move -j) -> ...

In this example, the first and second minus signs are both treated as binary subtraction statements. That is, the first three clauses are (1) **a-b**, (2) **c** and (3) **d-e**. Because the rule syntax allows arbitrary spacing between symbols and there is no syntax to separate clauses on the LHS of a rule, the interpretation of "**d -e**" is as a single clause (with the subtraction) instead of two clauses. To force the interpretation as a unary minus operator, one must use brackets as illustrated in the next clause. In this clause, the minus sign in the clause **[-f]** is treated as a unary minus because of the brackets. The minus sign in the function call **(g -h)** is treated as unary because there is no preceding argument. Similarly, the **-j** in the message expression is treated as unary because there is no preceding argument.

2.5 Interlisp Functions and Message Sending

Calls to Interlisp functions are parenthesized with the function name as the first literal after the left parenthesis. Each expression after the function name is treated as an argument to the function. For example:

a (Prime b) [a -b] -> c (Display b c+4 (Cursor x y) 2) ;

In this example, **Prime**, **Display**, and **Cursor** are interpreted as the names of Interlisp functions. Since the expression **[a -b]** is surrounded by brackets instead of parentheses, it is recognized as meaning **a** minus **b** as opposed to a call to the function **a** with the argument minus **b**. In the example above, the call to the Interlisp function **Display** has four arguments: **b**, **c+4**, the value of the function call **(Cursor x y)**, and **2**.

The use of Interlisp functions is usually outside the spirit of the rule language. However, it enables the use of Boolean expressions on the LHS beyond simple conjunctions. For example:

a (OR (NOT b) x y) z -> ... ;

Xerox LOOPS Objects and Message Sending: Xerox LOOPS classes and other named objects can be referenced by using the dollar notation. The sending of Xerox LOOPS messages is indicated by using a left arrow. For example:

**IF cell_ ((\$ LowCell) Occupied? 'Heavy)
THEN (_ cell Move 3 'North);**

In the LHS, an **Occupied?** message is sent to the object named **LowCell**. In the message expression on the RHS, there is no dollar sign preceding **cell**. Hence, the message is sent to the object that is the value of the variable **cell**.

For unary messages (i.e., messages with only the selector specified and the implicit argument **self**), a more compact notation is available as described below.

Unary Message Sending: When a period is used as the separator in a compound literal, it indicates that a unary message is to be sent to an object. (We will alternatively refer to a period as a *dot*.) For example:

tile.Type='BlueGreenCross command.Type='Slide4 -> ... ;

In this example, the object to receive the unary message **Type** is referenced indirectly through the **tile** instance variable in the work space. The left literal is the variable **tile** and its value must be a Xerox LOOPS object at execution time. The right literal must be a method selector for that object.

The dot notation can be combined with the dollar notation to send unary messages to named Xerox LOOPS objects. For example,

\$Tile.Type='BlueGreenCross ...

In this example, a unary **Type** message is sent to the Xerox LOOPS object whose name is **Tile**.

The dot notation can also be used to send a message to the work space of the RuleSet, that is, **self**. For example, the rule

IF scale>7 THEN .DisplayLarge;

would cause a **DisplayLarge** message to be sent to **self**. This is an abbreviation for

IF scale>7 THEN self.DisplayLarge;

2.6 Variables and Properties

When a single colon (:) is used in a literal, it indicates access to an instance variable of an object. For example:

tile:type='BlueGreenCross command:type=Slide4 -> ... ;

In this example, access to the Xerox LOOPS object is indirect in that it is referenced through an instance variable of the work space. The left literal is the variable **tile**, and its value must be a Xerox LOOPS object when the rule is executed. The right literal **type** must be the name of an instance variable of that object. The compound literal **tile:type** refers to the value of the **type** instance variable of the object in the instance variable **tile**.

The colon notation can be combined with the dollar notation to access a variable in a named Xerox LOOPS object. For example,

\$TopTile:type='BlueGreenCross ...

refers to the **type** variable of the object whose Xerox LOOPS name is **TopTile**.

A double colon notation (::) is provided for accessing class variables. For example

truck::MaxGas<45 ::ValueAdded>600 -> ... ;

In this example, **MaxGas** is a class variable of the object bound to **truck**. **ValueAdded** is a class variable of **self**.

A colon-comma notation (:,) is provided for accessing property values of class and instance variables. For example

wire:,capacitance>5 wire:voltage:,support='simulation -> ...

In the first clause, **wire** is an instance variable of the work space and **capacitance** is a property of that variable. The interpretation of the second clause is left to right as usual: (1) the object that is the value of the variable **wire** is retrieved, and (2) the **support** property of the **voltage** variable of that object is retrieved. For properties of class variables

::Wire:,capacitance>5 node::Voltage:,support='simulation -> ...

In the first clause, **wire** is a class variable of the work space and **capacitance** is a property of that variable. In the second clause, **node** is an instance variable bound to some object. **Voltage** is a class variable of that object, and **Support** is a property of that class variable.

The property notation is illegal for ruleVars and lispVars since those variables cannot have properties.

2.7 Computing Selectors and Variable Names

The short notations for instance variables, properties, and unary messages all show the selector and variable names *as they actually appear* in the object.

```
object.selector
object:ivName
object::cvName
object:varname:,propName
```

```
(_ object selector arg arg )
```

For example,

apple:flavor

refers to the **flavor** instance variable of the object bound to the variable **apple**. In Interlisp terminology, this implies implicit quoting of the name of the instance variable (**flavor**).

In some applications it is desired to be able to compute the names. For this, the Xerox LOOPS rule language provides analogous notations with an added exclamation sign (!). After the exclamation sign, the interpretation of the variable being evaluated starts over again. For example

apple:!x

refers to the same thing as **apple:flavor** if the Interlisp variable **x** is bound to **flavor**. The fact that **x** is a Lisp variable is indicated by the backslash. If **x** is an instance variable of **self** or a temporary variable, we could use the notation:

apple:!x

If **x** is a class variable of **self**, we could use the notation:

apple!:::x

All combinations are possible, including:

```
object.!selector
object.!selector
object!:::selector
object:!ivName
object!:cvName
object:!varname:,propName
```

```
(_! object selector arg arg )
```

2.8 Recursive Compound Literals

Multiple colons or periods can be used in a literal, For example:

a:b:c

means to (1) get the object that is the value of **a**, (2) get the object that is the value of the **b** instance variable of **a**, and finally (3) get the value of the **c** instance variable of that object.

Similarly, the notation

a.b:c

means to get the **c** variable of the object returned after sending a **b** message to the object that is the value of the variable **a**. Again, the operations are carried out left to right: (1) the object that is the value of the variable **a** is retrieved, (2) it is sent a **b** message which must return an object, and then (3) the value of the **c** variable of that object is retrieved.

Compound literal notation can be nested arbitrarily deeply.

2.9 Assignment Statements

An assignment statement using a left arrow can be used for setting all kinds of variables. For example,

x_a;

sets the value of the variable **x** to the value of **a**. The same notation works if **x** is a task variable, rule variable, class variable, temporary variable, or work space variable. The right side of an assignment statement can be an expression as in:

x_a*b + 17*(LOG d);

The assignment statement can also be used with the colon notation to set values of instance variables of objects. For example:

y:b_0 ;

In this example, first the object that is the value of **y** is computed, then the value of its instance variable **b** is set to **0**.

Properties: Assignment statements can also be used to set property values as in:

box:x:,origin_47 fact:,reason_currentSupport;

Nesting: Assignment statements can be nested as in

a_b_c:d_3;

This statement sets the values of **a**, **b**, and the **d** instance variable of **c** to **3**. The value of an assignment statement itself is the new assigned value.

2.10 Meta-Assignment Statements

Meta-assignment statements are assignment statements used for specifying rule descriptions and audit trails. These statements appear in the MD part of rules.

Audit Trails: The default interpretation of meta-assignment statements for undeclared variables is as audit trail specifications. Each meta-assignment statement specifies information to be saved in audit records when a rule is applied. In the following example from Figure 4, the audit record must have variables named **basis** and **cf**:

```
{{(basis_Fact cf_1.)}  
IF buyer:familySize>2 machine:capacity<20  
THEN suitability_ 'Poor;
```

In this example, the RHS of the rule assigns the value of the work space instance variable **suitability** to **'Poor** if the conditions of the rule are satisfied. In addition, if the RuleSet was compiled in *audit* mode, then during RuleSet execution an audit record is created as a side-effect of the assignment. The audit record is attached to the **reason** property of the suitability variable. It has instance variables **basis** and **cf**.

In general, an audit description consists of a sequence of meta-assignment statements. The assignment variable on the left must be an instance variable of the audit record. The class of the audit record is declared in the *Audit Class* declaration of the RuleSet. The expression on the right is in terms of the variables accessible by the RuleSet. If the conditions of a rule are satisfied, an audit record is instantiated. Then the meta-assignment statements are evaluated in the execution context of the RuleSet and their values are put into the audit record. A separate audit record is created for each of the object variables that are set by the rule.

2.11 Push and Pop Statements

A compact notation is provided for pushing and popping values from lists. To push a new value onto a list, the notation **_+** is used:

```
myList_+newItem;
```

```
focus:goals_+newGoal;
```

To pop an item from a list, the **_-** notation is used:

```
item_-myList;
```


nextGoal_-focus:goals;

As with the assignment operator, the push and pop notation works for all kinds of variables and properties. They can be used in conjunction with infix operator << for membership testing.

2.12 Invoking RuleSets

One of the ways to cause RuleSets to be executed is to invoke them from rules. This is used on the LHS of rules to express predicates in terms of RuleSets, and on the RHS of rules to express actions in terms of RuleSets. A short double-dot syntax(..) for this is provided that invokes a RuleSet on a work space:

Rs1..ws1

In this example, the RuleSet bound to the variable **Rs1** is invoked with the value of the variable **ws1** as its work space. The value of the invocation expression is the value returned by the RuleSet. The double-dot syntax can be combined with the dollar notation (\$) to invoke a RuleSet by its Xerox LOOPS name, as in

\$MyRules..ws1

which invokes the RuleSet object that has the Xerox LOOPS name **MyRules**.

This form of RuleSet invocation is like subroutine calling, in that it creates an implicit stack of arguments and return addresses. This feature can be used as a mechanism for *meta-control* of RuleSets as in:

IF breaker:status='Open
THEN source_\$OverLoadRules..washingMachine;

IF source='NotFound
THEN \$ShortCircuitRules..washingMachine;

In this example, two "meta-rules" are used to control the invocation of specialized RuleSets for diagnosing overloads or short circuits.

2.13 Transfer Calls

An important optimization in many recursive programs is the elimination of tail recursion. For example, suppose that the RuleSet A calls B, B calls C, and C calls A recursively. If the first invocation of A must do some more work after returning from B, then it is useful to save the intermediate states of each of the procedures in frames on the calling stack. For such programs, the space allocation for the stack must be enough to accommodate the maximum depth of the calls.

There is a common and special case, however, in which it is unnecessary to save more than one frame on the stack. In this case each RuleSet has no more work to do after invoking the other RuleSets, and the value of each RuleSet is the value returned by the RuleSet that it invokes. RuleSet invocation in this case amounts to the evaluation of arguments followed by a direct transfer of control. We call such invocations transfer calls.

The Xerox LOOPS rule language extends the syntax for RuleSet invocation and message sending to provide this as follows:

RS..*ws

The RuleSet **RS** is invoked on the work space **ws**. With transfer calls, RuleSet invocations can be arbitrarily deep without using proportional stack space.

2.14 Stop Statements

To provide premature terminations in the execution of a RuleSet, the Stop statement is provided.

(Stop *value*)

[RuleSet Statement]

value is the value to be returned by the RuleSet.

[This page intentionally left blank]

3. USING RULES IN LOOPS

The Xerox LOOPS rules language is supported by an integrated programming environment for creating, editing, compiling, and debugging RuleSets. This section describes how to use that environment.

3.1 Creating RuleSets

RuleSets are named Xerox LOOPS objects and are created by sending the class **RuleSet** a **New** message as follows:

```
(_ ($ RuleSet) New)
```

After entering this form, the user will be prompted for a Xerox LOOPS name as

RuleSet name: *RuleSetName*

Afterwards, the RuleSet can be referenced using Xerox LOOPS dollar sign notation as usual. It is also possible to include the RuleSet name in the **New** message as follows:

```
(_ ($ RuleSet) New NIL RuleSetName)
```

3.2 Editing RuleSets

A RuleSet is created empty of rules. The RuleSet editor is used to enter and modify rules. The editor can be invoked with an **EditRules** message (or **ER** shorthand message) as follows:

```
(_ RuleSet EditRules)
```

```
(_ RuleSet ER)
```

If a RuleSet is installed as a method of a class, it can be edited conveniently by selecting the **EditMethod** option from a browser containing the class. Alternatively, the **EditMethod** message can be used:

```
(_ ClassName EditMethod selector) [Message]
```

Both approaches to editing retrieve the source of the RuleSet and put the user into the TTYIN or TEdit editor, treating the rule source as text.

Initially, the source is a template for RuleSets as shown in Figure 7. The rules are entered after the comment at the bottom. The declarations at the beginning are filled in as needed and superfluous declarations can be discarded.

```
RuleSet Name: RuleSetName;
WorkSpace Class:  ClassName;
Control Structure:  doAll;
While Condition:  ;
Audit Class:  StandardAuditRecord;
Rule Class:  Rule;
Task Class:  ;
Meta Assignments:  ;
Temporary Vars:  ;
Lisp Vars:  ;
Debug Vars:  ;
Compiler Options:  ;

(* Rules for whatever.  Comment goes here.)
```

Figure 7. Initial Template for a RuleSet

You can then edit this template to enter rules and set the declarations at the beginning. In the current version of the rule editor, most of these declarations are left out. If you choose the **EditAllDecls** option in the RuleSet editor menu, the declarations and default values will be printed in full.

The template is only a guide. Declarations that are not needed can be deleted. For example, if there are no temporary variables for this RuleSet, the **Temporary Vars** declaration can be deleted. If the control structure is not one of the **while** control structures, then the **While Condition** declaration can be deleted. If the compiler option **A** is not chosen, then the **Audit Class** declaration can be deleted.

When you leave the editor, the RuleSet is compiled automatically into a Lisp function.

If a syntax error is detected during compilation, an error message is printed and you are given another opportunity to edit the RuleSet.

3.3 Copying RuleSets

Sometimes it is convenient to create new RuleSets by editing a copy of an existing RuleSet. For this purpose, the method **CopyRules** is provided as follows:

```
(_ oldRuleSet CopyRules newRuleSetName) [Message]
```

This creates a new RuleSet by some of the information from the perspectives of the old RuleSet. It also updates the source text of the new RuleSet to contain the new name.

3.4 Saving RuleSets on Lisp Files

RuleSets can be saved on Lisp files just like other Xerox LOOPS objects. In addition, it is usually useful to save the Lisp functions that result from RuleSet compilation. In the current implementation, these functions have the same names as the RuleSets themselves. To save RuleSets on a file, it is necessary to add two statements to the file commands for the file as follows:

```
(FNS * MyRuleSetNames)
(INSTANCES * MyRuleSetNames)
```

where **MyRuleSetNames** is a Lisp variable whose value is a list of the names of the RuleSets to be saved.

If RuleSets are methods associated with a class, and they are saved by using (FILES?), then the file package saves the appropriate entries. The user does not have to be concerned with editing the filecoms of the file being made.

3.5 Printing RuleSets

To print a RuleSet without editing it, one can send a **PPRules** or **PPR** message as follows:

```
( _ RuleSet PPRules) [Message]
( _ RuleSet PPR) [Message]
```

A convenient way to make hardcopy listings of RuleSets is to use the function **ListRuleSets**. The files will be printed on the **DEFAULTPRINTINGHOST** as is standard in Interlisp-D. **ListRuleSets** can be given four kinds of arguments as follows:

```
(ListRuleSets RuleSetName)
(ListRuleSets ListOfRuleSetNames)
(ListRuleSets ClassName)
(ListRuleSets FileName)
```

In the *ClassName* case, all of the RuleSets that have been installed as methods of the class will be printed. In the last case, all of the RuleSets stored in the file will be printed.

3.6 Running RuleSets from Xerox LOOPS

RuleSets can be invoked from Xerox LOOPS using any of the usual protocols.

Procedure-oriented Protocol: The way to invoke a RuleSet from Xerox LOOPS is to use the **RunRS** function:

(RunRS RuleSet workSpace arg2 ... argN) [Function]

workSpace is the Xerox LOOPS object to be used as the work space. This is "procedural" in the sense that the RuleSet is invoked by its name. *RuleSet* can be either a RuleSet object or its name.

Object-oriented Protocol: When RuleSets are installed as methods in Xerox LOOPS classes, they can be invoked in the usual way by sending a message to an instance of the class. For example, if **WashingMachine** is a class with a RuleSet installed for its **Simulate** method, the RuleSet is invoked as follows:

(_ washingMachineInstance Simulate)

Data-oriented Protocol: When RuleSets are installed in active values, they are invoked by side-effect as a result of accessing the variable on which they are installed.

3.7 Installing RuleSets as Methods

RuleSets can also be used as methods for classes. This is done by installing automatically-generated invocation functions that invoke the RuleSets. For example:

```
[DEFCLASS WashingMachine
  (MetaClass Class doc (* comment) ...)
  ...
  (InstanceVariables (owner ...))
  (Methods
    (Simulate RunSimulateWMRules)
    (Check RunCheckWMRules
      doc (* Rules to Check a washing machine.))
  ...]
```

When an instance of the class **WashingMachine** receives a **Simulate** message, the RuleSet **SimulateWMRules** will be invoked with the instance as its work space.

To simplify the definition of RuleSets intended to be used as Methods, the function **DefRSM** (for "Define Rule Set as a Method") is provided:

(DefRSM ClassName Selector RuleSetName) [Function]

If the optional argument *RuleSetName* is given, **DefRSM** installs that RuleSet as a method using the *ClassName* and *Selector*. It does this by automatically generating an installation function as a method to invoke the RuleSet. **DefRSM** automatically documents the installation function and the method.

If the argument *RuleSetName* is **NIL**, then **DefRSM** creates the RuleSet object, puts the user into an Editor to enter the rules,

compiles the rules into a Lisp function, and installs the RuleSet as before.

DefRSM can be invoked with the browser as follows:

- Position the cursor over a class in a browser.
- Press the middle mouse button. A menu pops up.
- Select the Add option in this menu, and drag the mouse to the right to display the submenu that includes the "DefRSM" option. You are prompted to enter a selector name.

After a RuleSet has been installed as a method by using **DefRSM**, you can then edit that RuleSet by selecting the "EditMethod" option from the browser editing menu.

3.8 Installing RuleSets in Active Values

Note: The following section and any other references to active values within the rule documentation refer to active values as they were implemented in the Buttruss release. The functionality of triggering rules from active values has not been tested using the current implementation of active values. It should work to use the **ExplicitFnActiveValue** class to implement this behavior.

RuleSets can also be used in data-oriented programming so that they are invoked when data is accessed. To use a RuleSet as a *getFn*, the function **RSGetFn** is used with the property **RSGet** as follows:

```
...
(InstanceVariables
  (myVar #(myVal RSGetFn NIL) RSGet RuleSetName))
...
```

RSGetFn is a Xerox LOOPS system function that can be used in an active value to invoke a RuleSet in response to a Xerox LOOPS get operation (e.g., **GetValue**) is performed. It requires that the name of the RuleSet be found on the **RSGet** property of the item. **RSGetFn** activates the RuleSet using the local state as the work space. The value returned by the RuleSet is returned as the value of the get operation.

To use a RuleSet as a *putFn*, the function **RSPutFn** is used with the property **RSPut** as follows:

```
...
(InstanceVariables
  (myVar #(myVal NIL RSPutFn) RSPut RuleSetName))
...
```

RSPutFn is a function that can be used in an active value to invoke a RuleSet in response to a Xerox LOOPS put operation (e.g., **PutValue**). It requires that the name of the RuleSet be found on the **RSPut** property of the item. **RSGetFn** activates the RuleSet using the *newValue* from the put

operation as the work space. The value returned by the RuleSet is put into the local state of the active value.

3.9 Tracing and Breaking RuleSets

Xerox LOOPS provides breaking and tracing facilities to aid in debugging RuleSets. These can be used in conjunction with the auditing facilities and the rule executive for debugging RuleSets. The following summarizes the compiler options for breaking and tracing:

T	Trace if rule is satisfied. Useful for creating a running display of executed rules.
TT	Trace if rule is tested.
B	Break if rule is satisfied.
BT	Break if rule is tested. Useful for stepping through the execution of a RuleSet.

Specifying the declaration **Compiler Options: T;** in a RuleSet indicates that tracing information should be displayed when a rule is satisfied. To specify the tracing of just an individual rule in the RuleSet, the **T** meta-descriptions should be used as follows:

{T} IF *cond* THEN *action*;

This tracing specification causes Xerox LOOPS to print a message whenever the LHS of the rule is tested, or the RHS of the rule is executed. It is also possible to specify that the values of some variables (and compound literals) are to be printed when a rule is traced. This is done by listing the variables in the **Debug Vars** declaration in the RuleSet:

Debug Vars: a a:b a:b.c;

This will print the values of **a**, **a:b**, and **a:b.c** when any rule is traced or broken.

Analogous specifications are provided for breaking rules. For example, the declaration **Compiler Options: B;** indicates that Xerox LOOPS is to enter the rule executive (see Section 3.10, "The Rule Exec") after the LHS is satisfied and before the RHS is executed. The rule-specific form:

{B} IF *cond* THEN *action*;

indicates that Xerox LOOPS is to break before the execution of a particular rule.

Sometimes it is convenient in debugging to display the source code of a rule when it is traced or broken. This can be effected by using the **PR** compiler option as in

Compiler Options: T PR;

which prints out the source of a rule when the LHS of the rule is tested and

Compiler Options: B PR;

which prints out the source of a rule when the LHS of a rule is satisfied, and before entering the break.

3.10 The Rule Exec

A Read-Compile-Evaluate-Print loop, called the rule Executive, is provided for the rule language. The rule Executive can be entered during a break by invoking the Lisp function **RE**. During RuleSet execution, the rule executive can be entered by typing **^f** (<control>-f) on the keyboard.

On the first invocation, **RE** prompts the user for a window. It then displays a stack of RuleSet invocations in a menu to the left of this window in a manner similar to the Interlisp-D Break Package. Using the left mouse button in this window creates an Inspector window for the work space for the RuleSet. Using the middle mouse button pretty prints the RuleSet in the default prettyprint window.

In the main rule Executive window, **RE** prompts the user with "**re:**". Anything in the rule language (other than declarations) that is typed to this Executive will be compiled and executed immediately and its value printed out. For example, you may type rules to see whether they execute or variable names to determine their values. For example:

re: trafficLight:color

Red

re:

this example shows how to get the value of the **color** variable of the **trafficLight** object. If the value of a variable was set by a RuleSet running with auditing, then a **why** question can be typed to the rule executive as follows:

re: why trafficLight:color

IF highLight:color = 'Green farmRoadSensor:cars timer.TL
THEN highLight:color _ 'Yellow timer.Start;

Rule 3 of RuleSet LightRules

Edited: Conway "13-Oct-82"

re:

The rule executive may be exited by typing **OK**.

3.11 Auditing RuleSets

Two declarations at the beginning of a RuleSet affect the auditing. Auditing is turned on by the compiler option **A**. The simplest form of this is

Compiler Options: A;

The **Audit Class** declaration indicates the class of the audit record to be used with this RuleSet if it is compiled in *audit* mode.

Audit Class: StandardAuditRecord;

A **Meta Assignments** declaration can be used to indicate the audit description to be used for the rules unless overridden by a rule-specific meta-assignment statement in a meta-descriptor.

Meta Assignments: cf_.5 support_ 'GroundWff;

3.12 Loading Rules

Set the variable LOOPSUSERSDIRECTORIES to include the directory where the Rules files are stored.

Load the file LOOPSRULES-ROOT.LCOM, which will load the following files from LOOPSUSERSDIRECTORIES:

- LOOPSBACKWARDS.LCOM
- LOOPSMIXIN
- LOOPSRULES.LCOM
- LOOPSRULESP.LCOM
- LOOPSRULESC.LCOM
- LOOPSRULESD.LCOM, which will load the file TTY.LCOM from LISPUSERSDIRECTORIES.

Editing rules will be easier if TEdit is loaded. Loading the Rules does not automatically load TEdit.

3.13 Known Problems

In a rule, the expression \$pipe.ri..\$p compiles to (RunRS (QUOTE (\$ pipe)) (\$ p)), which fails.

Meta-assignment statements cannot handle expressions. This means that statements like {cf _ .5} work fine, but {validity _ 'fact} fails.

A value of 1 in a meta-descriptor statement is always taken to be a one-shot designator. You cannot have a meta-descriptor statement like {cf_1}. However, the number 1.0 can be used; the meta-descriptor statement, {cf_1.0}, works.

Rules have not been tested without loading TEdit in order to edit RuleSets.

[This page intentionally left blank.]

RULES

Modified by: Rick Martin (Martin.pasa@Xerox.com)

14-Apr-86

[This page intentionally left blank.]

--- TABLE OF CONTENTS ---

1. INTRODUCTION TO RULE-ORIENTED PROGRAMMING IN LOOPS	15
1.1 Introduction	15
1.2 Basic Concepts	16
1.3 Organizing a Rule-Oriented Program	17
1.4 Control Structures for Selecting Rules	18
1.5 One-Shot Rules	22
1.6 First-Last Rules	23
1.7 Saving an Audit Trail of Rule Invocation	23
1.7.1 Motivations and Applications	23
1.7.2 Overview of Audit Trail Implementation	24
1.7.3 An Example of Using Audit Trails	24
1.8 Comparison with Other Rule Languages	26
1.8.1 The Rationale for Factoring Meta-Level Syntax	26
1.8.2 The Rationale for RuleSet Hierarchy	27
1.8.3 The Rationale for RuleSet Control Structures	28
1.8.4 The Rationale for an Integrated Programming Environment	29
2. THE RULE LANGUAGE	31
2.1 Language Introduction	31
2.2 Kinds of Variables	32
2.3 Rule Forms	34
2.4 Infix Operators and Brackets	35

2.5 Interlisp Functions and Message Sending	37
2.6 Variables and Properties	38
2.7 Computing Selectors and Variable Names	39
2.8 Recursive Compound Literals	40
2.9 Assignment Statements	41
2.10 Meta-Assignment Statements	41
2.11 Push and Pop Statements	42
2.12 Invoking RuleSets	42
2.13 Transfer Calls	43
2.14 Stop Statements	43
3. USING RULES IN LOOPS	45
3.1 Creating RuleSets	45
3.2 Editing RuleSets	45
3.3 Copying RuleSets	46
3.4 Saving RuleSets on Lisp Files	47
3.5 Printing RuleSets	47
3.6 Running RuleSets from LOOPS	47
3.7 Installing RuleSets as Methods	48
3.8 Installing RuleSets in ActiveValues	49
3.9 Tracing and Breaking RuleSets	50
3.10 The Rule Exec	51
3.11 Auditing RuleSets	52
3.12 Loading Rules	52
3.13 Known Problems	52

LIST OF FIGURES

1. Basic RuleSet	18
2. RuleSet Showing Do1	19
3. RuleSet with WhileAll	20
4. RuleSet Showing Evaluation	25
5. Example of an Audit Trail	26
6. RuleSet Invoked as a Method	30
7. Initial Template for a RuleSet	46

[This page intentionally left blank.]

A. CONVERTING FROM BUTTRESS RULES

The primary change to rules was the removal of the **VarLength** mixin as a super class of **RuleSetSource**. Rules are now stored on the IV **ruleList** of **RuleSetSource**. If any user had added properties to a rule with a construct similar to the following expression, these will need to be reimplemented in some fashion:

```
(PutNthValue ($ some-Instance-Of-A-RuleSetSource) ruleNumber prop)
```

A number of other changes were made to account for changes from Buttress LOOPS to the product release. These should not be noticeable to the general user.

A utility was built to allow rules that were saved in Buttress to be loaded into the product release of LOOPS. In order to convert from Buttress, follow this procedure.

- Load the necessary files for Rules as described in the Section 3.12, "Loading Rules."
- Load the file `LOOPSRULESBACKWARDS`. This redefines `ConvertLoopsFiles` to add functionality for converting rules.
- Do not load the Buttress files that need to be converted. Instead call:

```
(ConvertLoopsFiles files-to-convert T T)
```

This will load, convert, remake, and recompile the files specified by *files-to-convert*.

Any rules that were saved with auditing turned on will need to run through the rule compiler. The old rule compiler output a form that was `(_ ($ StandardAuditRecord) NewTemp)`. The message **NewTemp** no longer exists. The rule compiler now puts out the form `(_ ($ StandardAuditRecord) New)`.

If rules were saved with auditing turned on, then call `(ConvertLoopsFiles files-to-convert T)`. Deleting the final "T", will not enable the cleanup. After recompiling the rules to eliminate the message **NewTemp**, remake your files.

[This page intentionally left blank]

SOURCES>

- * LOOPS 10183 10346 10456
- * LOADLOOPS 10346
- * MEDLEY-PATCH 10346
- * SEDIT-PATCH 10189
- * LOOPS-FILEPKG 10495
- * LOOPSBROWSE 10235
- * INSPECT-PATCH 10357
- * LOOPSUTILITY 10370

LIBRARY>

- * LOOPSMS 10281
- * MASTERSCOPE 10346 10436
- * MSPARSE 10346
- * MSANALYZE 10346
- * MSCOMMON 10346

USERS>

- * LOOPSBACKWARDS
- * CONVERSION-AIDS

USERS>RULES>

- * LOOPSRULES
- * LOOPSRULES-ROOT
- * LOOPSRULESC
- * LOOPSRULESD
- * LOOPSRULESP
- * LOOPSRULESTTY

Ron:

10464 Comment removal AR

10436 MASTERSCOPE fix to MSANALYZE for CLASSES, into Medley

CACHEOBJECT

By: sML (Lanning.pa@Xerox.com)

4-Sep-86

INTRODUCTION

The file CACHEOBJECT defines a Loops mixin that defines a protocol for instances that cache computed values.

CLASSES

ObjectWithCache [Class]

Instance of the class ObjectWithCache follow a standard protocol for manipulating a dynamic cache. The cache is not stored when an ObjectWithCache is saved on a file.

(← *self* ClearCache) [Method of ObjectWithCache]

Clears the entire cache of *self*.

(← *self* ClearCacheEntry *name*) [Method of ObjectWithCache]

Removes the cache entry for *name* on *self*.

(← *self* GetCache *name*) [Function]

Returns the value of the cache entry for *name* on *self*. If there is no cached value, returns NIL.

(← *self* PutCache *name datum*) [Function]

Stores *datum* as the value of the cache entry for *name* on *self*. Returns *datum*.

LOOPS-FB

By: sML (Lanning.pa@Xerox.com)

INTRODUCTION

LOOPS-FB adds a command to the Lisp File Browsers for opening Loops browsers on files.

Loading the file will automatically add a "Browse" command to all new Lisp File Browsers. Selecting the "Browse" item will open a Loops FileBrowser on each of the currently selected files. The files will be loaded first if they are not currently loaded.

LOOPSIDLE

By: sML (Lanning.pa@Xerox.com)

4-Sep-86

INTRODUCTION

LOOPSIDLE make IDLE "bouncing box" function bounce a Loops icon about about the screen.

VARIABLES

BouncingLoopsIcon [Variable]

The (value of the) variable BouncingLoopsIcon is an expanded copy of the Loops icon. LOOPSIDLE sets IDLE.BOUNDING.BOX to (the value of) BouncingLoopsIcon.

Loops Image Objects

Unknown IMAGEOBJ GETFN: LoopsImageObjectGetFn	Unknown IMAGEOBJ GETFN: LoopsImageObjectGetFn

Interlisp IMAGEOBJS are "objects" that know how to display themselves on IMAGESTREAMs. IMAGEOBJS are most often used to insert non-character items into a TEdit document. Standard Interlisp IMAGEOBJS are available for displaying bitmaps, graphs, and horizontal bars.

Interlisp IMAGEOBJS are not objects in the sense of Loops: there is no provision for specialization of existing IMAGEOBJS and no default behavior is provided. Creation of a new type of IMAGEOBJS requires some effort. Functions must be specified for printing, reading, displaying, and handling button events for the new IMAGEOBJ type.

Loops image objects are Loops objects that can be used as Interlisp IMAGEOBJS. They let the programmer use the full power of the Loops environment in the creation of new IMAGEOBJS.

The LoopsImageObjects file defines a number of classes that can be used to create Interlisp IMAGEOBJS, and an interface that makes it easy to insert these Loops image objects into a TEdit document. Section `LoopsImageObjects:in` defines how to insert Loops image objects into a TEdit

document; section `LoopsImageObjectGetFn` describes some image object classes; sections `LoopsImageObjectGetFn` and `LoopsImageObjectPutFn` define the image object protocol, for people wishing to define their own Loops image objects.

Unknown IMAGEOBJ GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ GETFN: LoopsImageObjectGetFn

Inserting Loops image objects into a TEdit document is easy.

(LIO)

[Function]

The standard way to insert IMAGEOBJS into a TEdit file is via the CONTROL-O character. Hitting CONTROL-O lets you type in a LISP form, and the value of this form is an IMAGEOBJ that will be inserted into the document at the current location. (LIO) will present you with a menu of all known Loops image object classes. If you select a class from this menu, (LIO) will return the IMAGEOBJ that points to a new Loops image object of the specified class. If the instance has IVs that can be edited, you will be given a chance to edit the instance first.

LIOInsertCharCodes

[Variable]

When the Loops image object package is loaded, it redefines the interpretation in TEdit of the characters specified by the variable LIOInsertCharCodes. Hitting one of these characters is equivalent to hitting CONTROL-O and then typing in the form (LIO). The default value of LIOInsertCharCodes is (the value of) (LIST (CHARCODE #W) (CHARCODE #w)). This is an INITVAR, so you override the default before you load the package.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Many of these objects display themselves as text. This can be confusing for the user, who can easily mistake the object for a sequence of characters. To avoid this possibility for confusion, these objects appear boxed when displayed in a TEdit window. The box will not appear in a printed document.

The character looks of these objects can be set from TEdit in the same way that normal characters can have their looks set: select the object and use TEdit's Character Looks Menu, or the Looks item on the title bar pop-up menu. Note that the object can have only a single character looks--all the characters in the object will be displayed with the same looks. There is no way to change the looks of single characters displayed by a Loops image object.

These objects often violate the principle of WYSIWYG (What You See Is What You Get) in that they display more text in a TEdit window than they do when printed. They let you know what text will appear in print by displaying all non-printing text inverted. Thus, the object that displays in a TEdit

window as **Index: IndexEntry** will print up as the string "IndexEntry" when printed. (Note the interaction of this inversion with the boxing mentioned in the previous paragraph: the inversion is performed after the boxing, so that the entire object will still appear boxed if displayed inverted (as in TEdit's highlighting to indicate pending-delete).)

These objects respond to a button event by presenting a menu of options to the user. The items in the menu depend on the class of the object. All objects include a menu item for storing the Loops instance in (SavedValue), and for inspecting the instance. If the instance has IVs that can be set by the user, the menu will include an "Edit" item. The "Edit" item will bring up the standard Loops instance editor on the object. If the instance has a textual IV, the menu will include an option to edit the value of this IV in a new TEdit window.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Many people include in a TEdit file the time the file was last edited, or the name of the file that holds the document. The following classes can be used to automate this process.

WhenLastSaved [Class]

A WhenLastSaved image object will display a time stamp indicating the time the TEdit document was last saved. For example, a WhenLastSaved instance in this document might produce the image  (but would appear without the box when hardcopied).

If the document has not been saved since the WhenLastSaved object was inserted, the WhenLastSaved object will display the string "NotYetSaved" instead of the time stamp.

WhereLastSaved [Class]

A WhereLastSaved image object is much like a WhenLastSaved object, except that it displays the name of the file that the TEdit document was last saved in, instead of the time last saved.

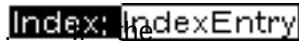
WhenLastSaved and WhereLastSaved objects are used in the running footers in this document.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Using image objects, it is easy to build an index table for a TEdit document.

IndexEntry [Class]

An IndexEntry object associates a string with a page number for inclusion in an index. The IV text of an IndexEntry will be added to the accumulated index, together with the number of the page containing the object. If the IV displayText? is non-NIL (the default), then the text string will also appear in the document, otherwise the object will be invisible in the final hardcopy. For example, an IndexEntry that adds a reference to the string "IndexEntry" appears in a TEdit window as

 `displayText?` IV were NIL, it would appear as  it window, but would be invisible when printed.

`InitIndex` [Class]

An `InitIndex` object must appear in a TEdit document before any `IndexEntry` object. The `InitIndex` object initializes the collection of the index table. It will be invisible when printed.

`CollectIndex` [Class]

A `CollectIndex` object should appear after all the `IndexEntry` objects that appear in a TEdit document. It will sort the index and print the index information into a new TEdit window (you will be prompted for the window). You can then format the text, save it away, print it, or discard it. The IVs `looks`, `paraLooks`, `firstPageFormat`, `rectoPageFormat`, and `versoPageFormat` determine the initial formatting of the index. See the TEdit documentation for a discussion of the definition of these fields. The `CollectIndex` object will not be visible in the printed document.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Automatic chapter and section numbering, and generation of a table of contents, is provided by the following classes.

`SectionHeading` [Class]

`SectionHeading` objects have three IVs: `level`, `text`, and `displayText?`. The `level` IV determines the subsection nesting of the object. Each `SectionHeading` object in the document increments the current section number at the given level. Smaller level numbers will not be affected. Larger level numbers are reset to zero. For example, a sequence of `SectionHeading` objects with `level` IVs of 1, 2, 1, 2, 2, 3, 2, and 1 will produce final section numbers of 1., 1.1., 2., 2.1., 2.2., 2.2.1., 2.3., and 3.

The `text` IV determines the section title for inclusion in the table of contents. If the `displayText?` is non-NIL (the default), the `text` will also be displayed in the document.

As an example, when first displayed in a TEdit window the `SectionHeading` for this subsection looked like  `displayText?` IV were NIL, the text would have been inverted to indicate that it will not be present in the final hardcopy:



hardcopied, the object displayed the section number used at hardcopy time instead of the placeholder "n.n.". (Due to a "bug", you have to redisplay the window to see the updated section numbers.)

InitTOC [Class]

An InitTOC object must appear in a TEdit document before any SectionHeading object. The InitTOC object initializes the collection of the table of contents. The IV initialSectionNumbers can be used to initialize the first section numbers to values other than 1. For example, if the initialSectionNumbers IV is (2 1 3), and the first SectionHeading object has a level IV of 3, it will be given the subsection number "2.1.4.". The InitTOC object will be invisible when printed.

CollectTOC [Class]

A CollectTOC object should be placed at the end of a document, after all SectionHeading objects, to collect the table of contents information into a new TEdit stream. When the document is hardcopied, a new TEdit stream will be opened containing the accumulated table of contents. You can then format this TEdit document as you wish. CollectTOC uses the same IVs as does the CollectIndex to determine the initial formatting of the table of contents. The CollectTOC object not be visible in the final hardcopy.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

The following classes provide a somewhat clumsy way of having page and section references included an a document. Because there is no lookahead in the TEdit page formatting, documents containing these objects must be hardcopied twice. The first time will compute the correct page and section numbers, the second time will incorporate them into the document.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

PageNote [Class]

At hardcopy time, a PageNote object remembers the current page number, associating it with the value of its tag IV. See the PageReference class, below.

PageReference [Class]

At hardcopy time, a PageReference object looks up the value of its tag IV to see if a PageNote object has noted a page number for that tag. If there is page number stored, the PageReference object will display the number, otherwise it will display the string "nn". If the PageNote occurs before

the `PageReference` object in the document, this reference will be found the first time the document is printed. If the `PageNote` occurs after the `PageReference` object, a second hardcopy must be generated.

Unknown IMAGEOBJ
GETFN: `LoopsImageObjectGetFn`

`SectionNote` [Class]

At hardcopy time, a `SectionNote` object remembers the current section number, associating it with the value of its tag IV. See the `SectionReference` class, below.

`SectionReference` [Class]

At hardcopy time, a `SectionReference` object looks up the value of its tag IV to see if a `SectionNote` object has noted a section number for that tag. If there is section number stored, the `SectionReference` object will display the number, otherwise it will display the string "n.". If the `SectionNote` occurs before the `SectionReference` object in the document, this reference will be found the first time the document is printed. If the `SectionNote` occurs after the `SectionReference` object, a second hardcopy must be generated.

`SectionReferences` are used in this document in the last paragraph of the first section.

Unknown IMAGEOBJ
GETFN: `LoopsImageObjectGetFn`

The following class illustrates the possibility of nesting image objects inside of other image objects.

`BoxedImageObject` [Class]

IVs: `boxShade` (the shade of the box: either a shade or a form that will be evaluated to a shade; e.g. `GRAYSHADE`), `boxWhiteSpace` (blank space between the boxed object and the box), `boxWidth` (the width of the box), `boxedObject` (a `LoopsImageObject`).

Unknown IMAGEOBJ
GETFN: `LoopsImageObjectGetFn`

A common document form consists of a table of text entries. These are easy to build in TEdit when each entry fits within its row without having to occupy multiple lines.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Example: BoxedImageObjects and TableTextObjects

The following class takes care of the case where the table entries are too large to fit on a single line.

TableTextObject [Class]

describes a (possibly multiple line) table entry. The class `TableTextObject` defines five important IVs.

`text` [IV of TableTextObject]

contains the text that will be displayed in the table entry.

`widthInWs` [IV of TableTextObject]

defines the width of the table entry. This width is measured in "W"s, in the current font. If the `text` is too long, it will be broken at word boundaries and displayed in multiple lines. The default value is 10.

`nLines` [IV of TableTextObject]

defines the maximum number of lines to be used to print the text. If `nLines` is `NIL` (the default), the object will be as tall as it needs to be to display the entire `text` subject to the width constraint specified by the `widthInWs` IV.

`justify` [IV of TableTextObject]

determines the horizontal justification of the individual lines within the table entry. Possible values are `left` (for left-justify), `center` (for centered text), and `right` (for right-justify). The default is `left`.

`verticalJustify` [IV of TableTextObject]

determines the vertical justification of the lines within the table entry. Possible values are `top` (the first line will be flush with the top of the table entry), `center` (the text will be centered vertically), and `bottom` (the last line of text will be at the bottom of the entry). The default is `bottom`. Note that this IV only has an effect if the IV `nLines` is specified.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Image objects can be used to include prettyprinted forms in a document.

PPIImageObject [Class]

prettyprints a form in the image stream. The relevant IVs of PPIImageObject are:

form [IV of PPIImageObject]

~mumble~

minWidth [IV of PPIImageObject]

~mumble~

maxWidth [IV of PPIImageObject]

~mumble~

Due to an unfortunate bug in Interlisp, PPIImageObject work correctly on display devices .

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Example: PPIImageObject

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Loops objects that are wrapped around an Interlisp IMAGEOBJ. The IMAGEOBJ in turn points back to the Lops image object. When Interlisp "sends a message" to the underlying IMAGEOBJ (more correctly, applies one of the IMAGEOBJ's IMAGEFNS to the IMAGEOBJ), the IMAGEOBJ forwards this message on to the Loops image object. The translation from Interlisp IMAGEOBJ protocol to Loops image object protocol is accomplished by a special set of IMAGEFNS and the LoopsImageObject class.

LoopsImageObject [AbstractClass]

~mumble~

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

The following methods implement the IMAGEFNS for the Loops image object. These message are not intended to be sent by the user; they are sent automatically by TEdit and other packages.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

```
(_ self ImageBox imageStream currentX
    rightMargin)
```

[Method of LoopsImageObject]

returns an instance of the record IMAGEBOX with fields XSIZE, YSIZE, XKERN, and YDESC. This is used by TEdit to determine the size of the object for formatting purposes. Specializations of LoopsImageObject should override this method.

```
(_ self Display imageStream)
```

[Method of LoopsImageObject]

should actually display the object on the *imageStream*. Specializations of LoopsImageObject should override this method.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

```
(_ self ButtonEventIn windowStream selection
    relX relY window textStream button)
```

[Method of LoopsImageObject]

~mumble~

```
(_ self CopyButtonEventIn windowStream)
```

[Method of LoopsImageObject]

~mumble~

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

```
(_ self Copy)
```

[Method of LoopsImageObject]

~mumble~

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Due to restrictions imposed by IMAGEOBJets, LoopsImageObjects cannot duplicate the full generality of saving and retoring Loops objects. Specialized subclasses of LoopsImageObjects cannot change how they are stored or read back in. However, the following methods do give the user some control over what extra information is dumped out when a LoopsImageObjects is stored.

(*_ self BeforePutToFile stream*) [Method of LoopsImageObject]

~mumble~

(*_ self AfterPutToFile fileStream*) [Method of LoopsImageObject]

~mumble~

(*_ self AfterGetFromFile textStream*) [Method of LoopsImageObject]

~mumble~

(*_ self PrePrint*) [Method of LoopsImageObject]

~mumble~

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

(*_ self WhenCopied targetWindowStream
sourceTextStream targetTextStream*) [Method of LoopsImageObject]

~mumble~

(*_ self WhenDeleted targetWindowStream
sourceTextStream targetTextStream*) [Method of LoopsImageObject]

~mumble~

(*_ self WhenInserted targetWindowStream
sourceTextStream targetTextStream*) [Method of LoopsImageObject]

~mumble~

(*_ self WhenMoved targetWindowStream
sourceTextStream targetTextStream*) [Method of LoopsImageObject]

~mumble~

```
(_ self WhenOperatedOn windowStream howOperatedOn
    selection textStream)
```

[Method of LoopsImageObject]

~mumble~

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

The class `LoopsImageObject` provides a number of other methods that can be used by specializations of the above methods.

```
(_ self CachedImageBox imageStream)
```

[Method of LoopsImageObject]

returns the image box computed by the last `ImageBox` method. This is often used inside the `Display` method to avoid resending the `ImageBox` message.

```
(_ self DisplayImageStream? imageStream)
```

[Method of LoopsImageObject]

returns `NIL` unless the image stream is a display image stream.

```
(_ self PrintText imageStream text font)
```

[Method of LoopsImageObject]

prints the string *text* in the font *font*, centered in the object's `CachedImageBox`.

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

Certain subclasses of `LoopsImageObject` capture some commonly desired functionality. These may be useful to anyone interested in building their own Loops image objects.

`TEditImageObject`

[AbstractClass]

The class `TEditImageObject` contains a few methods that are only applicable when the object is being displayed in a `TEdit` stream.

```
(_ self CurrentFont imageStream)
```

[Method of TEditImageObject]

returns the font of the image object in the current `TEdit` stream.

(*_ self* TextStream *imageStream*) [Method of TEditImageObject]

returns the text stream that is being displayed in *imageStream*.

(*_ self* TEditIV *ivName* *textStream*) [Method of TEditImageObject]

edits the text stored in the IV *ivName* of *self* in a new TEdit window. The image object should be part of the TEdit stream *textStream*. The *textStream* argument is used to update the display when the IV has been edited.

(*_ self* AllObjects *textStream*) [Method of TEditImageObject]

returns a list of all imageobjects contained in *textStream*, in order.

WhenSavedImageObject [AbstractClass]

~mumble~

HardcopySideEffectObject [AbstractClass]

~mumble~

LabelImageMixin [AbstractClass]

~mumble~

EditableImageObjectMixin [AbstractClass]

~mumble~

TEditableImageObjectMixin [AbstractClass]

~mumble~

Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

On occasion, it may be useful to wrap a Loops image object around an existing Interlisp IMAGEOBJ, say to wrap a BoxedImageObject around it.

ImageObjectWrapper [Class]

~mumble~



Unknown IMAGEOBJ
GETFN: LoopsImageObjectGetFn

LOOPSINDEX

By: sML (Lanning.pa@Xerox.com)

4-Sep-86

INTRODUCTION

LOOPSINDEX sets up the SINGLEFILEINDEX package to dump out classes, methods, and instances in an intelligible way. LOOPSINDEX can be loaded into any Interlisp sysout: you do not need to have Loops loaded to load LOOPSINDEX.

LOOPSINDEX adds appropriate entries to the lists SINGLEFILEINDEX.TYPES and SINGLEFILEINDEX.FILTERS to cause SINGLEFILEINDEX to index all classes, methods, and instances defined on a file. See the documentation of SINGLEFILEINDEX for a detailed description of these variables.

LOOPSINDEX will load SINGLEFILEINDEX if it is not already loaded.

HOW TO SET UP FOR TESTING LOOPS

Start with a fresh LISP.SYSOUT from {erinyes}<lisp>lyric>basics>, dated 27-Apr-87.

Give no INIT file

Log in.

Type into the exec:

```
(SETQ IL:DISPLAYFONTDIRECTORIES '("{ERINYES}<lisp>LYRIC>FONTS>"))
CONN {ERINYES}<lisp>Lyric>library>
(il:load 'TEDIT.lcom)
(il:tedit '{erinyes}<cate3>loops>LOOPS-setup.tedit)
```

This should bring up this document.

Shift selected all lines below here into the EXEC:

```
(il:load 'FILEBROWSER.lcom)
CONN {erinyes}<lisp>lyric>lispusers>
(il:load 'WHO-LINE.dfasl)
(il:load 'filewatch.lcom)

CONN {erinyes}<lispusers>lyric>
(il:load 'CROCK.lcom)

CONN {erinyes}<test>lisp>lyric>internal>library>
(il:load 'do-test.dfasl)

(setq il:*DEFAULT-CLEANUP-COMPILER* 'cl:COMPILE-FILE)

(setq il:ch.default.domain "AISNORTH")
(SETQ IL:CH.DEFAULT.ORGANIZATION "XEROX")
CONN |{PELE:AISNorth:Xerox}<CATE3>|

(il:closew il:logow)
(IL:CHANGEBACKGROUND 42405)
(il:crock)
```

Follow the instructions for loading loops from the release notes manual.

As long as the workstation can get the font files from the network, steps 1-3 can be ignored in the Installation of Loops (4.3 of Release Notes)

For step 4:

The modified network version:

```
CONN {ERINYES}<lisp>Lyric>library>
(il:load 'grapher.lcom)
```

For steps 5-6:

Insert Lyric LOOPS System #1 floppy, then type:

```
(il:fb '{floppy})
```

and copy all files to {dsk}<lispfiles>loops>

Do the same for the Lyric LOOPS System #2 floppy

Then type or shift select in an Interlisp window:

```
CONN "{DSK}<lispfiles>loops>"
(LOAD 'LOOPS)
```

Stuff the following lines into a temp file in core then shift select out the sysout command.

```
(il:sysout '{erinyes}<cate3>loops>test-loops.sysout)
(il:login)
(il:fb '{erinyes}<cate3>loops>)
; Make sure the Loops stuff is copied to <lispfiles>loops>
load the right software, and change to loops stuff
```


The basic loop is (TIMEALL *form* 10000).

[[Timing old SmallLoops, on an 1132]]

Test name	Time (secs)	Time in test (less var lookup)
Global variable lookup	0.05	0.00
Field fetch	0.07	0.02
Function call	0.11	0.06
Send	0.48	0.43 (= x 7 a function call)
Inherited GetValue	1.95	1.90 (= x 32 a function call)
Local GetValue	0.58	0.53 (= x 9 a function call)
Local PutValue	0.69	0.64 (= x 11 a function call)
Inherited GetIVProp	1.97	1.92 (= x 32 a function call)
Local PutIVProp	1.20	1.05 (= x 18 a function call)
Instance creation, 2IVs	???	??? (= x ??? a function call)

[[Timing new Loops, on an 1132]]

Test name	Time (secs)	Time in test (less var lookup)
Global variable lookup	0.05	0.00
Field fetch	0.07	0.02
Function call	0.11	0.06
Send (local cache)	0.22	0.17 (= x 3 a function call)
Send (global cache)	0.??	0.?? (= x 7 a function call)
Send (no cache)	0.??	0.?? (= x 7 a function call)
Inherited GetValue	1.77	1.72 (= x 29 a function call)
Local GetValue	0.37	0.32 (= x 5 a function call)
Local PutValue	0.48	0.43 (= x 7 a function call)
Inherited GetIVProp	3.36	3.31 (= x 55 a function call)
Local PutIVProp	1.11	1.06 (= x 18 a function call)
Instance creation, 2IVs	7.85	7.85 (= x 131 a function call)

Inspecting and Repairing Notefiles

Xerox Corporation

Randy Trigg

[First written: 8/9/85 Randy Trigg]
[Last updated: 8/26/85 Randy Trigg]

This document describes the Notefile inspector facility available via the Inspect&Repair option on the Notefile Ops menu in NoteCards Release1.2i.

The old Repair Notefile facility rebuilt the links in a Notefile from the contents of card substances. This was used whenever a notefile was thought to have inconsistent links. The problem was that notefiles with inconsistent links often had other problems that caused Repair to break. Thus the motivation for developing the notefile inspector documented here was to verify a notefile's readability before invoking the link rebuild. As it turns out, this inspector is useful generally for checking the health of a notefile, deleting cards and backing up other cards (or more precisely, card parts) to previous versions. Thus, you may want to use the inspector even if your notefile is healthy and doesn't need its links rebuilt.

The notefile inspector has three separate phases: reading the notefile's data area searching for healthy card parts, allowing the user to make modifications, and rebuilding the links. The process can be aborted after phase 1 or 2 if desired. This document begins with a brief discussion of the organization of a notefile. Then follows sections describing each of the three phases. Finally, I outline some tips, strategies and pitfalls to watch for.

1. What you need to know about a notefile's innards.

1.1 Notefile structure.

A notefile consists of two parts, the index and the data area. Each card in the notefile has an entry in the index. An index entry has 5 parts, a status field and 4 pointers. The status field specifies whether the index entry is free or occupied by an active or deleted card. There is one pointer in the index entry to each of the 4 parts of a card: substance, title, links and property list. These point into the data area. Whenever you change, say, the title of a card, the new title is written to the end of the data area and the index entry title pointer for that card is updated to point to the new location in the data area. Thus, in general, a notefile's data area grows every time any part of any card is changed. To throw away the old versions of card parts, it is necessary to compact the notefile.

1.2 Card IDs.

Every card in the notefile has a unique ID, e.g. NC00023. The top level fileboxes; **Contents**, **Orphans**, and **To Be Filed** have IDs NC00001, NC00002, and NC00003, respectively. Note that these boxes cannot be deleted. The IDs from NC00004 through NC00020 are unused. Currently, an old ID is never reused, even if its card is deleted and the notefile is compacted. Thus, if the inspector shows no entry in the card inspector menu for some ID, it is because that card has been deleted. If you've asked to show deleted cards and it still doesn't appear, it's because the notefile has been compacted since that card was deleted.

1.3 Card parts.

Of the four parts of a card, the title and property list are simplest. The title is simply a string while the property list is a list of attribute value pairs. If you have not been attaching properties to your cards, then the inspector will only show those properties that the system maintains. Currently the only such property is "Updates" with value equal to a list of dates on which the card was updated going chronologically backwards from the front of the list to the back.

The substance of the card is simply its contents. Thus a text card's substance is a text stream, a browser card's is its graph, etc. These are stored on the file in a manner appropriate to the substance type. Thus a text substance on the notefile looks like the way TEdit writes out text streams (text followed by "looks" information).

1.4 Links on the notefile.

The links of a card are divided into three groups: **to links**, **from links**, and **global links**, where the global links form a subset of the to links. The to links of a card are those links pointing from this card to some other card. The from links are those links pointing from another card to this one. Finally, the global links are those to links that are global, that is, they point from this card as a whole to another card. (Global links are the ones that aren't anchored in the source card's substance. Source links are an example.)

The confusing thing about links out on the notefile is that they are stored in several places. All links are stored as to links with their source card and as from links with their destination card. Furthermore, links that are not global are also stored within the substance of their source card inside of a link icon. Links that are global are also stored on the global links list of their source card. Thus, all links are stored in three places: as a to link on the source card, as a from link on the destination card and either in the substance of the source card or on its global links list. If these three records of a link don't agree for some reason, then we say that the notefile is **inconsistent** and needs its links rebuilt.

1.5 The links rebuilder.

The third phase of the inspector rebuilds the links of a notefile as follows: First it removes all the to and from links for every card. Then it reads the substances for each card and recreates to links and from links by looking at the links found inside the link icons in the substance.

The link rebuilder is also able to rebuild bad filebox substances. It does this by looking for all cards in the notefile with from links from the bad box and creating a new substance for the box containing only links to those cards. This process loses any text that the box might have contained as well as scrapping the original ordering of links. Nonetheless, in some cases this may be preferred to backing up the substance to a previous version or to deleting the box altogether.

The links rebuilder can rebuild the notefile's list of link types in a similar manner. That is, it records the set of link types seen on valid links and replaces the old links types with the new set. Note that this throws away any link types for which there are no links in the notefile.

Finally, the links rebuilder can rebuild bad global links for a card. It does this by looking for any cards with from links from the bad card that are global. This assumes that the card at the destination end has good links. Thus, if the cards at both ends of a global link have unreadable links, then there is no way to recover that link.

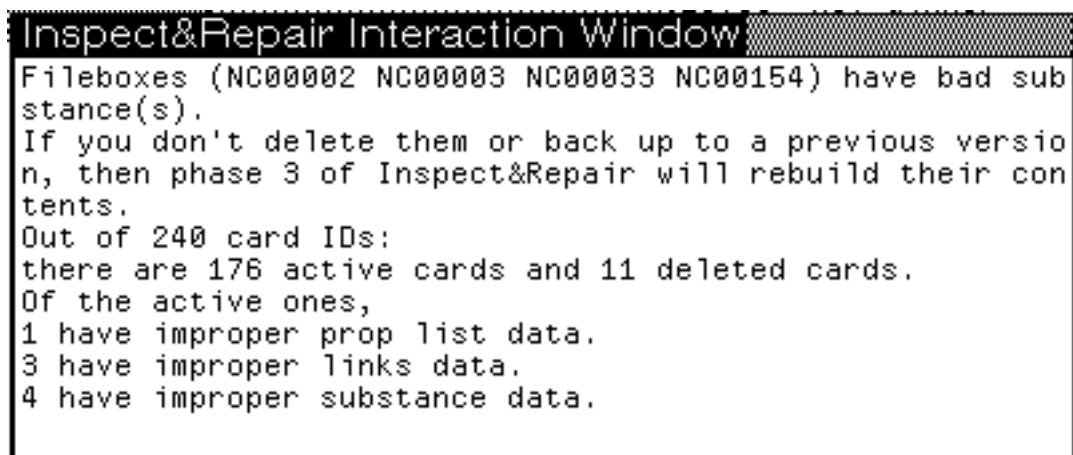
The inspector provides the option of having the links rebuilder phase rebuild bad filebox substances, bad link types, and bad global links. See Section ?? below.

2. Running the Notefile inspector: Phase 1: Scouring the data area

To start the inspector, first be sure that there is no open notefile. Then select the item **Inspect&Repair** from the NoteFile Ops menu. There is one option available at this level by "pulling to the side" called **ReadSubstances**. This ensures that substances of all cards pronounced valid by the inspector are readable. If this option is not invoked, then a check is still run on the length of the substance, but not on its contents. Unfortunately, the ReadSubstances option requires MUCH more work by phase 1. I recommend that you only use this option if Phase 3 (link rebuilding) breaks with some error like "Bad Piece Tbl" from TEdit. In that case, up-arrow out of the break and start the Inspect&Repair process over again, this time using the slower but more comprehensive ReadSubstances option.

Selecting Inspect&Repair will invoke phase 1 of the inspector, wherein the data area of the notefile is scoured for valid card parts. A record of all such parts is kept and statistics printed out at the end. You'll be asked to position the window in which those statistics as well as later inspector communications will be printed. You can monitor the progress of phase 1 by watching the prompt window. It will be printing messages like "Processing byte xxxxx of yyyy."

When phase 1 has completed and you've positioned the interaction window, statistics on your notefile will be provided. You'll be told the total number of card IDs used and the number of those that are currently associated with active and deleted cards. (The rest are free and will never be reused. See Section 1.2 above.) If all seems well with the world, the next line will read "All active cards look okay." If not, there will be various messages outlining the problems. (See Figure 1.)



```

Inspect&Repair Interaction Window
Fileboxes (NC000002 NC000003 NC000033 NC000154) have bad substance(s).
If you don't delete them or back up to a previous version, then phase 3 of Inspect&Repair will rebuild their contents.
Out of 240 card IDs:
there are 176 active cards and 11 deleted cards.
Of the active ones,
1 have improper prop list data.
3 have improper links data.
4 have improper substance data.
  
```

Figure 1: Snapshot of a sample interaction window

Note that several fileboxes have bad substances and that a special message is printed on their behalf. This indicates that if you don't wish to delete or back these up to a previous version, then phase 3 will rebuild them. (See Section 1.5 above.)

If there are cards having user-defined types whose type definition code has not been loaded, then you'll get a message to that effect, something like "<n> cards have unknown card types (FOO BAR)." At this point you should load the lisp files containing the definitions of the unknown card types. If not, then these cards will show up with bad substance in phase 2. If you have a card for which no substance versions could be read, then you'll also get unknown card type messages for it (reading something like "<n> cards have unknown card types (NIL)"). This is because the inspector couldn't find a card type on the notefile for that card.

A menu of options appears attached to the upper right corner of the interaction window. The particular options you get in that menu depend on the state of your notefile and are described below. The first two options appear in all cases. The other two may or may not be present in the menu you get. In any case, you should select one of the options before attempting any other NoteCards-related work.

ABORT: Choosing this option aborts the Inspect&Repair process entirely, throwing away any changes you might have made (such as card deletions or back ups.)

INSPECT CARDS: This brings up a menu of active card IDs with which you can inspect, delete, or back up particular cards. There is a "pull-across" menu item called **INCLUDE DELETED CARDS**, which if selected will include card IDs for deleted cards as well as active ones. Using this option, one can undelete deleted cards and restore some previous version.

END INSPECT&REPAIR: This option is only available if it seems that you don't need to continue to the link rebuilding phase. You will not get this option if you've deleted any cards, or generally if there are problems with the notefile. Choosing this option causes the Inspect&Repair process to end gracefully (via a normal checkpoint and close notefile), thus skipping phase 3, rebuilding links.

CONTINUE INSPECT&REPAIR: This option is only available if the notefile is in fairly good health (i.e. okay except for fileboxes to rebuild or global links to rebuild - see Section 1.5 above). Selecting it causes Inspect&Repair to move to phase 3 and rebuild your notefile's links.

3. Running the Notefile inspector: Phase 2: Your chance to tinker

After selecting **INSPECT CARDS** in the interaction window's attached menu, a menu containing Notecards IDs will pop up and be attached to the interaction window's lower left corner. It will contain IDs for all active cards and possibly deleted cards as well if you selected the submenu item **INCLUDE DELETED CARDS** described above. The menu can hold some 200 card IDs. If your notefile has more than that, then the menu will be composed of several pages each containing around 200 IDs. Rapid switching between pages is possible.

Attached to the upper right corner of the cards inspector menu is a menu containing at least the two options: **ABORT** and **DONE**. If the menu has multiple pages (there are more than 200 active cards in the notefile), then the attached menu will also include the items **NEXT PAGE**, **PREVIOUS PAGE**, and **FIRST PAGE**. Selecting these causes the current menu to be swapped with either the next menu, previous menu, or first menu, respectively.

Clicking **ABORT** causes the entire Inspect&Repair process to quit, throwing away any changes you've made. (This is equivalent to choosing **ABORT** from the inspector window as described in Section 2.)

Choosing **DONE** from this attached menu indicates that you're done tinkering with card parts and wish to return to the main interaction window. Normally, this causes the card inspector

menu to close and the phase 1 process outlined in Section 2 to be performed again. Thus the data area will be rescoured and new statistics on the health of your notefile will be printed. This cycle of scour data area (phase 1) followed by inspect (phase 2) can be repeated as often as desired. Eventually, you must either abort, end the Inspect&Repair process gracefully, or continue to phase 3. Because phase 1 can be quite slow for large uncompact notefiles, there is one optimization: if you've made no changes in phase 2, then the data area scouring is not repeated in phase 1. Rather, the old information from the last scouring is recovered and used instead.

If you've clicked DONE, but there are still cards with bad prop lists, titles, or links (because you haven't either deleted them or backed up their card parts to previous versions), you will be asked up to three questions. The list of card IDs of cards with bad prop lists is printed out and you're asked whether it's okay to set the prop lists of those cards to NIL. Then, the list of card IDs of cards with bad titles is printed out and you're asked whether it's okay to set the titles of those cards to "Untitled." Finally, card IDs for cards with bad links are printed out and you're asked whether it's okay to set the global links to nil. (Global links will be rebuilt as much as possible in phase 3. See Sections 1.5 and 4.) If you want phase 3 to be able to run, it is necessary to either answer yes to these questions or fix each of the bad card parts by hand in phase 2.

3.1 The card inspector menu

In the card inspector menu, those IDs corresponding to deleted cards have a line drawn through them. Those having some sort of problem appear shaded. In addition, an upper-case letter suffix is attached to such IDs indicating the problem. For example, a shaded menu item NC00023SL indicates that ID NC00023 has bad substance and bad links. The letter codes are S, L, P, and T indicating bad substance, links, property list, and title, respectively. If such a letter code appears in lower case, then the indication is that the current version of that card part is beyond the last checkpoint pointer. For example, NC00023t indicates that NC00023's current title was changed since the last checkpoint. (There may have been a crash, for example, thus preventing the notefile from closing normally.)

In addition to menu entries for each card ID, there is also one entry labeled LNTYPES allowing you to inspect (and possibly back up to a previous version) the Link Types for this notefile.

If you button an ID in the card inspector menu, then a popup menu allows up to two choices **Inspect** and/or **Delete**. If the card is currently deleted (has a line drawn through it), then the Delete option is replaced by **Undelete**. Certain card IDs cannot be deleted and thus their popup menus only contain the Inspect option. These are the top level file boxes NC00001, NC00002, NC00003. The link types menu entry LNTYPES also does not allow deletion.

Choosing Delete or Undelete from this popup menu causes the card to be deleted or undeleted, respectively and the line through the menu item either drawn or undrawn. Note, however, that this action (and all others) can be undone by choosing ABORT from either the card inspector menu or the interaction window menu.

Choosing Inspect from the popup menu for a card ID entry brings up a card parts inspector for that card.

3.2 The card parts inspector

Figure 2 below shows an example of a card parts inspector. It is composed of four attached menus arranged vertically and one attached operations menu atop the stack.

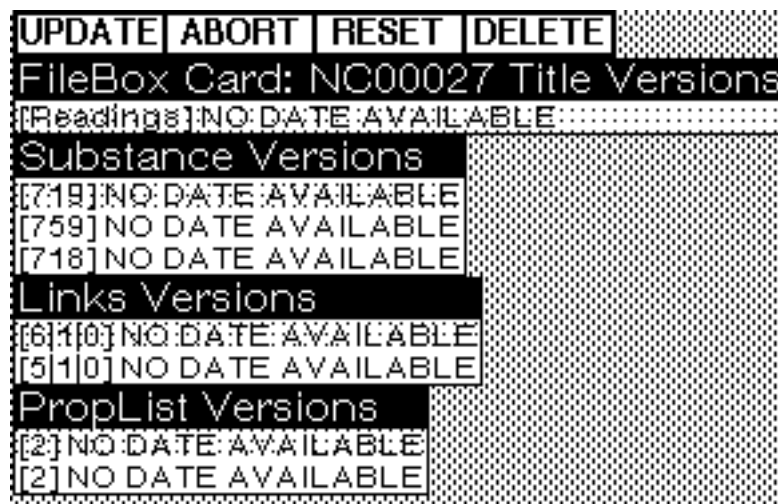


Figure 2: A card parts inspector

The four menus contain entries for every valid version of card parts for the card with ID NC00027. The top menu is for version of titles and below that are menus for versions of the card's substance, links, and prop list. For example, the Substance submenu contains entries for three versions of the substance of this card. The current version of each card part is shaded. Each menu entry gives the date that that version was written if available or the string "NO DATE AVAILABLE" if there is no date on the notefile. (The latter is the case for old notefiles prior to the time we began recording card parts write dates.)

If the current version of the card part is bad, then the menu entry will be a string so indicating, for example, "BADSUBSTANCE."

The title of the top menu includes the card's type and ID. In addition, each menu item contains a bit of information, in square brackets, before the date. In the title versions menu, this information is the first few characters of the title. In the substance versions menu, it is the number of bytes in the substance. In the links versions menu, it is a triple of numbers giving the number of to links, from links, and global links for this card. Finally, the proplist versions menu includes the number of entries on the property list for this card (i.e. twice the number of attribute-value pairs).

Atop the stack of menus is an attached menu of operations, described below.

ABORT: This aborts this card parts inspector, throwing away any changes made.

UPDATE: This closes the card parts inspector, effecting any changes (backing up to previous versions of card parts) you might have done.

DELETE: This option closes the card parts inspector and deletes the card. (Again, this can be undone by choosing ABORT from the card inspector menu.)

UNDELETE: For cards that have been deleted, this option appears instead of DELETE. Choosing it causes the card to be undeleted.

RESET: This causes the selections in the submenus to be restored to the values they had when the card parts inspector was first brought up. (Equivalent to doing **ABORT** and then inspecting this ID again from the card inspector menu.)

Note that cards that can't be deleted don't have the **DELETE** option on their card parts inspector.

Buttoning an entry in a submenu of a card parts inspector pops up a short menu unless the entry is for a bad version (e.g. "BADSUBSTANCE"). This menu contains at least the entry **Inspect** and possibly **Change Selection**, if the selected entry is not the same as the current one (i.e. not shaded).

Choosing **Inspect** allows further inspection of the details of the selected card part version. For example, inspecting a title version brings up the Interlisp inspector on a record containing the title, date and card ID. Similarly, inspecting a links or prop list version brings up the Interlisp inspector on a record containing the lists of links (for to links, from links and global links) or the prop list. Note that if you wish to continue inspecting a links version down to the single link level, choose to inspect the link as a **NOTECARDLINK**. This is somewhat more communicative about record field names. Note also that changing values out in the Interlisp inspectors has no affect on the notefile and is ignored.

All substance versions for cards having substance types **TEXT**, **SKETCH** and **GRAPH** can currently be inspected. (This includes all cards except those having user-defined substance types, like the **NCFile** card.) Inspecting a card's substance version will bring up a window showing a copy of the substance. (Note that changes to this copy have no affect on the notefile.) Any links in the substance of the card will show up as bracketed strings describing the link.

Choosing **Change Selection** from the card part version popup menu causes the current selection to be changed, thus backing up the card to the selected version. (This change can be undone by resetting or aborting the card parts inspector as well as by later aborting the card inspector or interaction window.)

4. Running the Notefile inspector: Phase 3: Rebuilding your links

To complete the **Inspect&Repair** process, select the **Continue Inspect&Repair** option from the interaction window menu. This invokes phase 3, the links rebuilder. Normally, this simply rebuilds links from card substances (see Section 1.5). In certain circumstances, it may do extra work as well. If your link types list is bad, and you didn't back it up to a previous version, then phase 3 will rebuild it. If there are fileboxes with bad substances that you haven't either deleted or backed up to previous versions, then phase 3 will rebuild them. Finally, if there are cards with bad links that you haven't backed up or deleted, then phase 3 will rebuild those links as well. (It rebuilds ALL to links and from links anyway. For those cards, it will rebuild global links as well.) Again, for details, see Section 1.5.

5. Tips and hints for using **Inspect&Repair**

This section contains a list of strategies and tips for using **Inspect&Repair**. For the most part, they are ordered from the useful and obvious to the esoteric. Several of these are implicit in the first four sections of the document, but are repeated here for emphasis and completeness.

When in doubt, abort! All your changes will be lost, but then if you're uncertain about what's happened this is the safe course. Often, in fact, you may simply want to check the health of your notefile and abort without tinkering.

Fixing versus tinkering. There are two main ways to use the inspector, either for fixing a broken notefile, or tinkering with a healthy one. The latter case occurs when you wish to recover some card that you inadvertently deleted. Or back up a card that you inadvertently changed to a previous version.

Compacting. Old versions of card parts have always been stored in notefiles, but up till now have been inaccessible. Thus, there was little reason not to compact your notefile often. Now there is a tradeoff between the need to save space by compacting versus the need to be able to back up using Inspect&Repair. Probably the safest course is to keep a backed up copy of the pre-compacted notefile around until you have confidence that the compacted one is healthy and that you have no need for previous versions of any of its cards.

Inspect&Repair can't run when notefile is open. This means that if you are working in your notefile and notice a card you'd like to inspect a previous version of, you must record the card's ID and close the notefile. Then, run Inspect&Repair, find the card ID in the inspector menu and tinker with it as desired.

Fixing enough problems to allow phase 3 to run. You can't run phase 3 unless Inspect&Repair thinks your notefile is above a certain threshold of health. There are certain problems it can handle (e.g. bad filebox substances, see Sections 1.5 and 4), and others that it can't (e.g. bad title). You have to decide either to fix these sorts of problems yourself in phase 2, let phase 3 attempt to rebuild them, or just abort the whole thing (always an option).

Sometimes these decisions can be tricky. For example, suppose a filebox's substance is bad. Call it BadBox. Should you (a) delete BadBox altogether, (b) back its substance up to a previous version, or (c) allow phase 3 to rebuild it by looking for from links in other cards from BadBox? Option (c) may not be advisable if there was important text in BadBox or if the order of cards in BadBox was important. On the other hand, option (b) may be of little use if the last good version is too out of date (or if there is no good version at all).

NoteCards Programmer's Interface

Release 1.2i

Randy Trigg

Xerox PARC

Updated version: 18-Mar-85

Modified: 26-Aug-85 by Randy Trigg

Modified: 1-Aug-85 by Lissa Monty

Introduction

This document describes a facility whereby users with some programming know-how can obtain a lisp interface to NoteCards. In this way, they can create and modify Notefiles, cards and links under program control.

The functions described below are divided into 7 groups:

1. NoteFile Creation and Access
2. Creating and Accessing NoteCard Types
3. Creating NoteCards and FileBoxes
4. Accessing NoteCards and FileBoxes
5. Creating and Accessing Links
6. Creating and Accessing Link Labels
7. Handy Miscellaneous Functions

1. NoteFile Creation and Access

For each of the following functions (except NCP.CloseNoteFile), the argument is a filename. The suffix ".NoteFile" is added if not already present. In any case, the filename used by NoteCards always has this suffix.

(NCP.CreateNoteFile <filename>)

If <filename> is not already a notefile, then create a notefile <filename>.NoteFile, and return this filename which can later be passed to NCP.OpenNoteFile.

(NCP.OpenNoteFile <filename> <don'tCreateFlg> <convertw/oConfirmFlg>)

If there is no currently open notefile, then open <filename> and make it the currently active NoteFile. Returns resultant stream if successful, else nil. If <don'tCreateFlg> is non-nil, then a new file will not be created if the given one doesn't exist. If <convertw/oConfirmFlg> is non-nil, then if needed, the file will be converted to release1.1 format without user confirmation.

(NCP.CloseNoteFile [<stream>])

Closes <stream> if it corresponds to a currently open Notefile. Returns its filename if successful. If <stream> is nil, then closes current open notefile.

(NCP.CheckpointSession)

Checkpoint the current Notecards session, first writing out any dirty cards. In case of a system crash or abort, the notefile can be recovered to the last checkpoint. Note that closing a notefile does a checkpoint.

(NCP.AbortSession)

Abort the current Notecards session, losing all work since last checkpoint or successful close.

(NCP.RepairNoteFile <filename>)

Rebuilds the link structure of <filename>. It must *not* be currently open.

(NCP.CompactNoteFile <filename>)

Copies <filename> to a later version, recovering space. Must not be open.

(NCP.CompactNoteFileInPlace <filename>)

Compacts <filename> in place, replacing the old version. Must not be open.

(NCP.DeleteNoteFile <filename>)

Removes the <filename> notefile. Must not be open.

(NCP.CurrentNoteFileStream)

Returns the currently open notefile stream if there is one, else nil.

(NCP.CurrentNoteFile)

Returns the full name of the currently active notefile if there is one, else nil.

(NCP.CheckOutNoteFile <fromFilename> <toFilename>)

Copies <fromFilename> to <toFilename> unless <fromFilename> is locked. If successful, creates a lock file in <fromFilename>'s directory. The name of the lock file is formed by concatenating the atom LOCKFILE onto <fromFilename>.

(NCP.CheckInNoteFile <fromFilename> <toFilename>)

Check lock file for <toFilename>. If none, then just copy <fromFilename> to <toFilename>. If there is one and it's owned by us, then do the copy and remove the lock file. If there is a lock file owned by someone else or if date of <toFilename> is more recent than date of lock file, then print a message and do nothing.

2. Creating and Accessing NoteCard Types

These functions give the user access to the NoteCard user-defined types facility. For an explanation of this facility, see the NoteCards Types Mechanism documentation.

(NCP.CardTypes)

(NCP.SubstanceTypes)

Returns lists of all currently defined NoteCard types and substances, respectively.

(NCP.CreateCardType <TypeName> <SuperType> <SubstanceType> <FnsAssocList> <VarsAssocList>)

Makes a new NoteCard type with name <TypeName>, super type <SuperType>, substance <SubstanceType>. Any functions not appearing in <FnsAssocList> will be inherited from <SuperType>. The CardWidth and CardHeight vars fields will be inherited if not specified in <VarsAssocList>. Other vars fields default to nil. Note that, for now, specializing the FileBox card type is not allowed.

(NCP.CreateSubstanceType <SubstanceName> <FnsAssocList> <VarsAssocList>)

Makes a new substance type with name <SubstanceName> and the given functions and vars fields. None of the function fields should be nil (but might conceivably be the function NILL).

(NCP.CardTypeSuper <type>)

Returns the super type of <type>.

(NCP.CardTypeSubstance <type>)

Returns <type>'s substance type.

(NCP.CardTypeLinkDisplayMode <type>)

Returns the link display mode of <type>.

(NCP.CardTypeFn <type> <fn>)

(NCP.CardTypeVar <type> <var>)

Returns the <fn> (<var>) field for <type>.

(NCP.CardTypeInheritedField <type> <field>)

Returns the value of the card type function or variable <field> for <type>. This is possibly different from the value returned by NCP.CardTypeFn or NCP.CardTypeVar in that if the defined value for <field> of <type> is nil, then the super is checked for a non-nil value. This checking continues until either a non-nil <field> is found or we reach the top of the super hierarchy. In that case, the value of <type>'s substance's <field> is used. Note that among the variable fields, only CardDefaultWidth and CardDefaultHeight inherit, so for the other Var fields, the result of NCP.CardTypeVar is valid (even if it's nil).

(NCP.SubstanceTypeFn <substance> <fn>)

(NCP.SubstanceTypeVar <substance> <var>)

Returns the <fn> (<var>) field for the substance <substance>.

(NCP.ValidCardType <type>)

Returns non-nil if <type> is an existing NoteCard type.

(NCP.ValidSubstanceType <substance>)

Returns non-nil if <type> is an existing NoteCard substance type.

(NCP.ValidCardTypeFn <fn>)

(NCP.ValidCardTypeVar <var>)

Returns non-nil if <fn> (<var>) is a valid function (variable) field for NoteCard types, for example, the litatom MakeCardFn (CardDefaultWidth). In other words, <fn> (<var>) can serve as the <fn> (<var>) arg to NCP.CardTypeFn (NCP.CardTypeVar).

(NCP.ValidSubstanceTypeFn <fn>)

(NCP.ValidSubstanceTypeVar <var>)

These return non-nil if <fn> (<var>) is a valid function (variable) field for substance types. In other words, <fn> (<var>) can serve as the <fn> (<var>) arg to NCP.SubstanceTypeFn (NCP.SubstanceTypeVar).

(NCP.CardTypeFns)

(NCP.CardTypeVars)

(NCP.SubstanceTypeFns)

(NCP.SubstanceTypeVars)

These return lists of all valid Fn (Var) fields for NoteCard types and substances respectively.

3. Creating NoteCards and FileBoxes

The following functions create various sorts of cards and boxes within the currently open notefile.

(NCP.CreateTextCard <title> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns a new notecard having type Text. If <title> is non-nil, it is installed as the Notecard's title, otherwise the title is "Untitled." <props>, if non-nil, should be a prop-list of properties and values to be placed on the user property list of the Notecard. If <parentfileboxes> is non-nil, then it should be a list of FileBoxes in which to initially file this card.

(NCP.CreateFileBox <title> <nodisplayflg> <props> <childcardsboxes> <parentfileboxes>)

Creates and returns a new Filebox with title <title> (or a gensym'ed name if <title> is nil). It will initially contain child cards and boxes from the list <childcardsboxes> (if that arg is non-nil). If <parentfileboxes> is nil, then the new filebox will be filed in the value of (NCP.GetToBeFiledFileBox). The <props> arg is handled as it was for NCP.CreateNoteCard.

(NCP.CreateBrowserCard <title> <paramList> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns a new browser card with given title, props and parents. <paramList> should be a prop list of browser parameters. The properties currently recognized are:

ROOTCARDS: A list of Notecards to serve as roots of the forest or lattice generated by the browser. If omitted or NIL then user is asked to choose root cards.

LINKTYPES: A list of link types to follow when creating the browser. Any label present in the list having the backarrow prefix ("_") represents that link type but in the reverse direction. This list can also contain the atoms ALL or _ALL in which case browsing will be done on all links in either the forward or reverse direction. If both ALL and _ALL are specified, then links in both directions will be used (generally making a mess).

DEPTH: The depth at which to cut off the browser. This should be a non-negative integer. If NIL or omitted, then will assume no limit. (Currently integers greater than 9 are assumed equivalent to infinity.)

FORMAT: This should be a list of one, two or three elements. The first should be an atom indicating grapher format. The choices are FAST (laid out as a forest, sacrificing screen space for speed), COMPACT (laid out as a forest, using minimal screen space), LATTICE (laid out as a directed acyclic graph, the default), *GRAPH* (laid out as a graph, i.e. virtual nodes are eliminated). The second element of the FORMAT list, if present, should be either HORIZONTAL (the default) or VERTICAL specifying whether the graph is laid on its side or up and down. The third element, if present, should be the atom REVERSE. This indicates that horizontal graphs should be laid out from right to left instead of left to right and that vertical graphs should be laid out from bottom to top rather than vice versa.

(NCP.CreateSketchCard <title> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns an initially empty sketch/map card having given title, props, and parents.

(NCP.CreateGraphCard <title> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns an initially empty graph card having given title, props, and parents.

(NCP.CreateCard <type> <title> <nodisplayflg> <props> <parentfileboxes> <otherargs>)

Creates and returns a card of the given (possibly user-defined) type, with given title, props, and parents. <otherargs> is a possibly nil list of args that will be passed to the MakeCardFn of <type>. Card is initially displayed or not according to value of <nodisplayflg>.

(NCP.MakeDocument <rootcard> <parametersProplist> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns a Document card starting from <rootcard>. The default set of parameters for making documents can be accessed via NCP.DocumentParameters, but some of these can be given new values just for the duration of this MakeDocument by specifying a non-nil <parametersProplist>. For example, a value of '(TitlesFromNoteCards Bold ExpandEmbeddedLinks ALL)' for <parametersProplist> would cause temporary changes to the values of the parameters TitlesFromNoteCards and ExpandEmbeddedLinks. As usual, the resulting card will have the given props and parents.

(NCP.MakeLinkIndex <linktypes> <backpointersP> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns a LinkIndex text card consisting of a sorted record of all instances of links in the current notefile having one of the given link types. <linktypes> can contain the litatoms ALL and/or _ALL as well as any particular backwards links. (See the above description of NCP.MakeDocument.) Backpointer links are inserted in the text if <backpointersP> is non-nil. Resulting card will have given props and parents.

4. Accessing NoteCards and FileBoxes

The following functions provide access to the cards and boxes present in the current notefile. Note that whether a card's window has been brought up on the screen has little or no effect on the following functions. If the user changes some field of a card while that card is visible on the screen, then the field will update itself automatically. Thus, users can switch between program-driven and screen-interface-driven modes at will.

Cards can be active or inactive. An active card has its information cached (on its property list) thus saving time at the expense of memory. All cards visible on the screen are active. Most of the following functions leave the card in the same state as it was when they started (except NCP.BringUpCard, which makes it active). Thus, users needing to do several consecutive operations to the same card should consider temporarily caching the card's information via NCP.ActivateCards (and then uncache with NCP.DeactivateCards).

Most of the following functions take as first argument a card or filebox. If this does not in fact correspond to an existing card or box, then an error message is printed and nil is returned.

(NCP.BringUpCard <card> <region/position>)

Brings up on the screen the given card in the given region or at the given position. If <region/position> is nil, then user is asked to specify position with mouse.

(NCP.ActiveCardP <card>)

Returns non-nil if given card or box is currently active (i.e. information is currently cached in memory).

(NCP.ActivateCards <cardList>)

For each card or box in <cardList> (or just the one, if the argument is atomic), make it active (i.e. cache its information in memory).

(NCP.DeactivateCards <cardList>)

For each card or box in <cardList> (or just the one, if the argument is atomic), make it inactive (i.e. uncache its information back into the file). If any cards in <cardList> were on the screen then this will close their windows.

(NCP.CardType <card>)

Returns the type of <card> or NIL if the card does not exist.

(NCP.ValidCard <card>)

Returns non-nil if <card> exists (hasn't been deleted). (This is currently a synonym for NCP.CardType.)

(NCP.CardTitle <card> [<newtitle>])

Returns old title of <card>. If <newtitle> is present, then set <card>'s title to <newtitle>. <newtitle> can be an atom or string. Note, however, that all titles are converted internally to strings by NoteCards.

(NCP.FileCards <cards> <fileboxes>)

Every card or box in <cards> is filed in every box in <fileboxes>. Either arg may be an atom or a list.

(NCP.UnfileCards <cards> <fileboxes>)

Every card or box in <cards> is unfiled from every box in <fileboxes>. Furthermore if <cards> is the litatom ALL, then the boxes in <fileboxes> will be cleared of all children. Similarly, if <fileboxes> is the litatom ALL, then the cards and boxes in <cards> will be unfiled from all their parent boxes. Either arg may be an atom or a list.

(NCP.CardParents <card>)

Returns list of fileboxes in which <card> has been filed.

(NCP.FileBoxChildren <filebox>)

Returns list of children of <filebox> in the order in which they appear in the box's textstream.

(NCP.GetLinks <cards> <destinationCards> <labels>)

Returns list of all links from any of <cards> to any of <destinationCards> having any label in <labels>. Any of these arguments can be nil. For example, if <destinationCards> is nil, then all links pointing from <cards> to anywhere with a label in <labels> are returned. If both <cards> and <destinationCards> are nil, then this returns all links having a label in <labels>. If all three args are nil, then this is a slow synonym for NCP.AllLinks.

(NCP.CardPropList <card>)

Returns the prop list of the given card.

(NCP.CardProp <card> <propname> [<newvalue>])

Returns old value of property <propname> on <card>'s prop list. If <newvalue> is present, then set <card>'s <propname> property to <newvalue>. (Semantics are analogous to the Interlisp function WINDOWPROP.)

(NCP.CardAddProp <card> <propname> <newitem>)

Adds <newitem> to the list present on the <propname> property of <card>. Returns old value of property. (Semantics are analogous to WINDOWADDPROP.)

(NCP.CardDelProp <card> <propname> <itemToDelete>)

Deletes <itemToDelete> from the <propname> property of <card> if it is there, returning the previous value of that property. If not there, return nil. (Semantics are analogous to WINDOWDELPROP.)

(NCP.CardSubstance <card>)

Returns the substance of <card>. This is a textstream in the case that the type of <card> has TEXT substance. Otherwise, it is the appropriate underlying structure if <card> has GRAPH or SKETCH substance.

(NCP.CardRegion <card>)

Returns the region of <card>. This works even if <card> is not currently up on the screen, since the region information is stored on the notefile.

(NCP.CardAddText <card> <textstr> <loc>)

Adds the text within the string <textstr> to the text card <card>. If <loc> is the litatom START or END, then the text will be placed at the start or end of the card respectively. If <loc> is a number, then it is assumed to be a character count within the card at which to place the new text. If <loc> is NIL, then the text is placed at the current cursor location.

(NCP.ChangeLoc <card> <loc>)

Changes the cursor's location in <card>'s textstream to <loc>. Possible values for <loc> are as described for NCP.CardAddText.

(NCP.DeleteCards <cards>)

Deletes the given cards and fileboxes from the current notefile, or deletes just the one if <cards> is atomic.

(NCP.FileBoxP <card>)

Returns non-nil if <card> is a filebox.

(NCP.AllCards)

Returns a list of all extant cards for the current notefile.

(NCP.AllBoxes)

Returns a list of all fileboxes in the current notefile.

(NCP.MapCards <fn>)

Maps down the set of all cards in the current notefile, applying <fn> to each.

(NCP.MapBoxes <fn>)

Maps down the set of all fileboxes in the current notefile, applying <fn> to each.

(NCP.GetContentsFileBox)

(NCP.GetOrphansFileBox)

(NCP.GetToBeFiledFileBox)

These functions retrieve the three predefined FileBoxes for the currently open NoteFile. These boxes can be modified (but not deleted) by the user in the same way as any other filebox.

5. Creating and Accessing Links

Links can be connected to points within a card or to the card as a whole, thus the following four link creation functions are provided. Those that connect to points within a card specify at least one of <fromloc> or <to loc>. If nil, then the link icon is placed at the current cursor location in the card. If the arg is the litatom START or END, then it is placed at the front or end of the text respectively. If the loc arg is a number, then it is assumed to be a character count at which to place the link icon.

(NCP.GlobalGlobalLink <label> <sourceCard> <destinationCard>)

Creates and returns a new link with label <label>, connecting <sourceCard> to <destinationCard>.

(NCP.LocalGlobalLink <label> <sourceCard> <destinationCard> <fromloc> <displaymode>)

Creates and returns a new link with label <label>, connecting from <fromloc> of <sourceCard> card to <destinationCard>. If <displaymode> is non-nil, then the new link is displayed in the given mode. Otherwise the default displaymode for the source card's type is used.

(NCP.GlobalLocalLink <label> <sourceCard> <destinationCard> <toloc>)

Not implemented at this time.

(NCP.LocalLocalLink <label> <sourceCard> <destinationCard> <fromloc> <toloc>)

Not implemented at this time.

(NCP.LinkDesc <link>)

Returns list of three items (<label> <sourceDesc> <destinationDesc>) where <label> is the link type and <sourceDesc> and <destinationDesc> have the form (<anchor mode> <card> <loc>). <anchor mode> is either LOCAL or GLOBAL, <card> is the card at this end of the link, and <loc> gives a position in the text of <card> if <anchor type> is LOCAL and <card>'s substance's type is TEXT.

(NCP.LinkDisplayMode <link> [<newdisplaymode>])

Returns old display mode of <link>. If <newdisplaymode> is present, then set <link>'s displaymode accordingly. If non-nil, it can be one of the litatoms Icon, Title, Label, or Both. Or it can be an instance of the LINKDISPLAYMODE record. This has the 3 fields SHOWTITLEFLG, SHOWLINKTYPEFLG, and ATTACHBITMAPFLG. Each field can have one of the three values T, NIL, or FLOAT. If a field, say SHOWTITLEFLG, has value FLOAT then the corresponding global parameter (DefaultLinkIconShowTitle, in this case) will be consulted to decide whether or not to display the destination card's title in this icon. (See Section 7 for a description of the global parameters.)

(NCP.LinkLabel <link> [<newlabel>])

Returns old label of <link>. If <newlabel> is present, set <link>'s label to <newlabel>.

(NCP.GetLinkSource <link>)

Returns the card at the source end of <link>.

(NCP.GetLinkDestination <link>)

Returns the card at the destination end of <link>.

(NCP.DeleteLinks <links>)

Removes all links in <links> (or the single one if <links> is atomic).

(NCP.ValidLink <link>)

Returns non-nil if <link> is a link in the current notefile.

(NCP.AllLinks)

Returns a list of all existing links in the current notefile. (This is equivalent to but faster than (NCP.GetLinks NIL NIL NIL).)

(NCP.MapLinks <fn>)

Maps down the set of all links in the current notefile, applying <fn> to each one.

6. Creating and Accessing Link Labels

The following functions allow the user to manipulate link labels.

(NCP.CreateLinkLabel <label>)

Creates a new link label with name <label> for current notefile unless there is already one defined by that name.

(NCP.DeleteLinkLabel <label>)

Deletes the link label <label> from the current notefile. The label must exist and must not be the label of any existing link, and it must not be a system-defined link label (e.g. SubBox or FiledCard).

(NCP.RenameLinkLabel <label> <newlabel>)

Changes any links having label <label> to have label <newlabel>. <label> must exist and neither <label> nor <newlabel> should be a system-defined label.

(NCP.GetLinkLabels)

Returns a list of all existing link labels including system-defined ones.

(NCP.GetReverseLinkLabels)

Returns a list of the reverse labels for every existing link label. Thus, whereas SubBox would appear in the list returned by NCP.GetLinkLabels, _SubBox would appear in the list returned by NCP.GetReverseLinkLabels.

(NCP.GetUserLinkLabels)

Returns a list of all existing user-defined link labels.

(NCP.ValidLinkLabel <label>)

Returns non-nil if <label> is a defined link label for current notefile.

7. Handy Miscellaneous Functions

(NCP.TitleSearch <key> <key> ...)

Returns a list of all cards having all of the <key>s (can be atoms, numbers or strings) within their titles.

(NCP.PropSearch <propOrPair> <propOrPair> ...)

Returns a list of all cards such that for every <propOrPair> arg, if it is atomic, then the card must contain that property. If it is a list of two elements, then the card must have a property EQ to the first element with value EQ to the second element.

(NCP.WhichCard <x> <y>)

Returns the card currently displayed on the screen whose window contains the position in screen coordinates of <x> if <x> is a POSITION, the position (<x>,<y>) if <x> and <y> are numbers, or the position of the cursor if <x> is NIL. Returns NIL if the coordinates are not in the window of any card. If they are in the window of more than one card, then returns the uppermost. If <x> is a window, then NCP.WhichCard will return the card associated with that window.

(NCP.CardFromWindow <window>)

Returns the card associated with <window>, or NIL if not a notecards window.

(NCP.CardWindow <card>)

Returns <card>'s window if <card> is currently displayed somewhere on the screen.

(NCP.SelectCards)

Returns a list of those cards selected from the screen. A menu appears near the top of the screen with buttons for "DONE" and "CANCEL". Selections are made by left buttoning in the title bars of the desired cards.

(NCP.DocumentParameters <parametersProplist>)

Returns the old value of the document parameters in the form of a proplist. If <parametersProplist> is non-nil then it should be a proplist whose properties are (some of the) valid document parameter names and whose values are permissible values for those parameters. The valid parameters and possible values are as follows:

HeadingsFromFileboxes: NumberedBold, UnnumberedBold, NONE.

TitlesFromNoteCards: Bold, NotBold, NONE.

BuildBackpointers: ToCardsBoxes, ToCards, ToBoxes, NONE.

CopyEmbeddedLinks: ALL, NONE, <listOfLinkLabels>.

ExpandEmbeddedLinks: ALL, NONE, <listOfLinkLabels>.

[See the Notecards user's manual for an explanation of these parameters and how their values affect the document created.]

(NCP.NoteCardsParameters <parametersProplist>)

Returns the old value of the global Notecards parameters in the form of a proplist. If <parametersProplist> is non-nil then it should be a proplist whose properties are (some of the) valid document parameter names and whose values are permissible values for those parameters. The valid parameters and possible values are as follows:

DefaultCardType: <legalCardType>

FixedTopLevelMenu: T or NIL

ShortWindowMenus: T or NIL

ForceSources: T or NIL

ForceFiling: T or NIL

ForceTitles: T or NIL

CloseCardsOffScreen: T or NIL

MarkersInFileBoxes: T or NIL

AlphabetizedFileBoxChildren: T or NIL

DefaultLinkIconAttachBitmap: T or NIL

DefaultLinkIconShowTitle: T or NIL

DefaultLinkIconShowLinkType: T or NIL

LinkDashingInBrowsers: T or NIL

ArrowHeadsInBrowsers: one of the litatoms {AtEndpoint, AtMidpoint, None}

SpecialBrowserSpecs: T or NIL

AnnoAccessible: T or NIL

EnableBravoToTEditConversion: T or NIL

DefaultFont: a font

LinkIconFont: a font

Here, <legalCardType> should be an existing Notecard type, i.e. one that appears in the list returned by NCP.CardTypes.

(NCP.PrintMsg <window> <clearFirstFlg> <arg1> <arg2> ...)

Prints a message in the prompt window of <window>. If <window> is NIL, then prints message in the Lisp prompt window. If <clearFirstFlg> is non-nil, then clears the prompt window first. The args are PRIN1'ed one at a time.

(NCP.ClearMsg <window> <closePromptWinFlg>)

Clears the prompt window associated with <window> (or with the main Lisp prompt window if <window> is NIL) and closes it if <closePromptWinFlg> is non-nil.

**(NCP.AskUser <Msg> <prompt> <FirstTry> <ClearFirstFlg> <MainWindow>
<DontCloseAtEndFlg> <DontClearAtEndFlg> <PROMPTFORWORDFlg>)**

This function can be used to ask questions of the user in a window's prompt window. The <Msg> and <prompt> are printed along with <FirstTry> (if non-nil). The value returned is whatever the user types. If <ClearFirstFlg> is non-nil, then the prompt window is cleared first. If <MainWindow> is nil, then the top level prompt window is used. If <DontCloseAtEndFlg> is non-nil, then the prompt window won't be closed after the question is answered and if <DontClearAtEndFlg> is non-nil, then the prompt window won't be cleared at the end. If <PROMPTFORWORDFlg> is non-nil, then the PROMPTFORWORD typein protocol will be used rather than TTYIN. The former doesn't allow mouse editing of the string typed in. On the other hand, typing automatically overwrites the prompt when PROMPTFORWORD is used.

(NCP.AddTitleBarMenuItems <Win> <NewMenuItems>)

Adds the given menu items to the left button title bar menu of <Win>. <Win> should be the window of a visible notecard.

(NCP.GetDates <Card>)

Returns a NOTECARDDATES record structure containing the dates of last modification of each of the four card parts of <Card>. The fields of the record are SUBSTANCEDATE, TITLEDATE, LINKSDATE and PROPLISTDATE.

Notice of release of Notecards 1.2i

The 1.2i Intermezzo release of Notecards is hereby officially released.

To run NoteCards, load onto an Intermezzo sysout the file {qv}<notecards>release1.2i>notecards.dcom. As usual, send bug reports by choosing "NoteCards Report" from the Lafite middle button send mail menu. Mail of more general interest to the NoteCards community should be sent to NoteCards↑.pa.

Even if you have been using 1.2i for some time and feel comfortable with Notecards, please take a look at the release notes in {qv}<notecards>release1.2i>doc>ReleaseNotes.ted.

The release notes describe in detail the changes since 1.1. These include incorporation of the latest version of sketch, new functionality in the browser, a new notefile inspect and repair facility, several new library packages, and many other feature additions and bug fixes.

Also on {qv}<notecards>release1.2i>doc> you can find updated documentation on the programmer's interface, ProgIntFace.ted, and a new document describing the inspect and repair facility, NoteFileInspector.ted.

The library packages and accompanying documentation can be found in {qv}<notecards>release1.2i>library>.

Enjoy!

- Randy

NoteCards Release1.2i Announcement

Xerox Corporation

Randy Trigg
Frank Halasz

[Location: {qv}<notecards>release1.2i>notecards.dcom]
[First written: 3/27/85 Randy Trigg]
[Last updated: 8/26/85 Randy Trigg]

This document updates the NoteCards Release1.1 User's Manual, describing changes and new features for Release1.2i. As usual, send bug reports to NoteCardsSupport.pa (or use the Lafite SendMail middle button menu) and matters of more global interest to Notecards↑.pa.

You must be in Intermezzo to run NoteCards Release1.2i. From now on, you can depend on the letter suffix following the release number to indicate the appropriate version of Interlisp.

Changes from 1.1 are mostly in the following areas: the NoteCards browser, notefiles interface, link icon display and user interface. In addition, there are various miscellaneous changes, a couple of new card types, and fixes of several outstanding 1.1 bugs.

1. Operating on a Notefile.

Checkpointing and aborting a session:

A fundamental change was made to the way Notecards updates its working notefile that allows 1.2i users to checkpoint their work, abort a session (losing work since the last checkpoint), and recover more gracefully from crashes. First, a word about the way Notecards notefiles are structured.

A notefile consists of two parts, an index area and a data area. The index includes for each notecard, several pointers into the data area. There are separate pointers for the notecard's substance, title, prop list, and links. When, say, a notecard's title is changed, the new title is written at the end of the data area (in fact the end of the file) and the index pointer is changed. In Release1.1 (and earlier), the index modifications happened out on the file as they occurred. Now, in Release1.2i they happen in an in-core array and are not written to the file till checkpoint (or close) time. In addition, there is a checkpoint pointer that points to the end of file at the time of the last checkpoint or close. New data (such as a new title) is still written to the file, but always at the end of the file. Thus if a crash occurs and later the notefile is reopened, Notecards can notice the extra data beyond the checkpoint pointer and truncate the file at that point (if you confirm).

More concretely, there are now two new NoteFile Ops menu entries: "Checkpoint Session" and "Abort Session." Checkpointing causes any active cards to have their contents saved to the notefile (but not closed), the index array to be written back out to the file, and the checkpoint pointer to be reset to the end of the file. (Note that closing a notefile automatically does a CheckpointSession.) Aborting a session causes Notecards to close down, discarding all work since the last checkpoint or close.

When a notefile is opened, the checkpoint pointer is compared with the end of file pointer. If they don't agree, then you're asked whether the file should be truncated. You're also given the option of saving the extra work since the last checkpoint to a file. If valuable cards were created (or modifications made) since the last checkpoint, then you should answer yes and provide the name of a file in which to store the truncated information.

Next, you should open the truncated notefile and bring up a separate TEdit window on the file containing the truncated information. Though TEdit formatting information is lost, you can recover a card's text by browsing this file. (Note that scrolling from back to front will retrieve the most recent version of each card.)

[Note that closing (or saving without closing) a card writes it out to the file, but does not force the index to be updated. Thus, if crashes are anticipated, do CheckpointSession often.]

Compacting a Notefile:

Because Notecards never actually overwrites any information in the data area of a notefile, it is necessary to periodically compact the notefile. This facility has been improved in Release1.2i in two ways. It is now possible to specify a target file name for the compaction (rather than always going to the same name), and it is now possible to compact a notefile in place. These two choices form a submenu of the CompactNotefile entry in the Notefile Ops menu.

Copying, restoring, and backing up notefiles:

The menu entries for RestoreFromFloppy and BackupToFloppy have been removed from the NotefileOps menu. In their place is a general CopyNotefile option. It prompts you for source and target file names for the copy.

There is a new facility for checking in and out notefiles using locks for multiple users sharing a notefile. Still in the experimental stages, it must be called via the programmer's interface. See the programmer's interface documentation.

Inspecting and healing broken notefiles:

The old Repair option on the Notefile Ops menu is now called Inspect&Repair and has been improved considerably. Before rebuilding the links of your notefile, it reads the entire data area looking for good card parts (including outdated and deleted versions). It then allows you to delete and/or back up card parts to previous versions. All this is done interactively through a menu driven interface. Only when the notefile is deemed healthy are you allowed to perform the link rebuilding. For details on the operation of Inspect&Repair, see the document titled NotefileInspector.ted.

2. Changes to the Notecards user interface.

Stylesheets:

Several places in Notecards now use Tayloe Stansbury's stylesheet package for user interaction, in particular, changing a link's display mode, a browser's specs, or the default text and link icon fonts from the global parameters menu. Stylesheets allow packaging of several menus together with "buttons" governing individual menus and the stylesheet as a whole. Menus within a stylesheet can optionally allow multiple selections. All stylesheets have three global buttons "Done," "Reset," and "Abort." "Done" causes the new values to be accepted. "Reset" causes the

original values (when the stylesheet was entered) to be recovered. "Abort" causes the stylesheet to be exited without changing any values. Menus allowing multiple selections also have the buttons "All" and "Clear" attached. "All" causes all values in the menu to be selected while "Clear" unselects the entire menu. Toggling of menu entries is accomplished by left clicking the entry.

New global parameters:

The top level global parameters menu has several new additions. These (as well as some old 1.1 ones) are described below. To change the value of a global parameter, click on the variable name. The value will toggle between "Yes" and "No" if binary, and allow selection from an appropriate menu otherwise.

ForceSources, ForceFiling, ForceTitles: These dictate whether to bother you at card closing time about incomplete information for the card. If ForceFiling is set, for example, then you are asked to designate parent fileboxes of the card before closing. Similarly, for sources and titles. If ForceFiling is off (value is "No"), then cards without parents will be filed automatically in the ToBeFiled filebox at closing time. If ForceTitles is off, then an untitled card will be left with the title "Untitled." ForceTitles and ForceFiling default to "Yes," while ForceSources defaults to "No."

CloseCardsOffScreen: If "Yes," then when a card is closed, it is first dragged off screen so that the close happens invisibly.

MarkersInFileBoxes: If "Yes," then new fileboxes will contain the markers "FILE BOXES" and "NOTE CARDS." New child boxes are inserted under the FILE BOXES marker and new child cards under the NOTE CARDS marker. If "No," then new fileboxes come up without markers and new children are inserted at the current cursor position. Note that regardless of the MarkersInFileBoxes setting, if a filebox has no markers (because you've deleted them) then new children are inserted at the cursor position.

AlphabetizedFileBoxChildren: If "Yes," then new fileboxes will have the property OrderingFn set to NC.IDAlphaOrder. This will cause any new cards put into such a filebox to be inserted in alphabetical order. For further details on OrderingFn's for fileboxes see Section 4.

DefaultLinkIconAttachBitmap, DefaultLinkIconShowTitle, DefaultLinkIconShowLinkType: These dictate the manner in which link icons are displayed if not currently specified in the icon. There are three fields of a link's display mode that can be set, unset, or floated independently. If a field is floated, then the global parameter for that field is consulted. For example, if a link icon's display mode has value FLOAT for the ShowTitle field, then whether the title gets shown inside the link icon depends on the value of DefaultLinkIconShowTitle. See below for a further description of a link's display mode.

LinkDashingInBrowsers: If "Yes," then browser links are drawn with dashed lines with the dashing style corresponding to the link's type. See Section 3 for further details on browser changes. Defaults to "No."

ArrowHeadsInBrowsers: This dictates whether arrow heads are drawn on browser links. The variable can be set to either "AtMidpoint," "AtEndpoint," or "None." See Section 3 for details. Defaults to "None."

EnableBravoToTEditConversion: If "Yes" then TEdit checks when getting a file whether that file is in Bravo format and if so, converts. This defaults to "No" for efficiency.

DefaultFont: This dictates the font that new text cards default to.

LinkIconFont: This dictates the font for text appearing in link icons.

Link icon display mode:

The display mode of a link icon can be changed by middle buttoning in the icon and selecting from the three menus in the resulting stylesheet. These are: AttachBitmap, ShowTitle, and ShowLinkType. AttachBitmap, if "Yes," causes link icons to be shown with a bitmap representing the type of the destination card attached at the left. ShowTitle and ShowLinkType, if "Yes," cause the link icon to contain the title of the destination card and/or the link type. Any of the three fields can have the value FLOAT, in which case the appropriate global parameter will be consulted. (See description of global parameters above.) If all fields are set to "No" (or the floating ones inherit No from the global parameters), then a small, uninformative icon is used to display the link.

"Pushing" and "Pulling" link icons:

There are now two ways to move or copy a link icon between cards or within a card. "Pulling" works like TEdit shift-select. That is, to move an icon, put the cursor where you want to move to and hold down the shift key (or shift and ctrl keys) while left clicking in the left or right quarter of the icon. The new style is called "pushing" and is done by holding down the shift key while left clicking in the middle part of the icon. Then move the cross-hairs cursor to the icon's new home and left-click. To abort a "push," just left click in the background. Note that "pushing" currently only works for copying, not moving.

Specifying notefile names and card titles:

A different editor has been incorporated into Notecards for obtaining card titles, file names, etc. This editor is the same one used in the top level lisp exec window (TTYIN). Thus you can change the title (or file name) given as prompt via mouse edits.

3. Changes to the Notecards browser.

Multiple roots:

Browsers can now contain multiple roots, in which case the graph will be laid out as a forest.

Dashed links:

Dashed browser links was a rarely used option in Release1.1, largely because of speed considerations. The speed of drawing dashed links has improved in Release1.2i by taking advantage of improvements in Grapher. There are currently nine different dashing styles possible. If a browser contains instances of more than nine different link types, then the last dashing style will be used repeatedly for each link type beyond the ninth. As before, link dashing is a user-settable option in the GlobalParameters menu (see Section 2).

Arrowheads:

Arrowheads can now be drawn on browser links. These show the direction of the notecards link being represented in the browser. This is a user-settable parameter in the GlobalParameters

menu with possible values AtMidpoint, AtEndpoint, or None. If AtMidpoint or AtEndpoint is specified, then arrowheads will be drawn at link midpoints or endpoints, respectively. However, in either case, if two browser nodes are connected by more than one link, then any arrowheads for those links will appear at the midpoints (so as not to overlap).

Browser specs:

Whereas in Release1.1 only the link types to traverse could be specified, in Release1.2i, link types is one of a number of browser specs. Also included are browser depth, format, and orientation. These are accessible through a BrowserSpecs stylesheet, a collection of 5 menus. For general details on the stylesheet interface see Section 2. In this case, the forward and backward link types menus are multi-selectable, that is, more than one entry can be chosen. The other three menus are used to make single selections.

Forward and backward link types function as in Release1.1. That is, the browser will contain only nodes for cards reachable from the root cards by following forward links in "line of direction" or backward links in "reverse line of direction."

Browser depth is chosen from a menu containing entries for the integers 0 through 9 and INF (or infinite depth). The default is INF, meaning that the browser will not be cut off until there are no more links to follow from leaf nodes. Choosing depth 0 means that only the root nodes will appear (and no links).

Browser format is one of *GRAPH*, LATTICE, COMPACT, or FAST. The latter three are provided by the grapher package and correspond to lattice, compact forest and fast forest, respectively. COMPACT and FAST generate virtual nodes (in double boxes) whenever two or more links would be drawn to the same node. LATTICE only generates virtual nodes when a cycle exists in the graph. *GRAPH* is a new format that never generates virtual nodes. The drawback to using *GRAPH* is that a cycle can cause lines to be drawn that cross boxes or overlap other lines. Thus you may have to move nodes around for legibility after computing the browser. The default is LATTICE.

Browser orientation is one of Horizontal, Vertical, Reverse/Horizontal, or Reverse/Vertical. These specify whether the graph is layed out left-to-right, top-to-bottom, right-to-left, or bottom-to-top, respectively. The default is Horizontal.

New middle button title bar menu options:

Several new entries have been added to the middle button menu invoked from a browser's title bar. The options are now RecomputeBrowser, RelayGraph, ReconnectNodes, UnconnectNodes, ExpandBrowserNode, GraphEditMenu, and ChangeBrowserSpecs.

RecomputeBrowser causes the current contents of the browser to be thrown away and recomputed as in Release1.1. However, in Release1.2i, you can optionally specify a new set of root nodes.

RelayGraph does not rebuild the graph, but rather causes the nodes and links of the graph to be repositioned on the screen (using Grapher's LAYOUTGRAPH). This will destroy any work you have done moving nodes within the graph.

ReconnectNodes first causes any link edges in the graph to be erased. (Note, however, that non-link edges, those created by "AddEdge" as described below, are ignored.) Then, each node in the

graph is connected to every other node in the graph for which there is a link between them having one of the currently selected link types. This can be useful for several reasons:

1. when the linking structure between cards has changed, but the current browser layout needs to be preserved.
2. when some browser nodes need to be moved, but dragging the connected links is too slow. In this case, do `UnconnectNodes` followed by `ReconnectNodes` (after you've moved the nodes around).
3. when special browser layouts are desired. For example, suppose you like the layout that Grapher gives you when certain links are left out or when you limit the depth. Then calling `ReconnectNodes` will fill in the missing links without affecting the graph's layout.

UnconnectNodes simply erases all edges in the browser. This is useful for positioning a browser's nodes before invoking `ReconnectNodes`.

ExpandBrowserNode allows you to enlarge the graph under a given node. After selecting a node, you're asked for a depth (defaults to 1). The graph is then expanded under the selected node to the given depth, following any currently selected links. Note that `ExpandBrowserNode` calls `LAYOUTGRAPH` so any existing special node arrangements will be lost.

GraphEditMenu brings up the graph editing menu. See the description below.

ChangeBrowserSpecs brings up the `BrowserSpecs` stylesheet to allow you to change any of the browser specs. These changes will be noticed at the next `RecomputeBrowser`, `ExpandBrowserNode`, etc.

Editing the browser manually and "structure editing":

The browser can be edited through the use of the `GraphEditMenu`. This menu can be obtained either by right-buttoning in the browser window or by choosing `GraphEditMenu` from the title bar middle button menu. The `GraphEditMenu` includes options for "structure editing"; that is, changing underlying `NoteCards` structure by editing the browser. The old options for editing without changing structure are also present. Given below are the menu items in `GraphEditMenu` and the actions they engender.

CreateCard&Node causes a new card to be created in the current Notefile and a corresponding node for it to be included in the browser. You're asked for the type of the new card, its title, and where to position the node representing it.

CreateLink&Edge causes a new link to be created between two existing cards and a corresponding edge to be drawn in the browser. (We call such an edge representing a Notecards link, a "link edge." See `AddEdge` below for creating non-link edges.) You're asked for the "From" and "To" nodes in the browser corresponding to the cards to be linked as well as a link type. The link icon for the new link is positioned at the cursor point in the From card if the card has text substance and an open window. Text cards with closed windows have links inserted at the start of the text stream. Otherwise, the new link is a global link. You can have multiple link edges between pairs of cards. In this case the edges are displayed in a spline or "flower" arrangement.

DeleteCard&Node causes a card to be deleted and its corresponding node in the browser to be removed. You are asked first to choose the node representing the card to be deleted and then to confirm the removal of the node (type "y" to confirm) and the deletion of the card. If the selected node is one of a set of virtual nodes (double boxed), then all nodes in the set (i.e. representing the given card) are removed.

DeleteLink&Edge causes a link in the Notefile to be deleted and the corresponding edge in the browser to be removed. You first pick the "From" and "To" nodes corresponding to the source and destination ends of the link respectively. Then, if there is only one link between those two cards, the link is deleted after user confirms. If there are multiple links between the two cards, then the user chooses from a menu of link types.

AddLabel puts a "label node" into the browser that does not represent a Notecard. You are prompted for a string forming the node's label and then must position the label node. This node is not boxed. (But note that "virtual" label nodes can be boxed and thus can be confused with non-virtual regular nodes.)

AddNode adds a node into the browser corresponding to some existing card. You are asked to point to a card (title bar or link icon) on the screen that this node is to represent and then to position the node.

AddEdge draws a line between two nodes in the browser. This edge does not correspond to a real link in the Notefile. To avoid confusion, it is best to have the arrowheads option on (see Section ??) in this case, since edges formed by AddEdge do not have arrowheads (or dashing). Only one such edge is allowed between any two nodes and none if there are already link edges between the nodes. Thus doing CreateLink&Edge will remove any existing non-link edge.

RemoveNode removes a node from the browser. It does not delete the card (if any) that the node represents. Edges into and out of the node are also removed. If the selected node is one of a set of virtual nodes representing the same card, then you will be told how many nodes will be removed with this one and will be asked to confirm. The only way to remove only one node of a set of virtual nodes, is to first manually remove edges into and out of it using RemoveEdge. Then RemoveNode can be used to remove only the one virtual node.

RemoveEdge removes an edge from the browser. It does not delete the link (if any) that the edge represents. The user is asked to select the "From" and "To" nodes of the edge.

MoveNode allows you to change the position of any node, rubber banding any edges pointing to it. You're asked to point to the node by left-buttoning, and holding down the left button, drag the node to its new position.

LabelSmaller is used to decrease the font size of label nodes. Note that it does not work for regular non-label nodes.

LabelLarger is used to increase the font size of label nodes.

<->Shade toggles the shade of a node between black-on-white and white-on-black. This can only be performed on label nodes (not on nodes representing Notecards).

FIX MENU causes the GraphEditMenu to be affixed to the lower right edge of the browser window. Note that this does not prevent you from obtaining the menu via right button inside the window.

[Note that the above editing commands do not work on old 1.1 browsers. Such browsers should either be recomputed (via `RecomputeBrowser`) or unconnected and reconnected.]

4. Miscellaneous changes.

Links ordering within text cards:

The internal list of outgoing links in a text card is now kept in the same order that the links appear in the card's text. This means, for example, that the daughters of a browser node for a filebox will appear in the correct order.

Link insertion:

The title bar menu entry for "InsertLinks" now has an attached submenu containing entries for adding single links, multiple links, and global links. When inserting multiple links (or adding multiple global links) you're only asked for one link type which is used to label all the new links and all are inserted at the same place in the text.

Show links:

This is now a normal entry in the left button title bar menu of a card (rather than a subentry under Edit Properties). The format of the ShowLinks display has been changed slightly. The prefix is now either TO, FROM, or Global TO. The link type is shown in the icon. Also, for text cards, the TO links should appear in the correct order.

Sketch changes and fixes:

Notecards now uses the latest version of sketch. See the sketch documentation for details on changes. Several long-standing bugs having to do with link icons in sketch cards have been fixed.

Sketches and graphs in text cards:

It is possible to shift-copy the contents of sketch and graph/browser cards into text cards. In addition, the Document card is now able to include the contents of sketch and graph cards if encountering them during card gathering. (It is still not possible for Document to include the contents of cards having user-defined substance types such as NCFFile cards.)

Data saved at card closing:

When a card is closed, only those parts that are dirty are written out to the notefile. A message indicating which parts are being saved is now printed to the card's prompt window during closing. Furthermore, certain card types (in particular, browsers) were saving their substance even if no changes were made. This source of space inefficiency has been fixed in Release1.2i.

Ordering cards in a filebox:

It is now possible to dictate the relative placement of new cards in a filebox. If the `OrderingFn` property of a card has a value, it should be a lisp function that takes two card ID arguments and returns T if the first should appear before the second and NIL otherwise. You can make such a function appear automatically on new boxes for the case of alphabetizing by using the global parameter `AlphabetizeFileBoxChildren`. See section 2.

Programmer's interface:

The Programmer's interface has been updated. Thus users with existing programmer's interface code should read the revised PI documentation. The changes are not all forward compatible.

Notecards system date:

You can find the date of your Notecards system in the variable NC.SystemDate. The 'NewestFile' property on the NC.SystemDate atom contains the name of the last modified Notecards file.

The Notecards library packages:

The old Release1.1 library packages have been converted to 1.2i and documented and several new ones have been added. These can be found on {qv}<notecards>release1.2>library> and include NCScreen, NCCluster, NCChain, NCFileCard, NCKeys, NCHacks, and ARIDemo. Documentation can be found in <filename>.ted.

NCScreen defines several handy functions for arranging cards on the screen callable from the programmer's interface. NCCluster defines several new card types, most notably CaseCluster, a cluster of cards for use in the sample domain of legal case analysis. NCChain defines the Chain card type, useful for breaking up a large text card into a linked chain of cards. NCFileCard defines the new File card type and FILE substance allowing a notefile to link to external files via standard Notecards links. NCKeys provides a shorthand language for invoking various handy programmer's interface functions. NCHacks contains several handy functions written using the programmer's interface. Two of these allow global text searches and replaces throughout a notefile. In addition there is a function that searches by last card modification date and one that links cards to form chains. Finally, ARIDemo is an example of how the programmer's interface can be used to construct notefiles that demo themselves.

Loading NoteCards from different directories:

NoteCards now uses the values of four directories variables to decide from whence to load the code. These are NOTECARDSDIRECTORIES, NOTECARDSMAPDIRECTORIES, QUADTREEDIRECTORY, and MAPFILEDIRECTORY. They default to ({QV}<NOTECARDS>RELEASE1.2I>), ({QV}<NOTECARDS>MAPS>NEW>), {QV}<NOTECARDS>MAPS>, and {QV}<NOTECARDS>MAPS> respectively.

5. Known bugs and plans for future improvements:

- o The compactor should check first for available space.
- o There are major speed problems in redrawing large browsers. Changing link display mode could also use some streamlining.
- o Integrate the document compiler and the types mechanism so that instances of new card types can be sucked into TEdit documents.
- o Make links into full-fledged objects having properties and type hierarchies.

The NoteCards Types Mechanism

Release 1.2

Frank G. Halasz

Xerox PARC

First Written: 22-Mar-85

Modified: 26-Mar-85 by Frank G. Halasz

Modified: 1-Aug-85 by Lissa Monty

1. Introduction

The NoteCards types mechanism allows a user with some knowledge of Interlisp to add new types of note cards to the system. The types mechanism is built around an inheritance hierarchy of note card types. If the user needs to create a new card type that is a small change from an already existing card type, he or she need only define the few functions or parameters that account for the differences between the new card and the existing card. However, if the user wishes to create a totally new type of card, then he or she must define the 20-odd functions and parameters that make up a note card type.

Every note card has a substance. A substance is essentially a data structure that contains the information in the note card. Different types of note cards have different types of substances. Associated with every substance type is an editor that can be used to create and/or modify the data structure of that substance type. For example, the substance of a Text card is a TEXTSTREAM that can be edited using TEdit. Similarly, the substance of a Browser card is a GRAPH record that can be edited using GRAPHER. Defining a new note card type involves specifying the functions necessary to handle the card's substance and its editor.

1.1 The Inheritance Hierarchy

The inheritance hierarchy in NoteCards has two parts: a tree of NoteCardTypes and a list of SubstanceTypes. Every NoteCardType has a super-type and a substance type. The super-type is an already existing NoteCardType from which the NoteCardType will inherit fields. Thus, the set of NoteCardTypes forms a tree structure based on the super-type field. The substance type of a NoteCardType is an already existing SubstanceType.

The inheritance process for a given field of a NoteCardType works as follows: if the field has a non-NIL value in the NoteCardType then this value is used, otherwise the field value is inherited from its super-type. If there are no non-NIL values anywhere in the inheritance path for the NoteCardType, then the field value is taken from the corresponding field in the substance type for the NoteCardType. Substance types are guaranteed to have values in all of their fields.

Example: ProtectedText is a card with super-type Text. Text in turn has super-type NoteCard (the null root of the NoteCardType tree). In addition, Text has substance type TEXT. If an EditCardFn is not defined in ProtectedText, then it will be inherited from Text. If Text doesn't have an EditCardFn then the EditSubstanceFn from the TEXT SubstanceType will be used (since NoteCard by definition does not have an EditCardFn).

Functions are inherited all or none. Often, however, a new NoteCardType will require only a minor addition to the corresponding function of its super-type. In this case, the new card type should define a new function, but this function can call the corresponding function of its super-type to do the bulk of the work. The following construction will accomplish this goal:

```
(APPLY* (NCP.CardTypeInheritedField (NCP.CardTypeSuper <type>) <fn>) <arg1>
<arg2> ...)
```

where <type> is the TypeName of the card type in question, <fn> is the name of the function in question, and <arg1> <arg2> ... are the arguments to that function. For example the following might be the definition of the EditCardFn for the passworded Text card called ProtectedText:

```
(DEFINEQ
(NC.EditProtectedTextCard
  (LAMBDA (ID Substance Region/Position)
    (* * Edit a Protected Text card, asking for the password first.)
    (PROG (Password Result)
      (* * Get this card's password from the prop list)
      (SETQ Password (NCP.CardProp ID (QUOTE Password)))
      (COND
        ((EQUAL Password (NC.GetPassword ID))
          (* Password is okay.
            Call the EditCardFn of my super-type)
          (SETQ Result (APPLY*
            (NCP.CardTypeInheritedField
              (NCP.CardTypeSuper (QUOTE ProtectedText))
              (QUOTE EditCardFn))
            ID Substance Region/Position)))
        (T (* Password is bad. Express condolences)
          (NCP.PrintMsg Window T "Sorry." (CHARACTER 13)
            "You do not know the password!!"
            (CHARACTER 13)
            "Bye."
            (CHARACTER 13))
          (DISMISS 2000)))
      (RETURN Result))))
```

1.2 Links and Link Icons

An integral part of NoteCards is the ability to create a link between two note cards. Presently, there are two kinds of links: *GlobalToGlobal* links and *LocalToGlobal* links. GlobalToGlobal links connect one entire card with another entire card and are stored separately from either card's substance.

GlobalToGlobal links are maintained (almost) entirely by the NoteCards system code and therefore do not vary across note card types.

LocalToGlobal links connect a particular position within the substance of one card (the *source* card) to the entirety of the other card (the *destination* card). Within the source card, the link is represented by an image object called a link icon that must be contained by the card's substance. Since substances vary across note card types, the handling of link icons varies across note card types. The destination (or Global) end of a LocalToGlobal links is maintained by the NoteCards system code.

Not all note cards can be the source of LocalToGlobal links. Card types that support LocalToGlobalLinks must have their LinkAnchorModesSupported parameter set to T. If a this parameter has any other value, then cards of this type can be the source of only GlobalToGlobal links. These Global-links-only card types need to provide only one piece of functionality in support of the linking mechanism. In particular, they must provide user access to the function **NCP.GlobalGlobalLink** from the editor that runs when the card is being displayed. For example, the editor's command menu might include an "Insert Global Link" command. All other link maintenance is carried out by the NoteCards system.

If a card type supports LocalToGlobal links, then it must contain the necessary mechanisms for supporting link icons in its substance. Link icons are instances of standard Interlisp-D image objects (See documentation of Image Objects in Interlisp-D). The mechanisms supporting link icons include functions for inserting, deleting, updating, and collecting the link icons contained in a card's substance. These functions are described in detail below. In addition to these functions, a note card type supporting LocalToGlobal links must provide user access to the function **NCP.LocalGlobalLink** from the editor that runs when the card is being displayed. In addition the editor must provide user access to the function **NCP.GlobalGlobalLink**.

Inside the link icon image object is a link record containing all of the information about the link. These link records can be manipulated using the link manipulation functions provided by NoteCards' programmer's interface (e.g., **NCP.GetLinkDestination** returns the destination field of a link record). The functions required to define a note card or substance type deal in both link records and link icons. You can translate between these two representations using the functions **NC.MakeLinkIcon** and **NC.FetchLinkFromLinkIcon**; **NC.MakeLinkIcon** will create a link icon image object from a link record, while **NC.FetchLinkFromLinkIcon** will return the link record contained in a link icon.

1.3 Using the Types Mechanism

Most uses of the types mechanism involve defining new NoteCardTypes. Usually, these new NoteCardTypes involve specifying a TypeName, a SuperType, a SubstanceType, and one or two functions that differ from the SuperType. The most commonly defined functions are the MakeCardFn, the EditCardFn and the QuitCardFn.

Definition of new substance types occurs only when a new kind of substance (e.g., a spreadsheet) and its corresponding editor are to be added to the system. When defining a substance, all of its fields must be fully defined since there is no inheritance among SubstanceTypes.

2. The NoteCardType

Each note card type in the system is defined by a record structure (i.e., a NoteCardType) containing about 20 names, functions and parameters. The functions implement behaviors that are required by the NoteCards system but vary across the different card types. For example, one function is responsible for writing the card's substance to the NoteFile. The parameters represent specifications that inform NoteCards about the specific properties of each card type, e.g., whether it handles local links or not.

The NoteCardTypes are organized into an inheritance hierarchy. Each NoteCardType has a super-type. If any of the functions or parameters is not specified for a given NoteCardType, that function or parameter is inherited from its super-type (or its super-type's super-type, if the function or parameter is not specified for the super-type either). Each NoteCardType also has a SubstanceType. If any of the functions or parameters cannot be found along the super-type chain of the NoteCardType, then the card type inherits the function or parameter from its SubstanceType.

Overall, a card type is a data structure with the following 21 fields:

Inheritance Hierarchy Specifications

- 1) TypeName
- 2) SuperType
- 3) SubstanceType

Functions

- 4) MakeCardFn
- 5) EditCardFn
- 6) QuitCardFn
- 7) GetCardFn
- 8) PutCardFn
- 9) CopyCardFn
- 10) MarkCardDirtyFn
- 11) CardDirtyPFn
- 12) CollectLinksInCardFn
- 13) DeleteLinksInCardFn
- 14) UpdateLinkIconsInCardFn
- 15) InsertLinkInCardFn
- 16) TranslateWindowPositionToCardPositionFn

Parameters

- 17) LinkDisplayMode
- 18) CardDefaultWidth
- 19) CardDefaultHeight
- 20) CardLinkAnchorModesSupported
- 21) CardDisplayedInMenuFlg

These fields are defined as follows:

- 1. TypeName:** The atom that is the name of this card type. TypeNames must be unique among the NoteCardTypes tree, though they may overlap with SubstanceNames. The convention is that NoteCardType TypeNames have only the first letter capitalized. This is to set them apart from SubstanceNames which are by convention all caps.
- 2. SuperType:** The TypeName of the NoteCardType that is the super-type for this NoteCardType. When a new NoteCardType is created, its SuperType must be an existing NoteCardType.
- 3. SubstanceType:** The SubstanceName for the substance of this card type. When a new card is created, its SubstanceType must be the name of an existing SubstanceType (see Section 3.0 below). The basic NoteCards system includes the following substance types: TEXT, SKETCH, GRAPH which represent the substances handled by the TEdit, Sketch, and Grapher packages respectively.
- 4. MakeCardFn:** The name of a function to be applied to an ID, a Title, and a NoDisplayFlg. The function should create a new card of this type. The ID is the note card ID that will be assigned to the newly created card. It should be used to set the various properties of the new card. The title is a string specifying the title of the new card. It can be used in messages to the user or to set the title of any windows created. NoDisplayFlg determines whether the new card is to be displayed on the screen or not. If NoDisplayFlg is non-NIL, then the card is to be displayed in a window on the screen. If NoDisplayFlg is NIL, then the card is to be created but not displayed on the screen.

The MakeCardFn should return the window of the new card if NoDisplayFlg is non-NIL and the ID if NoDisplayFlg is NIL.

Before returning, every MakeCardFn is required to set the substance property of ID by calling (**NC.SetSubstance** ID *Substance*) where *Substance* is whatever is considered a substance for this card type. For example, a TextStream for Text cards, a Graph record for Graph cards, or a Sketch record for Sketch cards.

By convention, every MakeCardFn sets the SHRINKFN of any window it creates to the function **NC.ShrinkFn** using WINDOWPROP.

- 5. EditCardFn:** The name of a function to be applied to ID, Substance, and Region/Position. The function should start an editor for the given card. ID is the note card ID of the card. Substance is the substance of the card; it will be a thing of whatever type is considered a substance for this card type, e.g., a TextStream or Sketch record. Region/Position is a Region or a Position on the screen that specifies where the card is to be placed. (**NC.DetermineDisplayRegion** ID Region/Position) is a function that will determine the exact region for the card's window given the ID and the Region/Position.

The EditCardFn should return the editor window.

The EditCardFn is responsible for checking to make there is not already an editor for card ID already on the screen. If there is, the EditCardFn should just flash the previous editor window.

By convention an EditCardFn sets the SHRINKFN of any window it creates to the function **NC.ShrinkFn** using WINDOWPROP. Also by convention, an EditCardFn should set the title of the editor window to be the value of (**NCP.CardTitle** ID).

6. QuitCardFn: The name of a function to be applied to WindowOrSubstanceOrID which is either the editor window for a card or the substance of a card or a note card ID. QuitCardFn should quit out of the editor currently operative on the specified card and close the window containing the card.

The value returned by QuitCardFn is unspecified.

Before returning the QuitCardFn should apply the function **NC.DeactivateCard** to the ID of the card. Note that the ID may have to be computed from the Window or Substance passed to the QuitCardFn. The function **NC.CoerceToID** will do this computation.

The QuitCardFn should also insure that all processes related to this card are completed (or guaranteed to eventually complete) before returning.

7. GetCardFn: The name of a function to be applied to the DatabaseStream, a card ID, and a screen Region. The GetCardFn should read the substance of the note card specified by ID from the DatabaseStream. The format of the data to be read is determined by the PutCardFn (see below). When the GetCardFn is called, the file pointer for DatabaseStream is positioned on the first byte of the data to be read.

The GetCardFn should return a pointer to the substance read from the DatabaseStream.

GetCardFn need produce no side-effects. The ID and the Region are for reference purposes only.

Note that the GetCardFn need only read the substance of the card, i.e., that information about the card which is specific to its card type. General information about a card such as its title, its property list, its list of links, etc. is read from the DatabaseStream by the system.

8. PutCardFn: The name of a function to be applied to a note card ID and the DatabaseStream. The PutCardFn should write the substance of the note card specified by ID to the DatabaseStream. When the PutCardFn is called, the file pointer for DatabaseStream is positioned at the first byte assigned to the card. When the PutCardFn returns, the file pointer should be positioned immediately after the last byte written.

The format for writing the card's substance is fairly unrestricted. The data written on the DatabaseStream can take up any number of bytes, but the bytes must be contiguous. It must be written so that it can be recovered by reading from the DatabaseStream using the GetCardFn. The only other restriction is that the first 6 bytes of the substance must contain the file position of the start and the end of the substance: 3 bytes for the start file pointer and 3 bytes for the end file pointer. These pointers are for use by the CopyCardFn.

The value returned by the PutCardFn is unspecified.

Note that the PutCardFn need only write out the substance of the card, i.e., that information about the card which is specific to its card type. General information about a card such as its title, its property list, its list of links, etc. is written to the DatabaseStream by the system.

9. CopyCardFn: The name of a function to be applied to a note card ID, a "from" DatabaseStream, and a "to" DatabaseStream. The CopyCardFn should copy the substance for the note card specified by ID from the "from" DatabaseStream to the "to" DatabaseStream. When the CopyCardFn is called the file pointer for the "from" DatabaseStream is positioned on the first byte of the data to be copied. The file pointer for the "to" DatabaseStream is positioned at the first byte of the space assigned to the card on the "to" DatabaseStream.

The format for writing the substance on the "to" DatabaseStream has the same restrictions as for the PutCardFn.

Most often the CopyCardFn is a simple COPYBYTES that uses the start and end pointers written by PutCardFn in the first 6 bytes of the substance. Note, however, that all file absolute pointers (including the start and end pointers) must be updated; the file location on the "to" DatabaseStream is almost never the same as the original file location on the "from" DatabaseStream.

The value returned by the CopyCardFn is unspecified.

The CopyCardFn is used primarily by the compactor that eliminates "dead" space in the database. Thus, it is important that the CopyCardFn be as time efficient as possible.

10. MarkCardDirtyFn: The name of a function to be applied to a note card ID and a ResetFlg. If the ResetFlg is non-NIL, the function should mark the card specified by ID as being dirty (i.e., changed since it was last written to the DatabaseStream). If the ResetFlg is NIL, the function should reset the "dirtiness" of the card.

The MarkCardDirtyFn is called by NoteCards system functions that change the card. It is not necessarily called by user operations inside the editor on the card. Therefore, it is best if the mechanism used by the MarkCardDirtyFn is somehow coordinated with the corresponding mechanism used by the editor on the card. (See the CardDirtyPFn below.)

The value returned by the MarkCardDirtyFn is unspecified.

The card specified by ID is guaranteed to be active.

11. CardDirtyPFn: The name of a function to be applied to a note card ID. The function should return a non-NIL value if the card specified by ID is dirty, i.e., if it was changed since it was last written to the DatabaseStream. NIL should be returned otherwise.

Note that a "dirty" card is one that has been changed in any way. Only NoteCards specific changes to a card will result in a call to the card's MarkCardDirtyFn. Changes made through the editor on the card will use the editors "mark dirty" mechanism and will not call the MarkCardDirtyFn. Therefore, the CardDirtyPFn should check all dirty flags, i.e., the dirty flag set by the MarkCardDirtyFn as well as any set by the card's editor.

The card specified by ID is guaranteed to be active.

12. CollectLinksInCardFn: The name of a function to be applied to a note card ID, a CheckAndDeleteFlg, a DatabaseStream, a ReturnLinkIconsFlg, and a ReturnLocationsFlg. The function should examine the substance of the card specified by ID and produce a list of the links (or link icons) contained by the substance. The ReturnLinkIconsFlg and the ReturnLocationsFlg determine the contents of the list to be returned as follows:

ReturnLinkIconsFlg and ReturnLocationsFlg both NIL: the list to be returned should be a list of link records.

ReturnLinkIconsFlg is non-NIL, ReturnLocationsFlg is NIL: the list to be returned should be a list of link icons.

ReturnLinkIconsFlg is NIL, ReturnLocationsFlg is non-NIL: the list to be returned should be a list of pairs where the first member of the pair is a link record and the second member of the pair is the "location" of the link icon for that link inside the substance.

ReturnLinkIconsFlg and ReturnLocationsFlg both non-NIL: the list to be returned should be a list of pairs where the first member of the pair is a link icon and the second member of the pair is the "location" of that link icon.

If CheckAndDeleteFlg is non-NIL, then the list produced by CollectLinksInCardFn should contain valid links only. Any links found to be invalid should be deleted. To check the validity of a link, the function **NC.ValidLinkP** should be applied to the link record and the DatabaseStream. To delete a link, apply the function **NC.MakeInvalidLink** to the link icon.

The CollectLinksInCardFn should return the list produced CONSed to a dirty flag. The dirty flag should be non-NIL if any links were deleted, NIL otherwise.

The card specified by ID is guaranteed to be active.

13. DeleteLinksInCardFn: The name of a function to be applied to a "source" note card ID and a link record or "destination" note card ID. If the second argument is a link, the function should

remove from the substance of the card specified by "source" ID the link icon corresponding to link. If the second argument is a "destination" note card ID, the function should remove from the substance of the card specified by "source" ID all link icons corresponding to links pointing to the card specified by "destination" ID.

To "remove" a link icon, the link icon should be replaced in the substance by the image object that is the value of **NC.DeletedLinkImageObject**. Note that before deleting the link icon, it is best to replace the IMAGEOBJFNS of the link icon with the value of **NC.NoDeleteImageFns**. This will prevent the link icon's WHENDELETEDFN from being activated when the deletion takes place.

The value returned by the DeleteLinksInCardFn is unspecified.

The card specified by "source" ID is guaranteed to be active.

14. UpdateLinkIconsInCardFn: The name of a function to be applied to a "source" note card ID or window and a "destination" note card ID. The function should update (i.e., force a redisplay of) all link icons in the "source" card that represent links pointing to the "destination" card. This function is called when some property of the link is changed by the NoteCards code. It is also called when certain properties of the destination card (e.g., its title) are changed.

The value returned by the UpdateLinkIconsInCardFn is unspecified.

The "source" card is guaranteed to be active.

15. InsertLinkInCardFn: The name of a function to be applied to a window, a link, and a position. The function should insert a link icon containing the link into the card being edited in the window at the position specified. The position is whatever object is returned by the TranslateWindowPositionToCardPositionFn.

The value returned by the InsertLinkInCardFn is unspecified.

The ID of the card being edited by the window is guaranteed to be the SOURCEID of the link.

16. TranslateWindowPositionToCardPositionFn: The name of a function to be applied to a window, an X-coordinate in that window, and a Y-coordinate in that window. The window is an editor window on the substance of some card. The function should return a position object that describes the position in the card substance that is currently located at the given X-Y position in the window. The format of the position object is undefined. It will be passed to the InsertLinkInCardFn and used as the position at which to insert a links in the card being edited in the window.

17. LinkDisplayMode: determines the default display mode for link icons inserted into cards of this type. It must be a record of type LINKDISPLAYMODE. LINKDISPLAYMODE describes what information will be displayed in a link icon. It consists of three flags: SHOWTITLEFLG, SHOWLINKTYPEFLG, and ATTACHBITMAPFLG. If SHOWTITLEFLG is non-NIL, the link icon will display the destination card's title. If SHOWLINKTYPEFLG is non-NIL, the link icon will

display the type of the link. If ATTACHBITMAPFLG is non-NIL, a bit map describing the type of the destination card will be attached to the right of the link icon.

Note: This property is NOT inherited.

18. CardDefaultWidth: The default width for editor windows on cards of this type.

19. CardDefaultHeight: The default height for editor windows on cards of this type.

20. CardLinkAnchorModesSupported: an atom that determines the kind of links this card type will support (i.e., the kind of links for which cards of this type can be a source). If NIL, then this card type does not support links of any type. If Global, this card supports only Global links. If Local, this card supports only local links. If T, this card supports both Global and Local links.

Note: This property is NOT inherited.

21. CardDisplayedInMenuFlg: if non-NIL then this card type will appear in the choice of card types in the menu used during card creation using the "Create" entry in the main NoteCards menu. If NIL, then this card type will not appear in this menu.

3. The SubstanceType

The SubstanceType is a record structure whose fields are virtually identical to those of the NoteCardType record. In particular, the SubstanceType has the following 17 fields:

- 1) SubstanceName
- 2) CreateSubstanceFn
- 3) EditSubstanceFn
- 4) QuitSubstanceFn
- 5) GetSubstanceFn
- 6) PutSubstanceFn
- 7) CopySubstanceFn
- 8) MarkSubstanceDirtyFn
- 9) SubstanceDirtyPFn
- 10) CollectLinksInSubstanceFn
- 11) DeleteLinksInSubstanceFn
- 12) UpdateLinkIconsInSubstanceFn
- 13) InsertLinkInSubstanceFn

14) TranslateWindowPositionToSubstancePositionFn

15) SubstanceDefaultWidth

16) SubstanceDefaultHeight

17) SubstanceLinkAnchorModesSupported

These fields are defined as follows:

1. SubstanceName: The atom that is the name of this substance type. SubstanceNames must be unique among the substance types, though they may overlap with card TypeNames. The convention is that SubstanceNames are all in caps. This is to set them apart from card TypeNames which by convention have only their first letter capitalized.

2 Thru 14. Functions: All of the functions are identical to the corresponding functions in the NoteCardType record structure. Note the (arbitrary) use of "create" instead of "make" in the name of the CreateSubstanceFn.

15 Thru 17. Parameters: The parameters are identical to the corresponding parameters in the NoteCardType data structure. There are no parameters for the LinkDisplayMode and the DisplayInMenuFlg because these two parameters are not inherited. They must be specified separately for each card type.

4. Adding a New NoteCardType or SubstanceType to the System

The functions **NCP.CreateCardType** and **NCP.CreateSubstanceType** can be used to add new Types to the system.

NCP.CreateCardType takes 5 arguments: the TypeName, its SuperType, its SubstanceType, a functions list, and a parameters list. The functions list is an ASSOC list where the CAR of each sub-list is one of the function field names given above (e.g., EditCardFn, MakeCardFn, etc.). The CDR of the sublist should contain the name of the required function. Any function field name for which there is no entry will be set to NIL and will thus be inherited. The parameters list is analogous to the functions list, except that it applies to the parameter field names (i.e., LinkDisplayMode, CardDefaultWidth, CardDefaultHeight, and CardLinkAnchorModesSupported).

NC.CreateSubstanceType takes 3 arguments: the SubstanceName, a functions list, and a parameters list. The functions and parameters list are analogous to those for **NCP.CreateCardType** except that all of the function and parameter fields specified above MUST have an entry in the ASSOC lists.

Both **NCP.CreateCardType** and **NC.CreateSubstanceType** will overwrite existing types (NoteCard and Substance, respectively) of the same name.

5. Example: Defining the ProtectedText NoteCardType

The following is an example of defining a new card type called the ProtectedText card. The card type is created by specifying new MakeCardFn and EditCardFn functions. All other functions are inherited from the super-type, i.e., the Text card. All of the parameters are specified directly for this card.

- The function that creates the new ProtectedText card type:

```
(NC.AddProtectedTextCardType
  (LAMBDA NIL (* fgh: "26-Mar-85 15:48")
    (* * Create the ProtectedText card type)
    (NCP.CreateCardType (QUOTE ProtectedText)
      (QUOTE Text)
      (QUOTE TEXT)
      (QUOTE ((MakeCardFn NC.MakeProtectedTextCard)
        (EditCardFn NC.EditProtectedTextCard)))
      (QUOTE ((LinkDisplayMode (T NIL NIL))
        (CardDefaultHeight 300)
        (CardDefaultWidth 400)
        (CardLinkAnchorModesSupported T)
        (CardDisplayInMenuFlg T))))))
```

- The MakeCardFn for the ProtectedText card type:

```
(NC.MakeProtectedTextCard
  (LAMBDA (ID Title NoDisplayFlg) (* fgh: "26-Mar-85 15:23")
    (* * Make a protected Text card
      by calling the make card fn for a Text card
      and then attaching a password to the card)
    (PROG (Window WindowOrID)
      (* * Create the Text card)
      (SETQ WindowOrID (APPLY*
        (NCP.CardTypeFn (NCP.CardTypeSuper
          (QUOTE ProtectedText))
            (QUOTE MakeCardFn))
          ID Title NoDisplayFlg))
      (* * Get the window for the card, if there is one)
      (SETQ Window (WINDOWP WindowOrID))
      (* * Get the password from the user
        and add it to the cards prop list)
      (NCP.CardProp ID (QUOTE Password)
        (NC.GetPassword Window))
      (* * Return whatever the super-type's MakeCardFn returned)
      (RETURN WindowOrID)))
```

- The EditCardFn for the ProtectedText card type:

```

(NC.EditProtectedTextCard
  (LAMBDA (ID Substance Region/Position)
    (* fgh: "26-Mar-85 17:21")
    (* * Edit a Protected Text card, asking for the password first.)
    (PROG (ExactRegion Window Password Result)
      (* * Open a window for this card)
      (SETQ ExactRegion (NC.DetermineDisplayRegion ID
        Region/Position))
      (SETQ Window (CREATEW ExactRegion))
      (* * Get this card's password from the prop list)
      (SETQ Password (NCP.CardProp ID (QUOTE Password)))
      (COND
        ((EQUAL Password (NC.GetPassword Window))
          (* Password is okay.
            Call the EditCardFn of my super-type)
          (SETQ Result (APPLY*
            (NCP.CardTypeInheritedField
              (NCP.CardTypeSuper (QUOTE ProtectedText))
              (QUOTE EditCardFn))
            ID Substance ExactRegion)))
        (T (* Password is bad. Express condolences)
          (NCP.PrintMsg Window T "Sorry." (CHARACTER
            13)
            "You do not know the password!!"
            (CHARACTER 13)
            "Bye."
            (CHARACTER 13))
          (DISMISS 2000)))
      (* * Close the window you created.
        The super-types EditCardFn will
        have created another window.)
      (CLOSEW Window)
      (RETURN Result))))

```

· A utility used by the MakeCardFn and the EditCardFn:

```

(NC.GetPassword
  (LAMBDA (Window)
    (* fgh: "26-Mar-85 15:50")
    (* * Get a password from the user.
      Window is the main window for the card in question)
    (NCP.AskUser "What is the password for this card?" " -- "
      NIL T Window)))

```

NoteCards

Evolving Outline

Chapter Status

- 4 **Title Page**
- 4 **Preface**
 - Audiences
 - Small selection of article & book references.
- 3 **Table of Contents**

I Introduction & Installation

- 4 1 **Introduction**
 - NoteCards
 - System Overview
 - Useful SunOS and UNIX Conventions
 - NoteCards Device Conventions
 - Stylistic Conventions
 - Prompts
 - Font Usage
 - Keyboard Conventions
- 4 2 **System Requirements**
 - Prerequisites
 - Processor Hardware
 - Memory
 - Swap Space
 - Disk Space
 - Input/Output Devices
 - Bitmap Display
 - Printers
 - Tape Access
 - Operating System Requirements
 - Constraints
 - Resource Constraints
 - Shared Sun Workstations
 - Release Contents
 - Documentation
 - Software
- 4 3 **Software Installation**
 - Insuring Adequate Swap Space
 - Installing Software
 - Copy Protection
 - Configuring the Software
 - Relinking
 - Enabling PUP /XNS Ethernet
 - Using/Installing the Host Access Key
- 4 4 **System Use Issues**
 - Site Initialization File
 - Starting NoteCards
 - Exiting NoteCards and Saving State
 - Keyboard Interpretation
 - Sun Type 2 Keyboard
 - Sun Type 3 Keyboard
 - Sun Type 4 Keyboard
 - Console Messages
 - File Compatibility
 - Sysout Compatibility with Xerox Workstations

- File Compatibility with Xerox Workstations
- NoteFile Compatibility with Prerelease Versions of NoteCards
- Using SunOS Files from NoteCards
 - File Naming Conventions
 - Common {DSK} and {UNIX} Naming Conventions
 - {DSK} Naming Conventions
 - Version Numbering
 - Pathnames
 - {UNIX} Naming Conventions
 - Directories
 - Directory Enumeration
 - Directory Creation
 - Open File Limit
 - Default Pathname
- File Attributes
- File System Errors

II NoteCards Tutorial

1 5 NoteCards Basics

- Starting NoteCards
- Keyboard & Mouse
 - Keyboard map
 - Mouse button usage
 - Left, Middle & Right button menus
 - Accessing & using scroll bars
- Accessing menus
 - Making & not making selections
 - Accessing submenu items
- Exiting NoteCards
 - Saving state vs. not saving state
- Escaping from Error Conditions
 - UpArrow, Proceed, (& OK?)
- Basic NoteFile Commands
 - Create
 - Open
 - Close
 - Delete
- NoteFile menus
- Card Types
 - FileBoxes
 - Text
- Miscellaneous
 - Background Menu
 - What options will be included, excluded, added (Logout T)
 - {DSK} and {UNIX} devices

4 6 Building NoteCard Structures & Modes of Use.

- Introduction to hypertext
- Introduction to what NC is and does; isn't and doesn't
- Some examples of applications NoteCards has been used for
 - Documentation
 - Bug & feature tracking/Software development records
- Using embedded fileboxes or text to group cards
- When to use multiple notefiles for a single project

III Reference Manual

4 7 The User Interface

- Keyboard
- Mouse
 - Left mouse button behavior in (includes shift, copy & move)
 - Cards
 - Other windows
 - Card titlebars
 - Link Icons
 - Card & Shrink Icons

- Notefile Banners
- Middle mouse button behavior in...
- Right mouse button behavior in ...
- Window Menu
- Background Menu
- Window Icon Menu

4 8 Links

- Introduction to Links
- Links and Link Types
 - System-Reserved Link Types
 - User-Specified Link Types
 - Link Directions and Link Ends
 - Link Categories
 - Other Link Terminology
 - Link Types vs. Card Types
- Link Access and Use
 - Link Icon Functionality
 - Link Icons Active Regions
 - Mouse Button Actions in Link Icons
- Viewing Local and Global Links
- Unfiled and Lost Cards
- Creating Links
 - Creating Links in Text Cards
 - Insert Link
 - Insert Links
 - Add Global Link
 - Add Global Links
 - Creating Links in Other Card Types
 - Text-Based Cards
 - Sketch-Based Cards
 - Graph-Based Cards
- Deleting Links
 - Deleting Links from Card Contents
 - Text-Based Cards
 - Sketch-Based Cards
 - Graph-Based Cards
 - Deleting Links from the Show Links Display
- Tailoring Links
 - Link Icon Menu
 - Bring Up Card/Box
 - Change Link Type
 - Change Card Title
 - Change Display Mode
 - System Parameters for Links
- Cross-File Links
 - Appearance
 - Missing Notefiles

4 9 Cards and Banners

- The Notefile Banner
- The Banner Title Bar
 - Notefile Ops Menus
 - Middle-Mouse-Button Title-Bar Menu
 - Full File Name
 - File Capacity
- Special Cards
 - Table of Contents FileBox
 - To Be Filed FileBox
 - Orphans FileBox
- New Cards
 - User Cards and System Cards
 - Text-, Sketch-, and Graph-based cards
 - The Card Menus

4 10 User Cards

- Text Cards

- The Text-Card Menu
- FileBox Cards
 - Suggested FileBoxes and Note Cards
 - Bibliography
 - Index
 - Read Me
 - Active Cards
- The FileBox-Card Menu
- System Parameters Affecting FileBoxes
- Sketch Cards
 - The Sketch-Card Menu
 - System Parameters Affecting Sketch Cards
- Graph Cards
 - The Graph-Card Menu
- Bit Map Editor

4/2 11 System Cards

- Browser
- Search
- Link Index
- Document

4/3 12 The MenuBox Icon

- Notefile Options
 - Open
 - Compact
 - Inspect & Repair
 - Copy
 - Rename
 - Delete
 - Create
 -
 - Checkpoint
 - Close
 - Abort
 -
 - NC FileBrowser
- Card Options
 - Close (Structure)
 - Delete (Structure)
 - Copy (Structure)
 - Move (Structure)
- Other Options
 - Edit Parameters
 - NF Indicators On
 - TEdit Killer On

4 13 System Parameters

- Accessing System Parameters
- NoteCards System Parameters
 - Extra TEdit Props
 - Include Card Object in ShowInfo
 - Del TEdit Process When Shrinking
- Font Parameters
 - Menu Font
 - Default Font
 - Link Icon Font
- Notefile Parameters
 - Show Notefile On Cards
 - New Notefile Initial Size
 - Menu Lingers After Notefile Close
- Card Parameters
 - Force Filing
 - Force Titles
 - Default Card Type
 - Close Cards Off Screen
 - Bring Up Cards At Previous Pos

- FileBox Card Parameters
 - Markers In FileBoxes
 - Alphabetized FileBox Children
- Browser Card Parameters
 - Special Browser Specs
 - Arrow Heads In Browsers
 - Link Dashing In Browser
- Sketch Card Parameters
 - Attach Sketch Menu
- Link Parameters
 - Link Icon Border Width
 - Link Icon Multi Link Mode
 - Cross File Link Mode
 - Link Icon Max Width in Pixels
 - Link Icon Show Title Default
 - Link Icon Attach Bitmap Default
 - Link Icon Show Link Type Default
 - Use Deleted Link Icon Indicators

3 14 The FileBrowser

4 15 Other Tools

- The Clocks
 - Digital clock
 - Analog clock
- The Directory Connector

4 16 Printing

- Interpress
- Postscript
- Fonts available in each

4 17 Known Problems, Error Conditions and Recovery

- Known Problems
- Break Windows
- URAIID

2 Appendix A Notefile Concepts

2 Appendix B Notefile Inspector

4 Appendix C Initialization Files

4 Appendix D Checksum Control

4 Glossary

0 Index

#####

4 TEdit Manual

#####

4 Sketch Manual

#####

2 Programmers Interface to NoteCards

#####

Status Codes:

- 0 Nothing done.
- 1 Chapter started.
- 2 Initial draft written, needs major cleanup.
- 3 Needs minor cleanup.
- 4 Ready for initial release.
- 5 Ready for final release.

ENVOS NOTECARDS™

USER'S GUIDE



Release 1.1
300300
March, 1989

Address comments to:
Envos Corporation
User Documentation
1157 San Antonio Rd.
Mountain View, CA 94043
415-966-6200

ENVOS NOTECARDS™ USER'S GUIDE

Release 1.1

Part Number 300300

March, 1989

Copyright © 1989 by Envos Corporation.

All rights reserved.

Envos is a trademark of Envos Corporation.

NoteCards™ is a trademark of Xerox Corporation, used with permission of Xerox Corporation

Xerox® is a registered trademark of Xerox Corporation.

UNIX® is a registered trademark of AT&T Bell Laboratories.

Postscript is a trademark of Adobe Systems.

The following are trademarks of Sun Microsystems, Inc.:

SunOS

Sun® and Sun Workstation® are registered trademarks

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

The information in this document is subject to change without notice and should not be construed as a commitment by Envos Corporation. While every effort has been made to ensure the accuracy of this document, Envos Corporation assumes no responsibility for any errors that may appear.

Text was written and produced with Envos text formatting tools; Xerox printers were used to produce text masters. The typeface is Modern.

NoteCards

NoteCards is a computer environment designed to help people work with ideas. Its users are authors, researchers, designers, and other intellectual laborers engaged in analyzing information, constructing models, formulating arguments, designing artifacts, and generally processing ideas. The system provides these users with a variety of tools for collecting, representing, managing, interrelating, and communicating ideas. NoteCards is based on the notion that creative intellectual work is a hand-craft, a uniquely human skill that cannot be easily automated.

Notecards provides the user with a "semantic network" of electronic notecards interconnected by typed links. This network serves as a medium in which the user can represent collections of related ideas. It also functions as a structure for organizing, storing, and retrieving information. The system provides the user with tools for displaying, modifying, manipulating, and navigating through this network.

From: *NoteCards in a Nutshell*, by Frank G. Halasz, Thomas P. Moran, Randall H. Trigg of the Intelligent Systems Laboratory in the Xerox Palo Alto Research Center, Preceedings of the ACM CHI+GI '87 Conference, Toronto, Canada April 1987

System Overview

	Functionally, the NoteCards system for the Sun Workstation consists of the following parts:
<i>emulator</i>	A SunOS executable program which executes the NoteCards system contained in the sysout and provides access to the Sun host's hardware.
<i>sysout</i>	A virtual memory image (the <i>sysout</i>) containing both the NoteCards program and its data structures. The sysout provided can be used both on the Sun Workstations and on the Xerox 1100 series workstations.
<i>notefiles</i>	A file containing your cards on a particular topic and all their links. This is your data and its organizational structure.
<i>fonts</i>	Data describing the "looks" of printed characters used by NoteCards' graphics, windowing, and hardcopying subsystems. Font directories are in four groups, display fonts, PostScript printer fonts, Interpress printer fonts, and Press printer fonts.

Useful SunOS and UNIX Conventions

SunOS is Sun Microsystems' version of the UNIX operating system. For users unfamiliar with the Sun Workstation, the following SunOS (and UNIX) conventions, are used in the manual.

For complete information on UNIX and SunOS, refer to your Sun documentation set.

case, filenames

Type-in to UNIX is case sensitive. Typically, input is in lower case. When UNIX searches for a name, it is case sensitive; it distinguishes between lower and upper case characters. By convention, most names are lower case characters.

shell

Command interpreter; the commands shown are in the C-Shell, unless otherwise noted.

NoteCards Device Conventions

NoteCards allows users to interact with SunOS file systems (including file systems mounted from other machines) by using host device names. The device names are

{DSK}

A host name which gives you access to the SunOS file system using Xerox workstation local disk conventions.

{UNIX}

A host name which gives you access to the file system using normal SunOS conventions.

The {DSK} device name provides an interface to the Sun Workstation for users who want to maintain compatibility with existing development tools and applications originally developed on a Xerox workstation. The {UNIX} device name provides a way for new applications to interact naturally with UNIX. Chapter 4, System Use Issues, explains, in greater detail, some important exceptions and restrictions to the {DSK} and {UNIX} device name.

Stylistic Conventions

Text marked by a revision bar in the right margin contains information that was added or modified since the last release.

Prompts

All examples which include SunOS dialogues use the following conventions for the SunOS prompt:

A number sign (#), as part of the system prompt, indicates that the user is logged on as root or is running su; e.g.,

prompt #

A percentage sign (%), as part of the system prompt, indicates that a user other than root is logged on; e.g.,

prompt %

Font Usage

Bold text in TITAN font indicates text you should type in exactly as printed.

Regular text in TITAN font indicates what the system prints on your workstation screen. UNIX files and programs are shown in TITAN FONT.

Italic text in Titan font indicates variables or parameters that you should replace with the appropriate word or string.

Bold text in Modern font is used to indicate menu commands and NoteCards parameters.

Italic text in Modern font is used to indicate emphasis.

Quote marks are used to indicate window titles, parameter values, and when referring to the names of section headings within chapters; e.g., The "Stylistic Conventions" section in Chapter 1, Introduction.

Keyboard Conventions

Keys that you press are in uppercase (e.g., COPY, for the Copy key). A carriage return is displayed as <RETURN>. Instructions that ask you to press two or three keys simultaneously are indicated as follows:

"Press CONTROL-E"

Note that on some Xerox machines, the CONTROL key is labeled PROPS or EDIT, but has the same function as the CONTROL key.

[This page intentionally left blank]

Welcome to NoteCards, an idea manipulation tool for the modern intellectual. NoteCards provides you with a new way to organize and manage your ideas on your computer desk top.

This guide explains in detail how to install NoteCards on your Sun 3 or Sun 4 workstation, provides a basic tutorial for the NoteCards environment, and contains the reference manual for NoteCards.

Audience

The *Envos NoteCards Users's Guide* was written for users of NoteCards on Sun workstations. The *Guide* assumes that you are already familiar with UNIX and SunOS concepts.

Section I of this manual, Introduction and Installation, should be read by the System Administrator or person installing the NoteCards system. The NoteCards user should also read this section, in particular, Chapters 1 and 4, Introduction and System Use Issues. The user who wishes to remain safely ignorant of system-level aspects of UNIX and NoteCards can skip over Chapters 2 and 3, System Requirements and Software Installation. (The system installer should also read Appendices C and D.)

Section II, NoteCards Tutorial, provides an introduction to NoteCards and the system which underlies it. Chapter 5, NoteCards Basics, is the place for all new users of NoteCards to start once they have their system up and running.

Section III, Reference Manual, is a compilation of all the necessary background information on NoteCards new and old users will need to learn to fully exploit the NoteCards system.

The final section contains the Appendices, Glossary, and Index for the *Envos NoteCards User's Guide*.

Included with the NoteCards Guide are the manuals for TEdit and Sketch, *A User's Guide to TEdit* and *A User's Guide to Sketch*.

What's in the Manual

Here is what you will find in this manual, chapter by chapter, accompanied by a brief description of when you will want to read each chapter.

Chapter 1, Introduction, gives a brief introduction to and some background on NoteCards as well as explaining the stylistic conventions used in the *Guide*. Read this chapter before you install NoteCards.

Chapter 2, System Requirements, goes over the hardware requirements of the system. Read this chapter before you install NoteCards on a particular machine to verify that it will run acceptably on that machine.

Chapter 3, Software Installation, gives you step by step directions for installing NoteCards. Read this chapter before and during system installation.

Chapter 4, System Use Issues, discusses some of the basic ins and outs of using the system, Initialization, Starting, and Exiting NoteCards, as well as file access. Read this chapter after you have installed NoteCards. In particular, read the sections "Starting NoteCards" and "Exiting NoteCards and Saving State" when you are modifying your .cshrc and .login UNIX files for ideas on how to simplify the NoteCards booting process. In order to allow it to run on two different file systems NoteCards differs slightly from the standard SunOS and UNIX file systems. Read the section "Using SunOS Files from NoteCards" when you are learning how to access files from NoteCards.

Chapter 5, NoteCards Basics, takes you by the hand and leads you step by step through the basics of using the NoteCards system. This is the place for new users to start once you have NoteCards up and running. Once you get going, this is also a good chapter to review as it brings together ideas scattered through the users's guides whose importance many not be obvious from their context in the user's guides.

Chapter 6, Building NoteCards Structures, tries to give you some insights on how to use the NoteCards systems to its best advantage. NoteCards has been around as a PARC research prototype for several years. In this chapter we try to present some of what they have learned about using this system. Read this chapter after you have become somewhat familiar with NoteCards.

Chapter 7, The User Interface, covers some of the more basic aspects of using the system, the mouse, the keyboard, menus, etc. Read this chapter to understand more of the lower level capabilities of the system.

Chapter 8, Links, describes the link icon in all of its manifestations. Links and their physical representation, link icons, live at the core of NoteCards. Read and understand this chapter as soon as possible to make the most effective use of the system.

Chapter 9, Cards and Banners, discusses the user interface to individual notefiles and those aspects that all cards have in common. This is also an important chapter. The section "The Card Menu" is particularly important as it explains the functionality all cards have in common.

Chapter 10, User Cards, discusses cards where you are responsible for creating the contents. Read the sections "Text Cards" and "FileBox Cards" right away. The "Text Cards" section points you to the TEdit manual *A User's Guide to TEdit*. As TEdit forms the core of the system. We suggest that you make an effort to gradually learn more about TEdit all the time. Read the remaining sections as you explore the other card types. Note that the Bit Map Editor is discussed at the end of this chapter even though there is no bit-map card type

Chapter 11, System Cards, covers those cards where the system creates the contents for you. The section "Browser Cards" is the

most important one to read for a beginner. The others can be read as you need them.

Chapter 12, The MenuBox Icon, discusses the functionality on the three menus available from the MenuBox. The important commands for a new users are on the "Notefile Ops" menu. When just getting used to NoteCards, focus on the commands: **Open**, **Checkpoint**, **Close**, and **Abort**.

Chapter 13, System Parameters, explains how to change the global defaults for the system. New users can safely ignore this chapter. Most users will find that the default settings will suit all their needs.

Chapter 14, The FileBrowser, explains how to use the FileBrowser.

Chapter 15, Other Tools, explains how to use the Directory Connector, and the clocks.

Chapter 16, Printing, covers setting up print capabilities and printing to PostScript and Interpress printers. Read this chapter when you have the machine on a network and are ready to set up your print capability.

Chapter 17, Known Problems, Error Conditions and Recovery, covers those problems we know about as of this writing and how to cope with and avoid them. A quick reading of the sections "Known Problems" and "Break Windows" are all that most users will ever need.

Appendix A, Notefile Concepts, explains the inner workings and structure of notefiles. Read this if you want to understand how information is stored in the notefile and why you have to perform operations like compacting occasionally. This is essential background material if you are about to inspect or fix a damaged notefile.

Appendix B, Notefile Inspector, explains how to fix notefiles damaged by power failed, broken net connections and other unforeseeable calamities. Read this before you attempt to fix a damaged notefile.

Appendix C, Initialization Files, collects a lot of information on how to write a file which will automatically set site specific values for parameters like printer names and font locations. Read this appendix when you are first setting up your system.

Appendix D, Checksum Control, covers what to do if you believe the files loaded from the distribution tape are damaged.

Acknowledgements

NoteCards has a long history. Its gestation began in October 1982 when the U.S. Government funded XSIS (Xerox Special Information Systems) and Xerox PARC (Palo Alto Research Center) to prototype a Problem-Structuring-Aids system to allow users to build "semantic networks of textual information." The first successful prototype was completed by Frank Halasz, at PARC, in

November 1983. The first real user of the NoteCards system was Ken Allen, a history graduate student at Stanford University, in the spring of 1984. The birth came in June 1984 when NoteCards was released to the government, in fulfillment of the Problem-Structuring-Aids contract. Since then, XSIS and PARC have released three unsupported versions of NoteCards on Xerox D-machines. In November 1988, the Envos NoteCards product team started work on a supported, productized version of NoteCards. Envos NoteCards Release 1.1 for the Sun Workstation was completed in April 1989.

NoteCards is based on the work of many people.

From Xerox, Frank Halasz, Tomas Moran, Randall Trigg, Richard Burton, Ronald Kaplan, Peggy Irish, Catherine Marshall, and many others.

From Envos, Robert Krivacic, Keith Mountford, Craig Sweat, Karin Sye, Daniel Sagalowicz, Larry Harada, John Sybalsky, and others.

[This page intentionally left blank]

Preface

iii

I Introduction and Installation

1. Introduction 1-1

<u>NoteCards 1-1</u>	
<u>System Overview</u>	1-1
<u>Useful SunOS and UNIX Conventions</u>	1-2
<u>NoteCards Device Conventions</u>	1-2
<u>Stylistic Conventions</u>	1-2
<u>Prompts</u>	1-2
<u>Font Usage</u>	1-3
<u>Keyboard Conventions</u>	1-3

2. System Requirements 2-1

<u>Prerequisites</u>	2-1
<u>Processor Hardware</u>	2-1
<u>Memory</u>	2-1
<u>Swap Space</u>	2-1
<u>Disk Space</u>	2-1
<u>Input/Output Devices</u>	2-1
<u>Bitmap Display</u>	2-2
<u>Printers</u>	2-2
<u>Tape Access</u>	2-2
<u>Operating System Requirements</u>	2-2
<u>Constraints</u>	2-2
<u>Resource Constraints</u>	2-2
<u>Shared Sun Workstations</u>	2-2
<u>Release Contents</u>	2-3
<u>Documentation</u>	2-3
<u>Software</u>	2-3

3. Software Installation 3-1

<u>Insuring Adequate Swap Space</u>	3-1
<u>Installing Software</u>	3-1
<u>Copy Protection</u>	3-3
<u>Configuring the Software</u>	3-3
<u>Relinking</u>	3-4

Enabling PUP/XNS Ethernet	3-4
Using/Installing the Host Access Key	3-4
4. System Use Issues	4-1
Site Initialization File	4-1
Starting NoteCards	4-1
Exiting NoteCards and Saving State	4-2
Keyboard Interpretation	4-4
Sun Type 2 Keyboard	4-4
Sun Type 3 Keyboard	4-5
Sun Type 4 Keyboard	4-6
Consol Messages	4-6
File Compatibility	4-7
Sysout Compatibility between Sun and Xerox Workstations	4-7
File Compatibility between Sun and Xerox Workstations	4-7
Notefile Compatibility with Prerelease Versions of NoteCards	4-8
Using SunOS Files from NoteCards	4-8
File Naming Conventions	4-9
Common {DSK} and {UNIX} Naming Conventions	4-9
{DSK} Naming Conventions	4-10
Version Numbering	4-10
Pathnames	4-12
{UNIX} Naming Conventions	4-12
Directories	4-13
Directory Enumeration	4-13
Directory Creation	4-13
Open File Limit	4-14
Default Pathname	4-14
File Attributes	4-14
File System Errors	4-15

II NoteCards Tutorial

5. NoteCards Basics	5-1
A Few Pointers on Mouse Etiquette	5-1
Mouse Button Use	5-2
Left Mouse Button	5-2
Middle Mouse Button	5-2
Right Mouse Button	5-2
Regions Sensitive to Mouse Activity	5-2

<u>The Background</u>	5-2
<u>Windows and the Window Title Bar</u>	5-3
<u>Icons</u>	5-5
<u>Scroll Bars</u>	5-5
<u>Dealing with Simple Error Conditions</u>	5-6
<u>Starting NoteCards</u>	5-7
<u>Closing Notefiles and Exiting NoteCards</u>	5-8
<u>Closing Notefiles</u>	5-8
<u>Close</u>	5-9
<u>Abort</u>	5-9

6. Building NoteCards Structures and Modes of Use 6-1

<u>Introduction</u>	6-1
<u>A Standard View of Building NoteCards Structures</u>	6-1
<u>A Non-Standard View of Building NoteCards Structures</u>	6-1
<u>FileBoxes and Hierarchies</u>	6-1
<u>The Flat System</u>	6-2
<u>The Almost Flat System</u>	6-2
<u>Link Types</u>	6-2
<u>Multiple Users</u>	6-2
<u>Cards and Information Chunk Size</u>	6-2
<u>Types of Structures People Have Built with Notecards</u>	6-3
<u>Specalizing NoteCard</u>	6-3

III Reference Manual

7. The User Interface 7-1

8. Links 8-1

<u>Links and Link Types</u>	8-1
<u>System-Reserved Link Types</u>	8-1
<u>User-Specified Link Types</u>	8-2
<u>Link Directions and Link Ends</u>	8-3
<u>Link Categories</u>	8-3
<u>Other Link Terminology</u>	8-3
<u>Link Types vs. Card Types</u>	8-3
<u>Link Access and Use</u>	8-4
<u>Link Icon Functionality</u>	8-4
<u>Active Regions of Link Icons</u>	8-4
<u>Mouse-Button Actions in Link Icons</u>	8-5
<u>Viewing Local and Global Links</u>	8-7

TABLE OF CONTENTS

Unfiled and Lost Cards	8-8
Creating Links	8-9
Creating Links in Text Cards	8-9
Insert Link	8-9
Insert Links	8-12
Add Global Link	8-13
Add Global Links	8-13
Creating Links in Other Card Types	8-13
Text-Based Cards	8-13
Sketch-Based Cards	8-13
Graph-Based Cards	8-14
Deleting Links	8-14
Deleting Links from Card Contents	8-15
Text-Based Cards	8-15
Sketch-Based Cards	8-15
Graph-Based Cards	8-15
Deleting Links from the Show Links Display	8-16
Tailoring Links	8-16
Link Ops Menu	8-16
Bring Up Card/Box	8-16
Change Link Type	8-16
Change Card Title	8-16
Change Display Mode	8-16
System Parameters for Links	8-18
Cross-File Links	8-18
Appearance	8-18
Missing Notefiles	8-19

9. Cards and Banners	9-1
The Notefile Banner	9-1
The Banner Title Bar	9-1
Notefile Ops Menus	9-1
Middle-Mouse-Button Title-Bar Menu	9-3
Full File Name	9-3
File Capacity	9-3
Special Cards	9-4
Table of Contents FileBox	9-5
To Be Filed FileBox	9-5
Orphans FileBox	9-6
New Cards	9-6
User Cards and System Cards	9-7

Text-, Sketch-, and Graph-Based Cards	9-8	
The Card Menu	9-8	
The Standard Card Menu	9-8	
Edit Property List	9-8	
Show Links	9-9	
Show Info	9-10	
Designate FileBoxes	9-11	
Assign Title	9-11	
Title/FileBoxes	9-12	
Insert Link	9-12	
Close and Save	9-13	
The FileBox Card Menu	9-13	
Add Global Link	9-13	
Put Cards Here	9-14	
10. User Cards	10-1	
Text Cards	10-1	
The Text-Card Menu	10-1	
FileBox Cards	10-2	
Suggested FileBoxes and Note Cards	10-3	
Bibliography	10-3	
Index	10-3	
Read Me	10-3	
Active Cards	10-3	
The FileBox-Card Menu	10-3	
System Parameters Affecting FileBoxes	10-3	
Sketch Cards	10-3	
The Sketch-Card Menu	10-3	
The Map Option	10-4	
System Parameters Affecting Sketch Cards	10-4	
Graph Cards	10-4	
The Graph-Card Menu	10-5	
The Grapher Menu	10-5	
The Graph-Card and Grapher Menu Options	10-6	
Move Node	Move Node	10-6
Remove Node	Delete Node	10-7
Connect Nodes	Add Link	10-7
Disconnect Nodes	Delete Link	10-7
Add Label	Add Node	10-8
Change Label	Change Label	10-8
Smaller Label	Label Smaller	10-8

TABLE OF CONTENTS

Larger Label	Label Larger	10-8
Toggle Shade	<-> Shade	10-8
Toggle Border	<-> Border	10-8
Directed/Undirected	<-> Directed	10-8
Sides/Centers	<-> Sides	10-9
Fix Menu		10-10
	Stop	10-10
Bit Map Editor		10-10
Inserting Bit Maps into Cards		10-10
Text-Based Cards		10-10
Sketch-Based Cards		10-11
Bit Map Operations		10-11
Change Scale		10-11
Hand Edit		10-12
Trim		10-12
Reflect Left-to-right		10-12
Reflect Top-to-Bottom		10-12
Reflect Diagonally		10-12
Rotate Left		10-12
Rotate Right		10-12
Expand on Right		10-12
Expand on Left		10-13
Expand on Bottom		10-13
Expand on Top		10-13
Switch Black & White		10-13
Add Border		10-13
The Bit Map Editor		10-14
Paint		10-16
ShowAsTile		10-16
Grid On/Off		10-16
GridSize		10-16
Reset		10-17
Clear		10-17
Cursor		10-17
OK		10-17
Abort		10-18

11. System Cards	11-1
Search Cards	11-1
Link Index Cards	11-2
Document Cards	11-3

HeadingsFromFileBoxes	11-3
TitlesFromNoteCards	11-4
BuildBackLinks	11-4
CopyEmbeddedLinks	11-4
ExpandEmbeddedLinks	11-4
Browser Cards	11-5
Changes to the NoteCards Browser	11-6
Multiple Roots	11-7
Dashed Links	11-7
Arrowheads	11-7
Browser Specs	11-7
New Middle-Button Title-Bar Menu Options	11-9
Editing the Browser Manually and Structure Editing	11-9
Overview Windows on Browser Cards	11-12
Creating an Overview Window	11-12
Reshaping the Overview Window	11-13
Scrolling and the Wire Frame	11-13
Recomputing the Overview Contents	11-13
The Browser Overview Stylesheet	11-13
Tidbits	11-14

12. The MenuBox Icon 12-1

Notefile Options	12-1
Open	12-2
Compact	12-3
Inspect & Repair	12-4
Copy	12-4
Rename	12-4
Delete	12-4
Create	12-4
Checkpoint	12-5
Close	12-6
Abort	12-6
NC FileBrowser	12-6
Card Options	12-6
Close	12-7
Delete	12-8
Copy	12-8
Move	12-8
Other Options	12-9
Edit Parameters	12-9

NF Indicators On	12-9
TEdit Killer On	12-9
13. System Parameters	13-1
Accessing System Parameters	13-1
NoteCards System Parameters	13-3
Extra TEdit Props	13-3
Include Card Object In ShowInfo	13-3
Del TEdit Process When Shrinking	13-3
Font Parameters	13-4
Menu Font	13-4
Default Font	13-4
Link Icon Font	13-4
Notefile Parameters	13-5
Show Notefile On Cards	13-5
New Notefile Initial Size	13-6
Menu Lingers After Notefile Close	13-6
Card Parameters	13-6
Force Filing	13-6
Force Titles	13-6
Default Card Type	13-6
Close Cards Off Screen	13-7
Bring Up cards At Previous Pos	13-7
FileBox Card Parameters	13-7
Markers In FileBoxes	13-7
Alphabetize FileBox Children	13-7
Browser Card Parameters	13-7
Special Browser Specs	13-7
Arrow Heads In Browsers	13-8
Link Dashing In Browsers	13-8
Sketch Card Parameters	13-8
Attach Sketch Menu	13-8
Link Parameters	13-8
Link Icon Border Width	13-8
Link Icon Multi Line Mode	13-9
Cross File Link Mode	13-9
Link Icon Max Width in Pixels	13-9
Link Icon Show Title Default	13-9
Link Icon Attach Bitmap Default	13-10
Link Icon Show Link Type Default	13-10
Use Deleted Link Icon Indicators	13-10

14. The FileBrowser	14-1
User Interface	14-1
Starting FileBrowser	14-1
Specifying What Files to Browse	14-1
Examples	14-2
Using the FileBrowser Window	14-3
Selecting Files	14-5
Commands the Require Input	14-5
Aborting Commands	14-6
Quitting the FileBrowser	14-6
Copy-Selecting Files	14-7
Getting Hardcopy Directory Listings	14-7
FileBrowser Commands	14-8
Delete, Undelete	14-8
Copy	14-9
Rename	14-11
Hardcopy	14-12
See	14-13
FileBrowse	14-15
Edit	14-15
Load	14-16
Compile	14-16
Expunge	14-16
Recompute	14-16
New Info	14-17
Set Depth	14-19
Sort	14-19
Troubleshooting Problems with FileBrowser	14-20
15. Other Tools	15-1
Digital Clock	15-1
Introduction	15-1
Starting the Digital Clock	15-1
Stopping the Digital Clock	15-2
Changing the Digital Clock	15-2
Set Font	15-3
Set Time	15-4
Set Alarm	15-4
Quiet Alarm/Loud Alarm	15-5
Delete Alarm Setting	15-5

TABLE OF CONTENTS

Shape to Fit	15-5
12-Hour/24-Hour Clock	15-5
Set Local Time Zone	15-5
Add New Regional Time Zone	15-5
Delete This Window	15-6
Set Font for Aux Clocks	15-6
Set Aux Clock Font In Just This Window	15-6
Set Time-Zone Heading	15-6
Set Regional Time Zone	15-7
Analog Clock	15-7
Introduction	15-7
Starting the Analog Clock	15-7
Stopping the Analog Clock	15-8
Changing the Analog Clock	15-8
Numbers	15-9
Points	15-9
No Numbers	15-9
Rings	15-9
No Rings	15-9
Hands	15-9
No Hands	15-9
Times	15-9
No Times	15-9
Show Style	15-9
Set to Default	15-9
Change Default	15-10
Set Time	15-10
Known Problems	15-10
Directory Connector	15-10
Starting the Directory Connector	15-10
Stopping the Directory Connector	15-10
Changing the Directory Connector Fonts	15-10
Using the Directory Connector	15-11
Left Mouse Button	15-11
Middle Mouse Button	15-12
16. Printing	16-1
17. Error Recovery and Known Problems	17-1
System Status, Aborting Operations, and Spawning a New Mouse	17-1
System Status	17-1

Aborting Operations	17-1
Spawning a New Mouse	17-3
Break Windows	17-3
Errors While Running NoteCards	17-4
I/O Errors	17-4
Virtual Memory Errors	17-5
URAIID	17-5
Entering URAID	17-5
URAIID Commands	17-5
Other Fatal Error Conditions	17-6
System Error Conditions	17-6
Known Problems	17-7
Printing Browsers	17-7
Notefile Indicator Window	17-8
PostScript Fonts	17-8

Appendices, Glossary, and Index

Appendix A. Notefile Concepts	A-1
Appendix B. Notefile Inspector	B-1
Appendix C. Installation Files	C-1
Appendix D. Checksum Control	D-1
Glossary	GLOSSARY-1
Index	INDEX-1

[This page intentionally left blank]

BUILDING

6. NOTECARDS STRUCTURES

NoteCards has been around for some time and people have developed different ways of using it. In this chapter we have recorded some of the different approaches people have taken towards using NoteCards in the hopes that these notes will help you discover the best way of using NoteCards for you.

If you have ideas about ways of using NoteCards which you would like to share with others please send them to us.

Introduction

By its very design, NoteCards encourages you to use it in specific ways. However, some users have found that they make the best use of NoteCards by not following the intended path but by branching off and forcing NoteCards to conform to their own idea of what a hypertext system should look like.

A Standard View of Building NoteCards Structures

NoteCards was intended to support two parallel modes of organization, a hierarchial one, implemented with FileBoxes, and a relational one, implemented with links.

One possible scenario is that you would collect a body of material and break it up into logical bits which would then be placed into Text, Sketch, Graph, and other card types. Ideally, the material would be scanned in, or collected from electronic sources. You would then build a hierarchial structure with FileBoxes inserting the data cards in their appropriate place in the FileBox hierarchy. Once you had established the structure you would then read through the document building links between related ideas, and constructing other non-hierarchial structures keeping these several lines of thought apart by using different link types. The different lines of thought in the material could then be collected by using document cards to follow the different link types.

This is a stereotypical view of how you might generate a structure in NoteCards however, it is not the only view.

A Non-Standard View of Building NoteCards Structures

FileBoxes and Hierarchies

One frequent comment from long time users is that people have a tendency to use FileBoxes too soon. They try to organize their data into a structure before the best structure is apparent and wind up reorganizing their ideas, wasting much time and energy. These users suggest that you build your structures only after you have worked with your data for awhile.

Other users simply do not use FileBoxes at all or use them to build only very shallow structures. Note that you could use links directly to build the equivalent hierarchial structure as with FileBoxes.

The Flat System

In the flat system, all cards are filed in the "Table of Contents", the "To Be Filed" or some user-specified FileBox card. These users may specify one or several start points for the reader, but aside from that the system is flat.

The Almost Flat System

In this organization the cards are grouped into large bunches which can be based on source, topic or some other broad category. One FileBox is used for each source, topic, or whatever the divisions are. One user likens his use of FileBoxes to library shelves where each FileBox is a different shelf. He then uses Sketch cards as an organization and integration tool, to help him navigate through his system.

Link Types

The system is built with the idea that you will classify each link type so that later you can build documents or browsers based on link types. However, some users make all their link types "Unspecified."

One of the dangers with links is that new users tend to link everything to everything else and end up with spaghetti.

Multiple Users

NoteCards was not designed with collaboration in mind. One way users have gotten around the problem of tracking what each user has done is to assign each collaborator a different font. Each contributor's work can then be recognized by font face.

Cards and Information Chunk Size

Because cards come up on the screen a particular size, able to display a small amount of information, the perception grows among new users that the information chunks have to be small. It is important to recognize that cards can hold any amount of information, and that they can be scrolled or resized to display all the information they hold.

There are trade offs, however. If you work with large cards, the size of your notefile grows much more rapidly. This is not serious. It only means that you need to compact your notefile more frequently than people who work with smaller information chunks do.

A second trade off is that links always point to the beginning of a card never into the body of a card. If your cards are very large, it may not be immediately clear what the relationship is between the source card and the destination card.

Types of Structures People Have Built with NoteCards

One user working on a design project used a notefile structured with design goals as roots moving down to the component level at leaf nodes. This structure allowed him to see the design history, subgoals, and goals which were retracted.

Other users use NoteCards to maintain indices of where to find sources of information. This is fairly common. Users frequently implement on-line rolodexes, customer-support or customer-tracking notefiles, and other types of text databases, such as project tracking.

The most common use is in writing large reports. NoteCards is used to collect, organize, and cross-reference information until the writer has a firm grasp of the material and how various pieces of it interact.

Specializing NoteCards

It is possible to specialize NoteCards to specific tasks.

One user has created an interface to the Lexis legal database. To do this he created a new text-card type which receives retrieved data from Lexis. He is also working on an interface to videodisk and other sources.

At Xerox PARC there is a group working on an Instructional Design Environment for designing, developing, and presenting instruction. These people are extensively modifying the NoteCards environment

If you have a need for a particular extension to NoteCards there is a programmers interface which you or your company's developers can use to extend the NoteCards environment, or Envos Consulting Services group can provide you with specialized extensions to NoteCards.

[This page intentionally left blank]

7. THE USER INTERFACE

This chapter covers the basics of using the window system NoteCards is built on top of. Read this chapter if you are a new user of the system or you need to understand how to manipulate windows, icons, and use the functions on the background menu. For a lower-level introduction to mouse-button and keyboard use, read Chapter 5, NoteCards Basics.

This chapter explains:

- Where to find system menus.

- What these menus do.

The Window Menu

The NoteCards window system allows you to interactively manipulate the windows on the screen, moving them around, changing their shape, etc. by selecting various operations from the window menu.

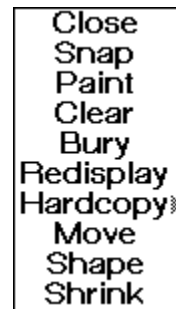


Figure 7-1. The window menu.

When you press and hold down the right mouse button in a window's title bar, as shown in Figure 7-2, that window will come to the top and a menu of window operations will appear. In some cases, you can hold down the right mouse button anywhere in the window to access the window menu.

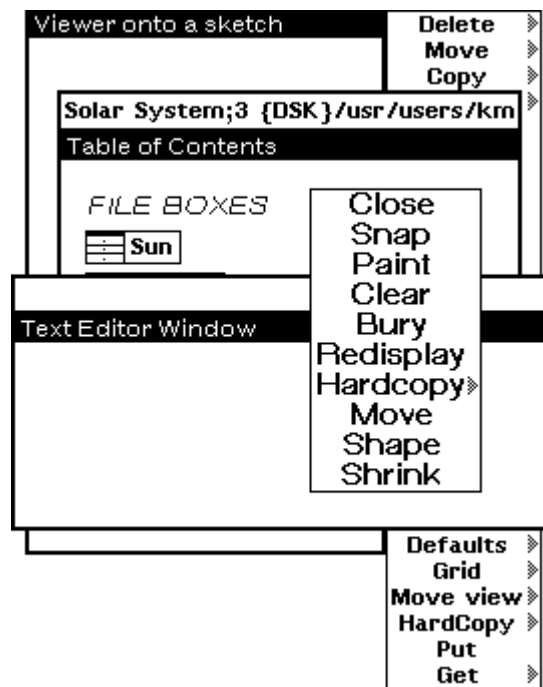


Figure 7-2. The window menu being accessed from a TEdit-window title bar. Note how the TEdit window has been brought to the top of all the windows.

Close

Closes a window, i.e., removes it from the screen. If you have modified the contents of the window, the system may ask you to confirm, by pressing the left mouse button, that you want to close the window and lose your changes. Pressing the right or middle mouse button cancels the close operation.

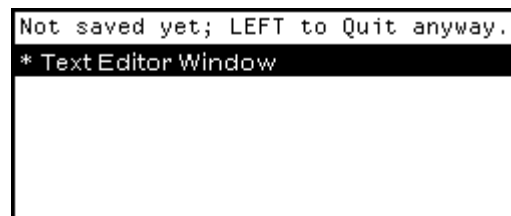


Figure 7-3. A TEdit window requesting you to confirm the **Close** command.

Snap

Prompts for a region on the screen and makes a new window whose image is a snapshot of the image currently in that region. Useful for saving some particularly choice image before the window image changes.

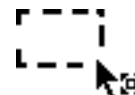


Figure 7-4. The mouse cursor prompting you to sweep out a new size and shape for a window.

Most of the images presented as figures in this manual are screen snaps.

When making a snap, it is possible to change corners and change the snap shape from a different side of the ghost frame. In the middle of a snap process, while you still have the left mouse button down, simultaneously press and hold down the right mouse button.

When you do this, the forceps cursor () will appear. Move this cursor to the corner of the ghost frame you want to move and still holding the left mouse button down, release the right mouse button. You will now be able to adjust the snap from the corner you just selected.

Paint

Switches to a mode in which the cursor can be used like a paint brush to draw in a window. This can be useful for making notes on a window or touching up snaps. While the left key is down, pixels are blackened. While the middle key is down, they are erased. The right button pops up a command menu that allows you to change the brush size, shape, shade, mode, and to quit the paint utility.



Figure 7-5. Paint command menu.

Set Mode

Brings up the menu shown in Figure 7-6.



Figure 7-6. The Mode menu for Paint.

Changing the paint mode changes the way the bits in the brush combine with the existing bits in the snap.

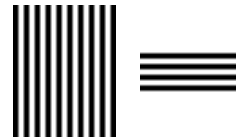


Figure 7-7. The vertical bars show a sample bit map pattern from a snap. The horizontal bars are the pattern in the example brush.

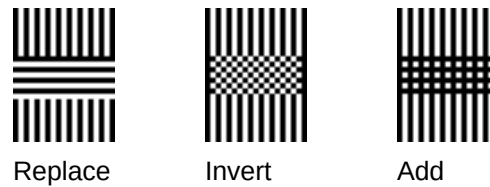


Figure 7-8. The snap and brush patterns shown in Figure 7-7 combine as shown here.

Set Shade

Pops up a menu which allows you to choose one of a preexisting set of shades or specify your own 4x4 shade. New shades are added to the "Choose shade" menu.

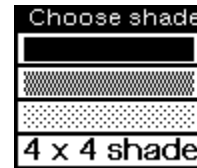


Figure 7-9. The "Choose shade" menu.

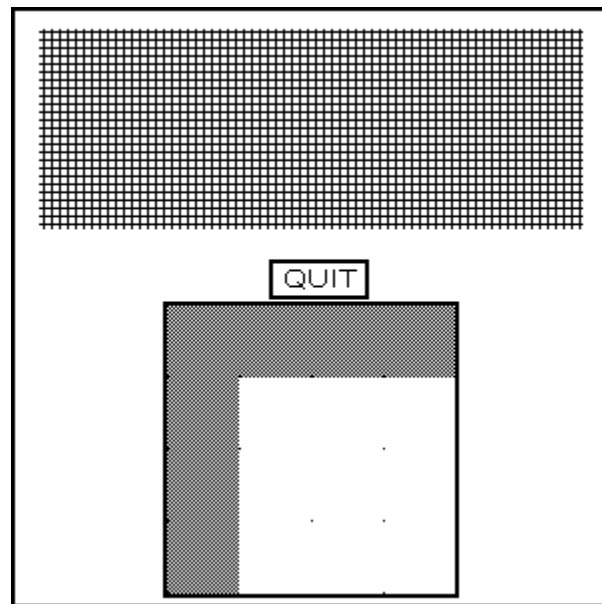


Figure 7-10. The 4x4 shade tool.

Set Shape

Pops up the menu below which allows you to set the shape of the mouse cursor.



Figure 7-11. The shape menu.

Set Size

Brings up the following menu which sets the cursor brush size.

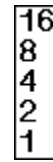


Figure 7-12. The size menu.

Quit

Quits the paint program.

Clear

Clears the window. Sets all pixels in the window to white.

Bury

Places the window under all other windows which it covers or overlaps, thereby exposing any windows that it was hiding.

Redisplay

Clears the window and rewrites the window contents to the window.

Hardcopy

Sends the contents of the window to the printer. If the window is associated with a text, sketch, or graph editor the editor's contents are printed. In the case of TEdit and Sketch, this means that the associated TEdit file, if there is one, is printed. If the window is not associated with an editor a bit-map image of the window is sent to the printer.

To save the image in a PostScript, Interpress, or Press formatted file, or to send it to a non-default printer, use the submenu of the **Hardcopy** command. When the mouse is moved off to the right of the **Hardcopy** menu item, a second pop-up menu appears giving the choices **To a file** or **To a printer**.

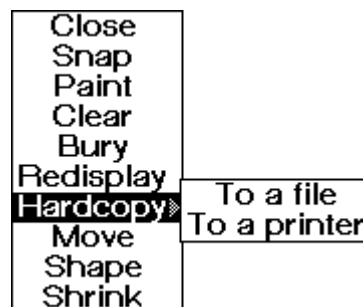


Figure 7-13. The Hardcopy submenu.

If **To a file** is selected, you are prompted to supply a file name, and the format of the file (PostScript or Interpress) from the "File type?" menu and the contents of the window is stored in that file, formatted for the printer type you specified.

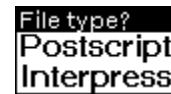


Figure 7-14. The "File type?" menu.

The system provides the suffixes for these files. PostScript files have a "ps" suffix. Interpress files have an "ip" suffix. For example, given a file name of "Letter," the system stores a file with the name "Letter.ps" for a PostScript formatted file and "Letter.ip" for an Interpress formatted file. If you change the suffix for a formatted file from "ps" or "ip" to some other name, the system will prompt you to let it know the file type when you try to print the file. For a formatted file, select the **BINARY** response. For an unformatted file, select the **TEXT** response. You print a formatted file the same way you print an unformatted file, by using the **Hardcopy** option on the FileBrowser menu.



Figure 7-15. The "File Type?" menu requesting the file type of a file formatted for a particular printer type.

If you select **To a printer**, you are prompted to select a printer from the list of known printers, or to type the name of another printer.

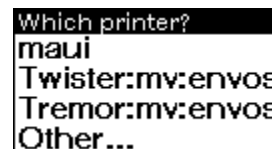


Figure 7-16. The "Which printer?" menu asking which printer you want to send to.

The topmost printer in the "Which printer?" menu is the default printer. If the printer selected is not the topmost printer on the list, indicating that it is not the default printer, you will be asked whether to make the printer you selected the new default printer.

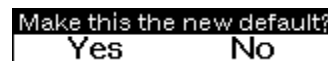


Figure 7-17. The "Make this the new default?" menu asking you whether you want to make the printer you selected the default printer.

Note, unless you are using only fully specified Interpress printer names or you have set the DEFAULTPRINTERTYPE variable from your initialization file, it is unadvisable to use the **Other...** option on the "Which printer?" menu shown in Figure 7-16, as using this option does not allow you to specify the printer type.

Move

Moves the window to a new screen location. When you select **Move** you are presented with a ghost frame which is the size and shape of the original of the window you are moving. You move the mouse to position the ghost frame where you want the window to appear and then you plant the window by clicking the left mouse button.

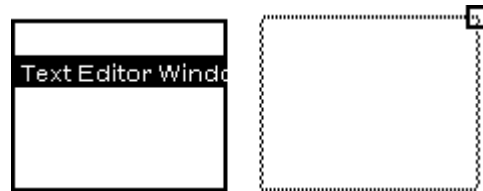


Figure 7-18. A TEdit window and its ghost frame.

Shape

Allows the user to specify a new size and shape for an existing window. You can use either the left or middle mouse buttons to shape a window.

Using the left mouse button allows you to change the size, shape, and location of a window. Once you have selected **Shape** from the window menu, position the cursor where you want the upper left corner of the window to be located, press and hold down the left mouse button, sweep out a new region for the window, and release the mouse button when the window has the size and shape you want.

Using the middle mouse button allows you to change the size and shape of a window by moving just one corner of the window. Select **Shape** from the window menu and position the mouse cursor at the corner you want to move. Press and hold down the middle mouse button and move the mouse cursor to the location you want that corner of the window to be at. A ghost frame will appear to indicate the new size and shape of the window.

When reshaping a window with either the left or middle mouse button it is possible to change corners and change the window shape from a different side of the window. In the middle of a shape process, while you still have the left or middle mouse button down, simultaneously press and hold the right mouse button down. When

you do this, the forceps cursor (0) will appear. Move this cursor to the window corner you want to move and still holding the left or middle mouse button down, release the right mouse button. You will now be able to reshape the window from the corner you just selected.

If you have too many windows on the screen and you are having problems accessing them, you probably want to try shrinking some windows.

Shrink

Removes the window from the screen and brings up its icon. The window can be restored by clicking the middle mouse button inside the window's icon, or by selecting **Expand** from the right button icon menu. Some icons are shown below.

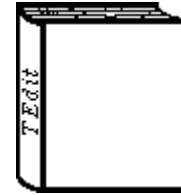


Figure 7-19. TEdit-window shrink icons look like books.

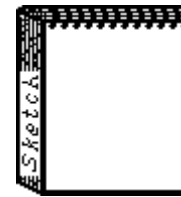


Figure 7-20. Sketch-window shrink icons look like sketch pads.



Figure 7-21. FileBrowser-window shrink icons look like filing cabinet drawers.

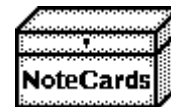


Figure 7-22. The NoteCards MenuBox shrink icon looks like a recipe file box.

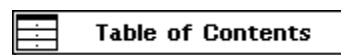


Figure 7-23. Notecard shrink icons look like link icons with an extra border.

The Icon Menu

Icons are a variant of windows and have a menu similar in form and function to the window menu.

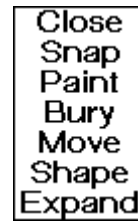


Figure 7-24. The icon menu.

For the options Close, Snap, Paint, Bury, Move, and Shape see the discussion above under the "Window Menu" heading.

Expand

Restores the window associated with the icon accessed and removes the icon image from the screen. You can also expand an icon by clicking the middle mouse button inside the icon.

The Background Menu

If the right button is pressed while the cursor is not in any window, the background menu appears.

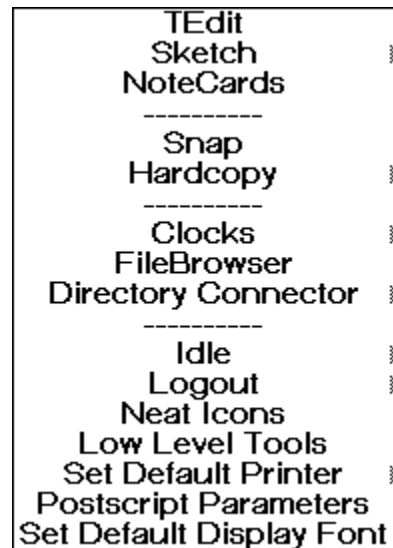


Figure 7-25. The background menu.

TEdit

Opens a new TEdit window and starts a new TEdit session.

Sketch

Opens a new Sketch window and starts a new Sketch session.



Figure 7-26. The Sketch submenu options.

Page sized sketch

Allows you to start a sketch which will fit exactly on an 8 1/2 x 11 inch sheet of paper when held in the normal vertical or portrait position.

Landscape sketch

Allows you to start a sketch which will fit exactly on an 8 1/2 x 11 inch sheet of paper when held in a horizontal or portrait position.

Sketch, from a file

Allows you to start a preexisting sketch which is stored in a file.

NoteCards

Opens the NoteCards MenuBox Icon which is the interface for a NoteCards session.

Snap

The same as the window menu command Snap described above.

Also, TEdit, Sketch, and NoteCards allow information to be shift-inserted at the current cursor position by selecting an area of the screen with the SHIFT key held down. To shift-insert the bitmap of a snap into an editor, position the cursor where you want the image to appear, hold the SHIFT key down, press and hold down the right mouse button in the background, and select **Snap** from the single item menu which appears. Finally, sweep out the area for the snap and release the mouse button and the SHIFT key. The snap will appear in the editor where the cursor was positioned. Note, sometimes it is necessary to scroll or redisplay the window for the snap to appear.

Hardcopy

Prompts for a region on the screen, and sends the bit map image to the default printer. Note that the region can cross window boundaries.

Like the window menu **Hardcopy** command discussed above, you can print to a file or specify a different printer than the default by using the **Hardcopy** submenu. See the discussion of the Hardcopy command under the "Window Menu" section above, for more detail.

Clocks

Starts an analog clock and places it on the screen. The Clocks submenu gives you access to a digital clock.



Figure 7-27. The Clocks option on the background menu.

For a complete discussion of this menu option, see Chapter 15, Other Tools.

FileBrowser

Opens a FileBrowser window which prompts you for a directory to browse.

For a complete discussion of this menu option, see Chapter 14, The FileBrowser.

Directory Connector

Opens a window which displays your currently connected directory, and allows you to change directories.

For a complete discussion of this menu option, see Chapter 15, Other Tools.

Idle

Enters idle mode, which blacks out the display screen to save the phosphor. Idle mode can be exited by pressing any key on the keyboard or mouse. This menu command has subitems that allow the user to interactively set idle options to erase the password cache (for security), to request a password before exiting idle mode, to change the timeout before idle mode is entered automatically, etc.

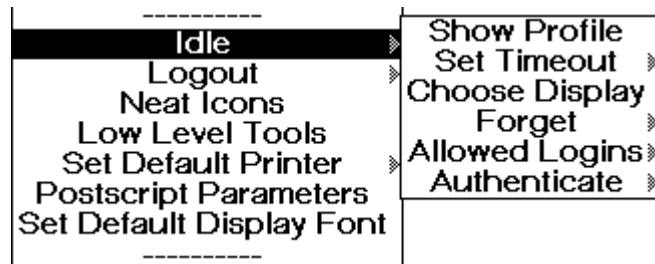


Figure 7-28. The Idle submenu.

If either shift key is pressed while NoteCards is in idle mode, the current user name and the amount of time spent idling are displayed in the prompt window. This information appears as long as the shift key is held down.

Show Profile

Displays the current idle profile in the system prompt window.

Set Timeout

Sets the amount of time that the machine will wait for a key stroke or mouse click before automatically going into idle mode.

When you select this option the system brings up a number pad on which you can enter the new idle time out. An idle time out of zero sets the machine so that it will never go into idle mode.

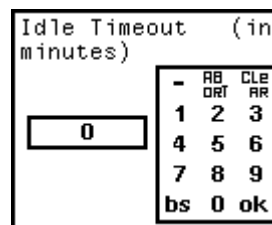


Figure 7-29. The number pad menu requesting a new idle timeout duration.

The submenu option **Never**, will set the machine so that it never goes into idle mode. We counsel against using this option as it can result in the phosphor on your screen being damaged.

Choose Display

Allows you to select what will be displayed on the blacked out screen when the machine is in idle mode. Selecting this option brings up the two-choice menu shown below.

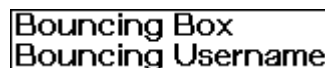


Figure 7-30. The idle display menu.

Bouncing Box

Chooses the Envos logo as the idle display.



Figure 7-31. The **Bouncing Box** Envos logo.

Bouncing Username

Chooses the your username as the item to bounce around on the screen while in idle mode.

Forget

If set to "Don't" (or NIL), your password is not erased when idle mode is entered. Default is "Do" (or T(rue)) erase password. The initialization file distributed with the NoteCards system resets **Forget** to "Don't."



Figure 7-32. The **Forget** submenu.

Note: If the password is erased, any programs left running when idle mode is entered will fail if they try doing anything requiring passwords, such as accessing file servers.

Allowed Logins

Determines who can exit idle mode.

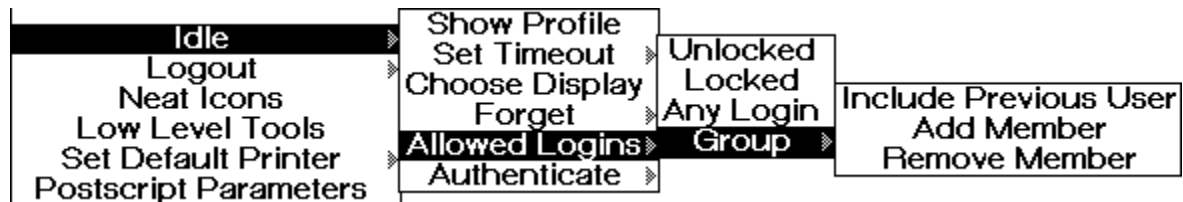


Figure 7-33. The "Allowed Logins" submenus.

- Unlocked** Sets the system such that login is not required to exit idle mode. Initialization file value is "NIL."
- Locked** Lets only the previous user exit idle mode. Initialization file value is "(T)."
- Any Login** Require login, but let anyone exit idle mode. Login overwrites the previous user's user name and password each time idle mode is exited. Initialization file value is "(*)."
- Group** Allow any members of a specified group to exit idle mode. Figure 7-33 shows the **Group** submenu which allows you to add and delete group members.

Authenticate

The value of this property determines what mechanism the system uses to check passwords.

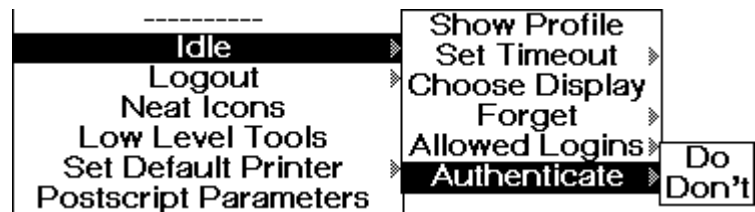


Figure 7-34. The Authenticate submenu.

If the value of this property is "Do" or "T(rue)," the Xerox network system protocol is used to do the authentication.

If the property value is "UNIX," the Sun Operating System is used to do the authentication. This option is set in the standard initialization file. See Appendix C for a complete discussion.

If the value of this property is "Don't" or NIL, the password is not checked. Any password is accepted.

Logout

Saves the NoteCards virtual memory working image to a file and returns the system to UNIX. This is the normal way to return to UNIX.

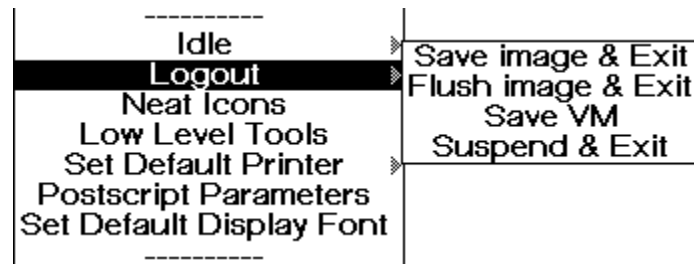


Figure 7-35. The Logout option and its submenu on the background menu.

Save image & Exit

The same as Logout.

Flush image & Exit.

Returns the system to UNIX without saving the working image. Use this option when you have saved all your work to files and you do not care if the system comes up in the same state it was in when you left it.

Save VM

Saves the NoteCards virtual memory working image to a file but does not return the system to UNIX. Use this option frequently if you are concerned about power failures or other problems which may cause your machine to crash.

Suspend & Exit

Suspends the NoteCards session and returns control to UNIX. You restart the NoteCards session by entering the command `fg` (foreground), at the UNIX prompt.

Low Level Tools

This menu option provides people extending the NoteCards system access to the Lisp executives, the process status window, and allows them to set the **Show TEdit Props** flag. The NoteCards user should never access these options except under the instruction of a developer or support person.

Set Default Printer

Allows you to specify which printers you want to print to and which one is the default.

For a complete discussion of this menu option, see Chapter 16, Printing.

Postscript Parameters

Allows you to control the way PostScript documents are printed.

For a complete discussion of this menu option, see Chapter 16, Printing.

Set Default Display Font

Brings up a series of menus which allow you to specify what fonts will be used on the screen. This is a particularly useful option if you are using a high resolution screen and want to increase font size or for people who are vision impaired.

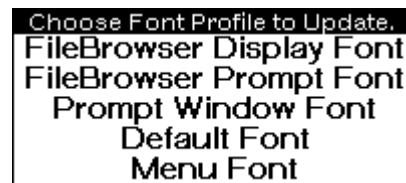


Figure 7-36. The "Choose Font Profile to Update" menu.

To change a display font, select the **Set Default Display Font** background menu option and choose the font you want to change from the "Choose Font Profile to Update" menu. Each of the font options is discussed below. Once you have selected the font you want to modify, the "Please select a font:" menu will appear. Select the font family, size, and face you want then select **DONE**. **RESET** makes the menu display the current setting for the default display font you are modifying. **ABORT** terminates this operation and closed the "Please select a font:" menu.

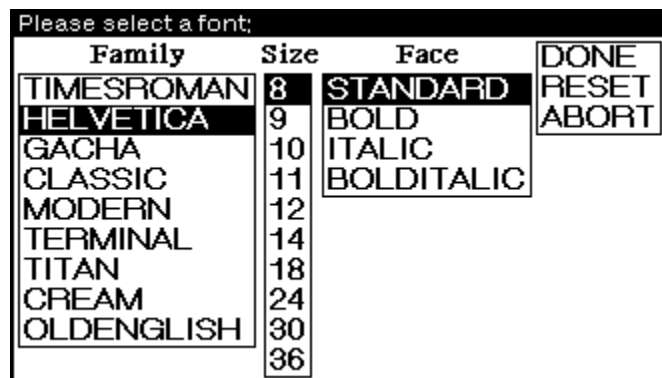


Figure 7-37. The "Please select a font:" menu.

If you choose a font which does not exist or the system is unable to find, the old font specification remains unchanged.

If the system is unable to find a font which you know to exist, check the setting of the display font directory list. This is done with the **Show Font Directories** subsubmenu option off the **Set Default Printer** background menu option. See Chapter 16, Printing, for a complete description of how to use this option.

FileBrowser Display Font

The font in which the information in the main display window is printed, initially 10-point Gacha.

FileBrowser Prompt Font

The font in which FileBrowser prompt messages are printed. Initially 8-point Gacha. Changing this value only affects new FileBrowsers created from the background menu. Existing FileBrowsers are unaffected.

Prompt Window Font

The font used to print information in the black system-prompt window.

Default Font

The starting font used in TEdit and Sketch windows as well as all other windows which do not have an explicitly specified font associated with them.

Note that the default font for NoteCards cards is specified from the "NoteCards System Parameters" menu. See Chapter 13, System Parameters for a complete discussion of how to specify this default font.

Menu Font

The font used in all system menus. The menu font is used in the window, icon, background, and other menus.

[This page intentionally left blank]

A link is a connection between a source card and a destination card. A link icon represents the link.



Figure 8-1. A link-icon image.

By clicking on the link icon, you can display the destination card. Links are the glue which holds together the relationships and conceptual structures represented in a notefile.

This chapter explains:

- What links and link types are.

- How links are accessed and used.

- How to create links.

- How to delete and undelete links.

- How to tailor links to your taste.

- What cross-file links are.

Links and Link Types

Link types specify the nature of the relationship between the source and destination cards. Some typical link types are Question, Answer, and Explanation.

Two groups of link types exist in the NoteCards system; one group is system-reserved, the other is user-specified. Some of the system-reserved links support the FileBox hierarchy that NoteCards helps you to create. The user-specified links add a relational dimension to information structuring. Here you can link cards together to form networks. These networks allow you to represent the interconnections between various ideas or pieces of information, independent of any categorization into topic areas. In contrast, the set of FileBoxes forms a strict hierarchy typically representing a breakdown of information into subtopics or subcategories. By strict hierarchy, we mean that you are not allowed to create circular lists of FileBoxes. Circular lists of other card types are allowed, however.

System-Reserved Link Types

SubBox	A link pointing to a FileBox from another FileBox.
FiledCard	A link pointing from a FileBox card to any non-FileBox card. Text, Sketch, Graph, Browser, Search, LinkIndex, and Document cards are non-FileBox cards.

BrowserContents	A link pointing to a card or box from a Browser card. This type of link is traversed when you bring up any of the cards or boxes represented as link icons in a Browser card.
ListContents	A link pointing to a card or box from a Search card, enabling you to bring up any of the cards or boxes found during the search process.
LinkIndexBackPtr	A link pointing to a card or box from a LinkIndex card. This type of link is built when you specify that a LinkIndex card should have back pointers to the cards and boxes being indexed, i.e., link icons should be included in the LinkIndex card.
DocBackPtr	A link pointing to a card or box from a Document card. This type of link is built when you specify that a Document card should include back pointers to each card and/or box used to build the document.

User-Specified Link Types

User-specified links indicate the logical relationships between cards. The system comes with three predefined user-specified link types, Comment, See, and Unspecified. However, the system allows you to create new link types as you need them. Typical link types users create include Explanation, Example, Question, Answer, Next, and Source. Choose your link types carefully as they determine the lines along which you will be able to extract information from your notefiles with Document cards.

Randy Trigg, one of the developers of the NoteCards prototype at Xerox PARC, proposed a long list of standard link types for users to follow in categorizing the relationships between their notes. A partial list is included here to give you some idea of the possibilities.

Citation

- C-source

- C-credit

- C-leads

Argument

- A-deduction

- A-induction

- A-analogy

- A-intuition

- Background/Future

- Refutation/Support

- Methodology/Data

- Solution

- Continuation

- Correction/Update

- Simplification/Complication

- Explanation

- Summarization/Detail

- Alternate-view

- Rewrite

- Generalization/Specification

- Abstraction/Example

- Formalization/Application

Note that some long-time users of the system use only one link type. NoteCards gives you a great deal of flexibility. Just because some functionality is there does not mean you should feel obligated to use it if it is not helpful to your task.

Link Directions and Link Ends

Links have a single start point and a single end point. However, whenever you create a link from one card to another, the system automatically builds a backwards link so that it is possible to traverse a linked sequence of cards in either direction. The result is that all cards are linked bidirectionally.

A link's start point is either the card itself or somewhere in the card's contents. In the case of a text card, this would be somewhere within the body of the card's text. A link's end point is always another card. Thus, there are three distinct link categories. There are also a few terms which refer to different groupings of these categories.

Link Categories

To-Links	Refer to links whose start point is in a card's contents and whose end point is another card. These links are represented as link icons in a card's body.
Global To-Links	Refer to links whose start point is a card itself and whose end point is another card. These links are not represented by link icons in a card's body. To see and access these link icons you must choose the Show Links option from a card's left-button title-bar menu.
From-Links	For every to-link and global to-link there is a corresponding from-link which points from the destination card to the source card. All from-links are global.

Other Link Terminology

Local Links	Links whose start point is somewhere in the contents of a card.
Global Links	Links whose start point is the card itself and not some point in the card's contents. These links are not represented by link icons in a card's body. To see and access these link icons, you must choose the Show Links option from a card's left-button title-bar menu.
Forward Links	Include both to-links and global to-links. This term is used when setting Browser and LinkIndex card parameters.
Backward Links	The same as from-links. This term is used when setting Browser and LinkIndex card parameters.
Back Links	System-generated to-links. These links point from a Document or LinkIndex card <i>back</i> to the source cards from which the Document or LinkIndex was built. Hence the name back link.

Link Types vs. Card Types

Link types and card types are frequently confused but they are totally independent of each other.

Link types classify the assigned relationship between two cards. Link types frequently include Explanation, Question, Answer, and Citation.

A card's type specifies the functional capabilities of that card and is associated with a text or graphics editor. The eight card types are Text, FileBox, Search, LinkIndex, Document, Sketch, Graph, and Browser. The Text, FileBox, Search, LinkIndex, and Document cards are all based on the text editor TEdit. The Sketch card is based on the Sketch editor and the Graph, and Browser cards are based on the graph editor Grapher.

Link Access and Use

Link Icon Functionality

A link icon is an active region with an associated menu and four functions. The most frequently used functionality is to click the left mouse button in the body of the link icon to traverse the link to the destination card. The middle mouse button brings up a menu which allows you to traverse the link, change the link type, change the card title and change the link display mode.

Active Regions of Link Icons

The entire link icon except for 10 pixels at the left edge and 10 pixels at the right edge responds to mouse clicks by carrying out a NoteCards action.

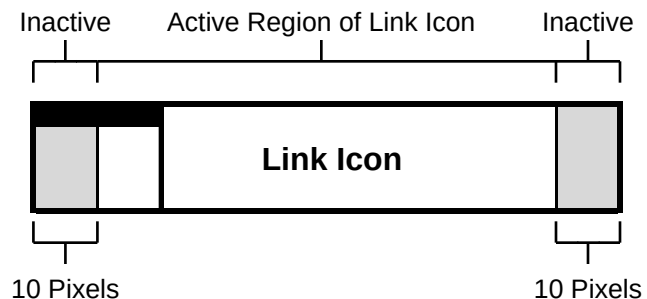


Figure 8-2. Link icon active and inactive regions.

Clicking in the "inactive" 10-pixel strips at the left and right edges of the link icon selects the link icon, in text-based cards (Text, FileBox, Search, LinkIndex, and Document cards), for delete, copy, and move operations, without traversing the link. This region does not exist on sketch- and graph-based cards (Sketch, Graph, and Browser cards).

The inactive area can be somewhat difficult to access without touching the active region of the card. One way to access it is to hold the left or middle mouse button down and slide into the link icon's inactive region. You have selected the link icon when it appears underlined.

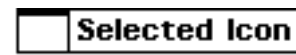


Figure 8-3. A selected icon

If you go too far and the card redisplay in reverse video, simply slide the mouse cursor out to the edge of the link icon and watch for it to underline again. Once the icon appears underlined release the mouse button.

If a link icon is so narrow that its active region would be less than 10 pixels wide, the active region is set to the middle 50% of the link icon width, with 25% at left and 25% at the right being the inactive strips. If the link icon is less than 10 pixels in width altogether, then the link icon will have no inactive strips. Its entire area will be active.

Mouse-Button Actions in Link Icons

The exact response to a mouse click in the active region of a link icon varies depending on a number of conditions. For more general mouse button behaviors, see Chapter 7, *The User Interface*, as well as the *User's Guide to TEdit* and the *User's Guide to Sketch*.

Left Button	Brings up the destination card represented by the link icon.
Middle Button	Brings up the link icon menu which allows you to change the link type, title, and display mode.
Right Button	Following a left- or middle-button click elsewhere in a TEdit-based card, selects a region of the window for moving, copying, or deletion which can include link icons.

Copy/Shift or Move/Shift-Control key plus mouse-button operations are three-step operations. First, you specify the destination window by clicking the left mouse button to position the flashing cursor in the window. Second, hold the appropriate keyboard key down. Third, while still holding the key down, use the mouse to select the piece of text/graphics to be operated on. Once you have completed your selection, release the keyboard key and what you have selected will be copied or moved to the destination window.

Link icons should only be copied or moved to NoteCards windows. TEdit and Sketch, when used outside of NoteCards text-based and sketch-based cards, will not understand link icons.

Copy/Shift key + Left Button	This operation is frequently referred to as a shift-select, occasionally as a copy-select. This operation copies a link icon, or link-icon name and link, to another card. If the flashing cursor is in the body of a card, holding the SHIFT or COPY key down and clicking the left mouse button on a link icon copies the link icon and the link it represents to the position of the caret cursor in the destination card.

If the flashing cursor is a crosshairs and is in a "Selecting cards to file" prompt window, holding the SHIFT or COPY key down and clicking the left mouse button on a link icon will copy the link icon name and link to the select-card prompt window. This operation is used when you are inserting links in cards or filing cards in a FileBox card.



Figure 8-4. A select card prompt window.

Copy/Shift key+ Middle Button

Same as left button.

Copy/Shift key + Right Button

Holding the COPY or SHIFT key down and pressing the right mouse button following a left- or middle-button click elsewhere in a text-based card, selects a region of the window for copying which can include link icons.

Move/Shift-Control + Left Button

Moves a link and link icon from one card to another card by deleting the link in the first card and inserting it into the second. To move a link, position the caret cursor in the destination window where you want the link to appear and, holding the MOVE key down or holding the CONTROL and SHIFT keys down simultaneously, select the link icon you want to move with the left mouse button. When you release the key or keys you are holding down, the link icon will move to its new position.

Move/Shift-Control + Middle Button

Same as left button.

Move/Shift-Control + Right Button

Holding the MOVE key or SHIFT and CONTROL keys down and pressing the right mouse button following a left- or middle-button click elsewhere in a text-based card, selects a region of the window to be moved which can include link icons.

CONTROL key plus mouse button operations are two-step operations. First, hold the CONTROL key down. Second, while still holding the key down, use the mouse to select the piece of text/graphics to be deleted. Once you have completed your selection, release the CONTROL key and what you have selected will be deleted.

Control key+ Left Button

Deletes a link and link icon from a card. Holding the CONTROL key down and pressing the left mouse button on a link icon will select the link icon for deletion. This is shown by displaying the link icon in reverse video.



Figure 8-5. A selected link icon.

The link icon and the link it represents are deleted when you release the CONTROL key. You can cancel this operation before you release the CONTROL key by clicking elsewhere in the window to deselect the link icon.

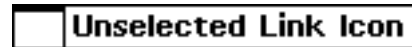


Figure 8-6. An unselected link icon.

You can undo this operation, after you have released the control key, by hitting the UNDO key.

Control key + Middle Button

Same as left button.

Control key + Right Button

Holding the CONTROL key down and pressing the right mouse button following a left- or middle-button click elsewhere in a text-based card, selects a region of the window for deletion which can include link icons.

It is also possible to just backspace over a link icon or to select it and then hit the DELETE key. Note that on some Sun keyboards, the backspace key has the word "Delete" printed on it and that the NoteCards DELETE key is one of the function keys, usually the key labeled "L10."

Viewing Local and Global Links

All links are stored in three places.

- 1) as a to-link on a source card
- 2) as a from-link on a destination card

and either

- 3a) for local to-links, in the card's contents as a link icon
- 3b) for global to-links, in the card's global links list

Only the local to-links, the links in a card's contents, are immediately visible.

To see all of the links to and from a card, hold the left mouse button down in the card's title bar and select the **Show Links** option.



Figure 8-7. The card menu with the **Show Links** option selected.

This will open a window titled: "List of Links". This window always opens with the same initial height. If the list of links is longer than three or four, you will need to scroll or reshape the window to see

the rest of the links. The link icons in this window have the standard left-button functionality, you can traverse links, but no middle-button functionality. Links can also be copied, moved, and deleted from this window.

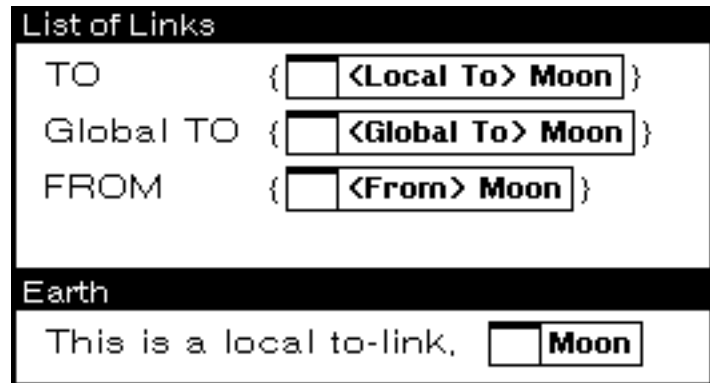


Figure 8-8. A "List of Links" window showing links to and from the "Earth" and "Moon" cards.

The headings to the left of the links are active but have no operations associated with them.

You can close the "List of Links" window by holding down the left or middle mouse button in the window title bar and selecting **Quit** from the single-option menu, or by pressing the right mouse button in the title bar and selecting the **Close** option from the standard window menu.



Figure 8-9. The "List of Links" left button menu.

Unfiled and Lost Cards

All cards, including FileBox cards, must be filed in at least one FileBox. This assures that you will have a pointer to your information and helps to remind you to hierarchically organize your information in addition to specifying other types of relationships. When you close a card without linking it to a FileBox card, the closed card is placed in the "To Be Filed" FileBox. You can access this FileBox card by holding the middle mouse button down on the notefile-banner **Special Cards** menu option and selecting the **To Be Filed** option from the menu which appears.

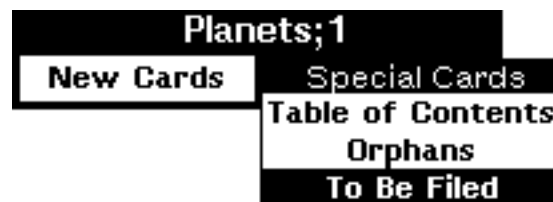


Figure 8-10. The **To Be Filed** option on the "Special Cards" menu.

From this FileBox all "unfiled" cards are accessible. It is possible to force yourself to file cards in a FileBox by setting the **Force Filing**

Edit Parameter. See Chapter 13, System Parameters, for a complete explanation.

You can remove all FileBox links to a card by accident or by design. When you do this, the following message is displayed in the System Prompt Window. "You have just unfiled *card name* from its last filebox. It is being filed in the Orphan FileBox." You can access this FileBox card by holding the middle mouse button down on the notefile-banner **Special Cards** menu option and selecting the **Orphans** option from the menu which appears.

Finally, you can delete card links from the "Orphans" FileBox. When you do this the following message is displayed in the card's prompt window. "You have just deleted the last filing link to *card name*. The Search operation can be used to find it." At this point, if there are no other cards with links to this card, it becomes a lost card and there is no simple way to access the card even though it still exists. As the message says, though, you can retrieve lost cards by building new links to them with a Search card. However, unless you remember some fragment of the card's name, using a Search card to find lost cards can be a time consuming task in a large notefile.

Creating Links

The links you create always run from a source card to a destination card. References to the source card refer to the card with the link icon. References to the destination card refer to the card which opens, or flashes if already open, when you click on the link icon.



Figure 8-11. An illustration of a link icon pointing to its destination card.

Creating Links in Text Cards

The procedures for creating links are basically the same for all cards. We will use Text cards as the model, and then discuss the differences between Text cards and other cards.

Insert Link

You use the **Insert Link** command to create a link originating in the source card's contents and terminating at another card. This is a multi-step process. First, you place the caret cursor in the card at the position where you want the link to appear by clicking in the source card's window. If the type-in process belongs to another card or window this usually takes two mouse clicks, one to make the card the active card and one to position the mouse cursor. Once you have established the location for the link icon in the

source card, you hold the left mouse button down in the source card's title bar and select the **Insert Link** option.

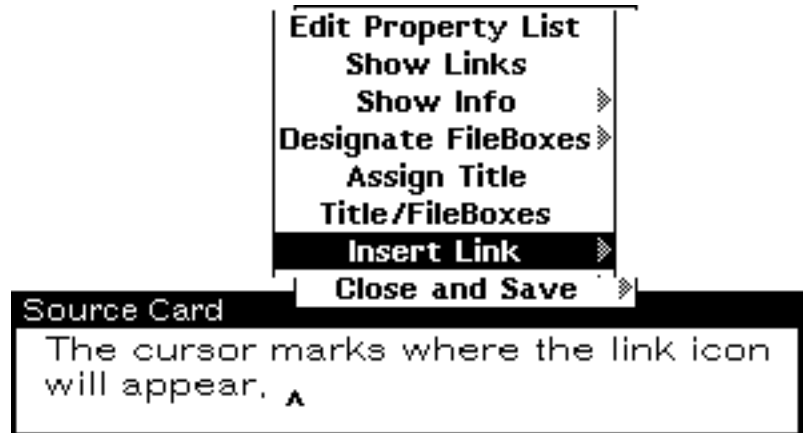


Figure 8-12. The card menu with the **Insert Link** option selected.

At this point the "Link Type" menu will open, asking you to specify the link type.



Figure 8-13. The "Link Type" menu with the **--New Link Type--** option selected.

Choosing appropriate link types is important as they will determine the ways you will be able to extract information from your notefiles. For more information on link types see the section User-Specified Link Types in this chapter and the section on Document cards in Chapter 12, System Cards. You can create new link types as you need them by selecting the **--New Link Type--** option and typing the name of the new link type into the prompt window which will appear above the source card. When you type a carriage return to this prompt window, without defining a new link type, the system assigns the link type the value "Unspecified."

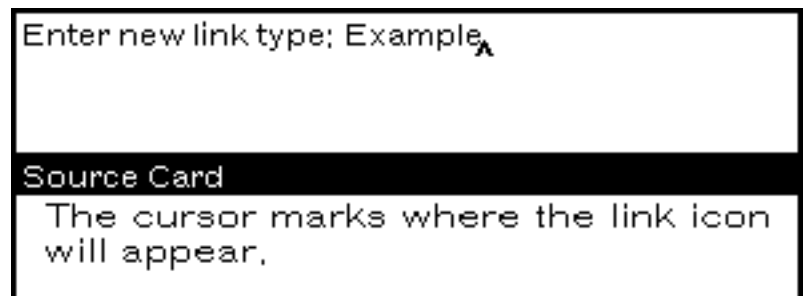


Figure 8-14. A note card prompt window requesting a new link type.

After you have selected an existing link type or defined a new one, the system opens a "Selecting Note Card" window and prompts you to shift-select the card UID from the destination card.

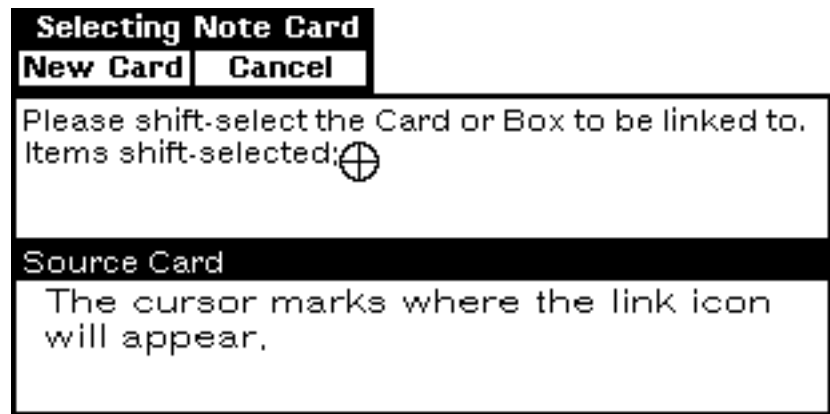


Figure 8-15. A "Selecting Note Card" window prompting the user to shift-select a card ID from the destination card's title bar.

A card UID is shift-selectable from three locations,



When shift-selecting the card UID, there are two distinct behaviors. From the link icon, it is necessary to lift the Shift or Copy key after each selection. This is the expected behavior when shift-selecting anything from an edit window. From the title bar and shrink icon, it is not necessary to lift the Shift key. The transfer happens immediately.

What you see transferred is the destination card's title, but NoteCards also maintains an identifier for each card which is unique across all cards and notefiles. This is what you are really shift-selecting from one card to another. And this is also the reason why it is not possible to shift-select or type text into this window.

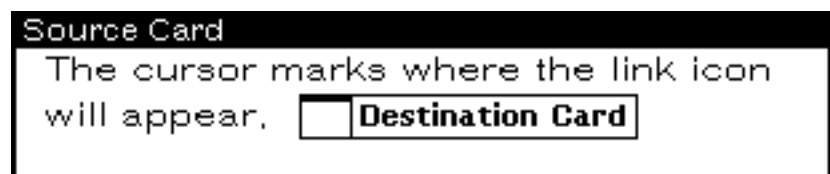


Figure 8-16. A link icon to a destination card as it appears in a source card.

At this point, the new link icon will appear in the source card. From now on, when you click on this link icon, it will open the destination card, or flash the destination card if it is already open.

Selecting **New Card** from the menu on top of the "Selecting Note Card" prompt window will bring up a menu allowing you to create a new card, of any type, which will be the destination card.

Selecting **Cancel** from this menu, cancels the entire **Insert Link** operation.

At any point in this process, it is possible to transfer the type-in process from a prompt window to some other window by clicking in the window you want to use. The prompt window will stay open waiting for you to complete the operation. When you are ready to finish inserting the link, simply click in the prompt window to make it the active window, shown by the flashing crosshairs, and continue where you left off.

Insert Links

If you need to insert multiple links of the same link type at the same location in a source card, **Insert Links** will save you from inserting each link individually.

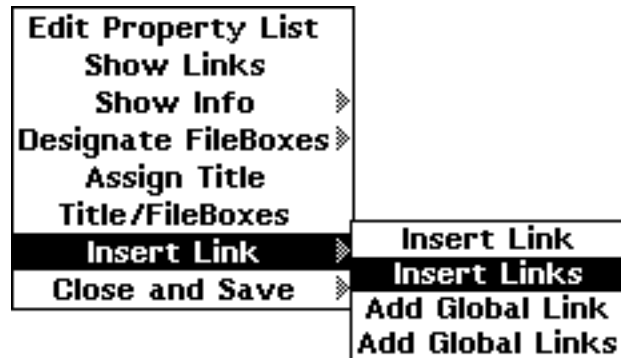


Figure 8-17. The card menu with the **Insert Links** option on the **Insert Link** submenu selected.

Insert Links works exactly as **Insert Link** except you will be allowed to select multiple destination cards. It is possible to shift-select a mixture of text and link icons from a card. In this case, the source card's prompt window will ignore all the extraneous text. To complete the insert operation, either type a carriage return or select the **Done** menu option.

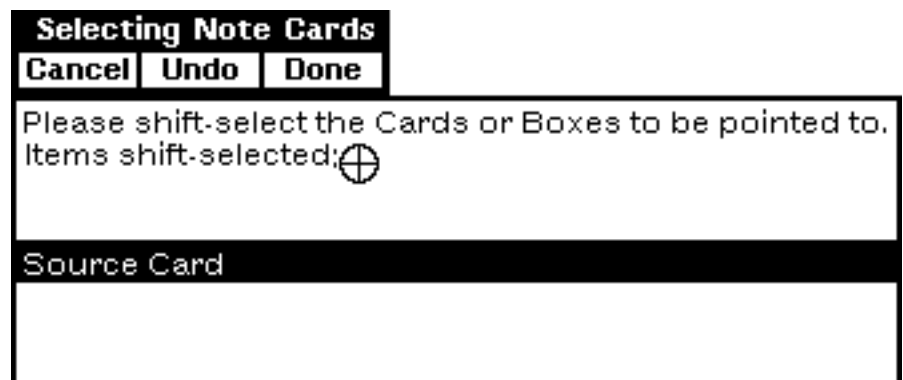


Figure 8-18. A "Selecting Note Cards" prompt window.

The **Cancel** option terminates the entire link insertion operation.

Undo allows you to remove the last item from the list. You can select **Undo** until you have removed all the items from the list.

Add Global Link

Global links are not rooted in a card's contents. For this reason we draw the contrast between *inserting* a link into a card's contents and *adding* a global link to a card.

A second result of this difference is that the procedure for **Add Global Link** is somewhat different from that for **Insert Link**. To add a global link, hold the left mouse button down on the card's title bar and slide off **Insert Link** to access the submenu and select the **Add Global Link** option.

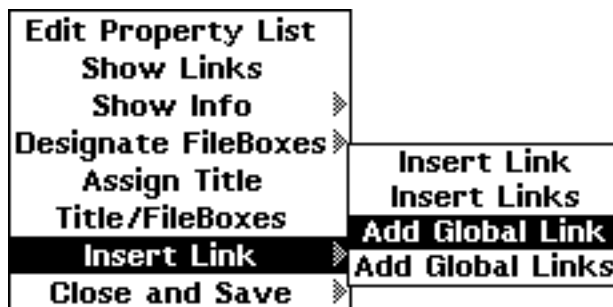


Figure 8-19. The card menu with the **Add Global Links** option on the **Insert Link** submenu selected.

Because you are adding a link to a card itself and not inserting a link into a card's contents, it is not necessary to first position the caret cursor to select the link icon's insertion point. Note that the link will not be visible unless you use the **Show Links** option to show the links to and from that card. Aside from these few minor differences the procedure is exactly the same as that for **Insert Link**.

Add Global Links

Add Global Links functions just like a combination of **Add Global Link** and **Insert Links**. It allows you to add multiple global links of the same link type to a source card without adding each link individually. If you want the global links to have different link types, they must be added one by one using **Add Global Link**.

Creating Links in Other Card Types

There are three basic card types in NoteCards, text-based, sketch-based and graph-based cards.

Text-Based Cards

Text-based cards are based on the TEdit editor and can contain the output from sketch-based and graph-based cards. Text-based cards include Text, FileBox, Search, LinkIndex, and Document cards. The link creation procedure for all text-based cards is the same as for Text cards.

Sketch-Based Cards

Sketch-based cards are based on the Sketch editor. There is only one sketch-based card and it is the Sketch card.

The link icon insertion procedure differs from that for text-based cards only concerning *when* the new link icon is positioned in the card. For text-based cards, the cursor is positioned where you want the link to appear and then one of the link insertion options is selected. For sketch-based cards, the order is reversed. The

menu option is selected first and then, after you have indicated which card you want to link to, you move the cursor into the Sketch card and position the link icon.

Graph-Based Cards

Graph-based cards are based on the graph editor. Graph and Browser cards are graph-based cards.

As with sketch-based cards, the link icon position is selected after the card you are linking to.

Deleting Links

When you delete a link icon you are deleting the to-link that points from the source card to the destination card and also the from-link which points from the destination card back to the source card.

To understand what happens when you delete a link, create a Source Card and a Destination Card. Next insert a link from the Source Card to the Destination Card, and open the "List of Links" windows on each card by selecting the **Show Links** option from the left-button title-bar menu. When you delete one link, all the links will be deleted.

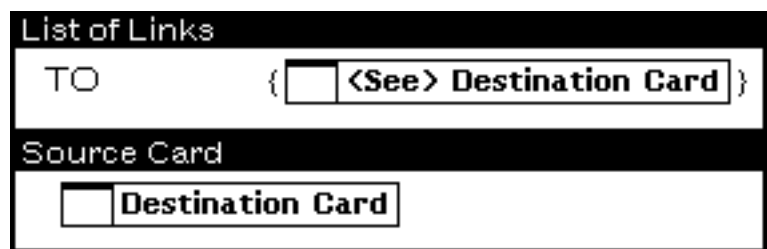


Figure 8-20. A "List of Links" window showing a to-link to a destination card.

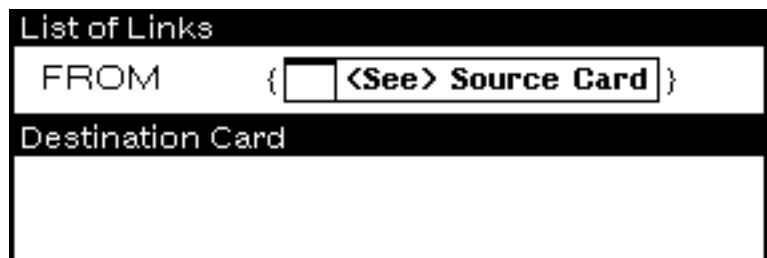


Figure 8-21. A "List of Links" window showing a from-link pointing back to a source card.

The delete procedure for links depends on where you are accessing the link and the card type you are deleting the link from.

Deleting Links from Card Contents

Text-Based Cards

Link icons in the body of text-based cards can be deleted in several ways.

You can select the link icon with the mouse by clicking in the *inactive* region of the link icon and hitting the delete key. Also, you can select a region of text in a text-base card with a sequence of left+right or middle+right mouse buttons and hit the delete key to delete the selected region which can include a mixture of text and link icons.

You can also hold the control key down and select the link icon by pressing the left or middle mouse button in the link icon active region, when you release the control key, the link icon will be deleted.

Finally, you can simply backspace over a link to delete it.

All of these operations can be undone by hitting the Undo key.

For more detail on inactive regions of link icons and coordinated keyboard-plus-mouse operations, see the Link Access and Use section above and as well as the TEdit documentation.

Sketch-Based Cards

You can delete link icons in the body of sketch-based cards by bringing up the sketch menu, if it is not permanently attached to the right side of the sketch window, and selecting the delete option. Next, click in the link icons control point which is the small box attached to one corner of the link icon.



Figure 8-22. A selected link icon in a Sketch card.

This operation can be undone by hitting the Undo key.

For more detail on this and other Sketch operations see the Sketch documentation.

Graph-Based Cards

To delete a link from a Graph card, hold the right mouse button down in the Graph card window, select the **Delete Node** option, and then click on the card you want to delete. When you confirm, the link will be deleted.

Note, this operation cannot be undone.

Because a Browser card is an overview of a link network in addition to being a member of that network, the meanings of delete operations in Browser cards is not straightforward. Please turn to the section on Browser cards in Chapter 11, System Cards, for a detailed explanation of links in Browser cards.

Deleting Links from the Show Links Display

Deleting links from the show links display is very similar to deleting links from text-based cards.

You select the link icon with the mouse by clicking in the inactive region of the link icon and hitting the delete or backspace key.

You can also hold the control key down and select the link icon with the mouse by pressing the left- or middle-mouse button in the link-icon active region, when you release the control key, the link icon will be deleted. This second way is frequently the easier of the two, as it does not require you to access the narrow inactive regions on either end of the link icon.

Note, this delete operation cannot be undone.

Tailoring Links

It is possible to modify the appearance of links to suit your personal tastes and the task at hand. Links can be modified using the middle-button icon menu and the NoteCards "System Parameters" menu.

Link Ops Menu

You bring up the "LinkOps" menu by holding the middle mouse button down in the active region of the link icon.



Figure 8-23. The "Link Ops" menu.

Bring Up Card/Box

Performs the same function as clicking the left mouse button in the link icon's active region. It traverses the link to the destination card and opens it, or flashes it if it is already open.

Change Link Type

Allows you to choose a different link type for the link you select.

Change Card Title

Changes the title of the card the link points to and the titles shown in all the link icons pointing to that card. The one exception to this statement is when you have created one-way cross-file links. These links update only when you access the link icon which points to the changed card.

Change Display Mode

Selecting **Change Display Mode** brings up the "Display Mode?" menu. This menu allows you to override the display mode defaults for the individual link icon you have selected.

Display Mode?			
Title?	LinkType?	AttachBitmap?	
Yes	Yes	Yes	DONE RESET ABORT
No	No	No	
System Defaults	System Defaults	System Defaults	

Figure 8-24. The "Display Mode?" menu.

There are two levels of defaults for link icon display modes. Each card defined in the NoteCards system sets how links will be displayed in its contents. This is the first level of defaults and you have no control over them. These defaults can be set to three values, "Yes," "No" and "System Defaults." These are the values which appear in the "Display Mode?" menu shown above. If the value is set to "System Defaults," NoteCards consults the System Parameters to determine whether or not to display that particular attribute. See Chapter 13, System Parameters for how to set link default display modes.

With all the values set to "No," the link appears as a small rectangular box.



Figure 8-25. A link icon with all display mode values set to "No."

With all the values set to "yes," the links have the following appearance.

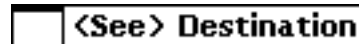


Figure 8-26. A link icon with all display mode values set to "Yes."

The link type is shown in angled brackets and the card title is to the right of the link type. The card-type bitmap is the square piece on the left of the link icon and varies in design depending on the card type.



Represents Text cards.



Represents FileBox cards.
The image is a stylized file drawer.



Represents Sketch cards.
The image is a stylized sketch.



Represents Graph cards.
The image is a stylized letter "G."



Represents Browser cards.
The image is a stylized Browser.



Represents Search cards.
The image is a stylized eye.



Represents LinkIndex cards.
The image is a stylized list of link icons.



Represents Document cards.
The image is a stylized linear list of note cards.



Represents cards for which the type is unknown occurs with cross-file links.

System Parameters for Links

There are nine system parameters to enable you to customize the appearance of links. They are **Link Icon Font**, **Link Icon Border Width**, **Link Icon Multi Line Mode**, **Cross File Link Mode**, **Link Icon Max Width in Pixels**, **Link Icon Show Title Default**, **Link Icon Attach Bitmap Default**, **Link Icon Show Link Type Default**, and **Use Deleted Link Icon Indicators**. For a complete discussion of these parameters, see Chapter 13, System Parameters.

Cross-File Links

The NoteCards system allows you to have links from one notefile that point to cards in another notefile. These types of links are known as cross-file links. See also Chapter 13, System Parameters for how to set the cross-file link mode.

Appearance

Cross-file links are indicated by an arrow bitmap on the right-hand side of the link icon.



Figure 8-27. A cross-file link indicated by an arrow bitmap on the right-hand side of the link icon.

It is also possible to cross-file cards. This is to say that you can file a card from one notefile in the FileBox of another notefile. Note that this is only possible after you have first filed it in its origin notefile. This is, in effect, just another way of creating a cross-file link. Cards are always physically filed in their origin notefiles.

When a cross-file link is displayed, and the notefile the link points to is closed, then the link can have the following form.



Figure 8-28. A cross-file link to a closed notefile. The question-mark bitmap indicates that the card cannot be accessed to determine its card type.

The card-type bitmap on the left of the link icon is a question mark because the card type is stored on the card and this link has no access to the card because the notefile is closed. As a result, the

link cannot display the appropriate card-type link. When you click on the cross-file link the system will ask you if you want to open the corresponding notefile. At this point, if you redisplay the card, by selecting the **Redisplay** option on the window menu, the link will be shown with its appropriate card-type bitmap.

Missing Notefiles

If you rename or delete a notefile which has cross-file links pointing to it, the system will try to open the notefile by its original name. When it cannot find the notefile under its original name, it will ask you which notefile to look in. You cannot give a name with wild card characters like "*" or "?." You must give actual notefile names. The system will continue to prompt you for new notefiles to search until you give up.

[This page intentionally left blank]

9. CARDS AND BANNERS

Cards are the basic units of information in the NoteCards system. This information can be a mixture of text and graphics. The cards can contain as much or as little information as you deem necessary. You can link each card to any number of other cards to create an information network which you can randomly browse or systematically read. You access cards through the notefile Banner.

This chapter explains:

- What you can do from the notefile Banner.

- How to access old cards.

- How to create new cards.

- What card types there are.

- What the card menu does.

The Notefile Banner

Each notefile which you have opened is represented on the screen by a notefile Banner. The notefile name appears across the top of the Banner in the region called the title bar. The semi-colon after the name is followed by the notefile version number.

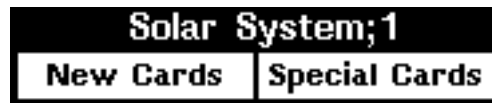


Figure 9-1. The Banner for the Solar System notefile.

The Banner has three active regions, the title bar, the **New Cards** option and the **Special Cards** option.

The Banner Title Bar

From the title bar you can access three notefile menus.

Notefile Ops Menus

There are two "Notefile Ops" Menus. One displays only those options which work on open notefiles. The other displays only those options which work on closed notefiles.

Holding the left mouse button down in the Banner title bar when the notefile is *closed* brings up the following "Notefile Ops" menu. Note that the **New Cards** and **Special Cards** options are grayed over when the notefile is closed.

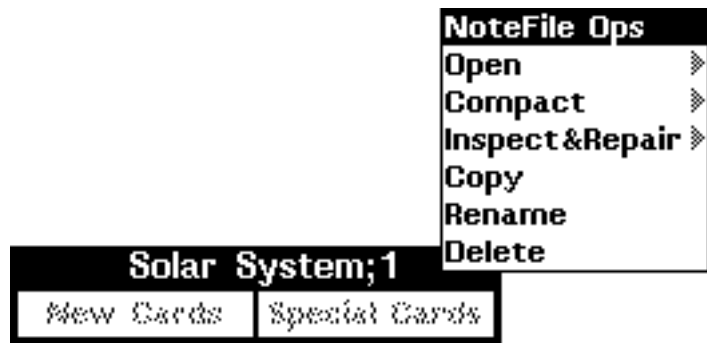


Figure 9-2. "Notefile Ops" menu from the Banner of a closed notefile.

Holding the left mouse button down in the Banner title bar when the notefile is *open* brings up the following "Notefile Ops" menu.

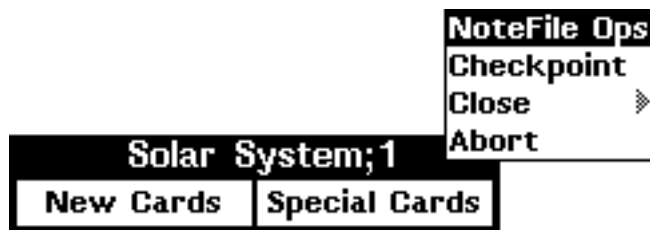


Figure 9-3. "Notefile Ops" menu from the Banner of an open notefile.

These two sets of options are a subset of those found on the "Notefile Ops" menu accessed by holding the left mouse button down on the MenuBox Icon's **Notefile** option.

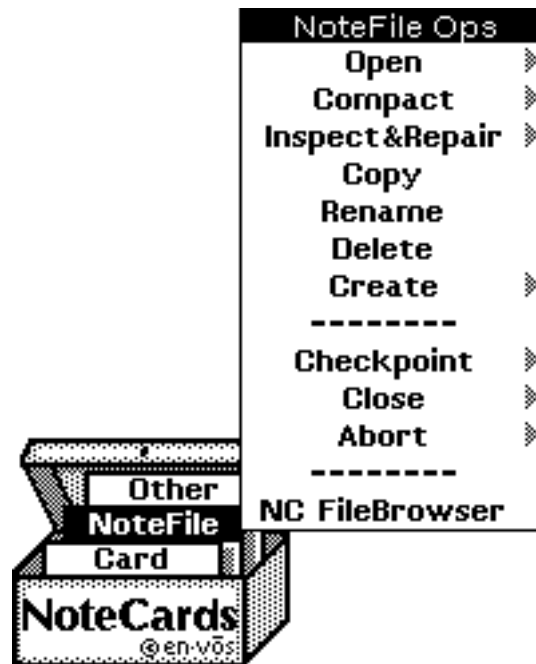


Figure 9-4. The MenuBox Icon's "Notefile Ops" menu.

All of these commands are discussed in Chapter 12, The MenuBox Icon.

Middle-Mouse-Button Title-Bar Menu

Holding the middle mouse button down in a notefile Banner's title bar brings up the following menu.



Figure 9-5. Middle-button menu in notefile Banner title bar.

Full File Name

Selecting the **Full File Name** option will bring up a window containing the full name of a file, including its directory path.

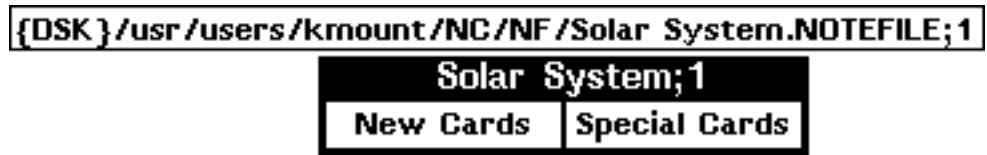


Figure 9-6. The full file name of the Solar System notefile.

Press any mouse button to continue. The window will close and you will be allowed to continue your work. This option is useful when you have two files with the same name, but different directories, on the screen and you need to be able to differentiate their Banners.

File Capacity

When the notefile is open, selecting the **File Capacity** option will attach a window to the top of the notefile Banner which will display the percentage and ratio of used cards in the notefile.



Figure 9-7. Notefile Banner with file capacity window open.

To close the file capacity window, hold the right button down in the file capacity window, not in the Banner title bar, and select the **Close** option off the window menu.

When 90% of the cards in a notefile have been used, the file capacity window will automatically open and warn you that the file is almost full and that it needs to be compacted. You must close a notefile before you can compact it. See Chapter 12, The MenuBox Icon, for instructions on compacting notefiles.

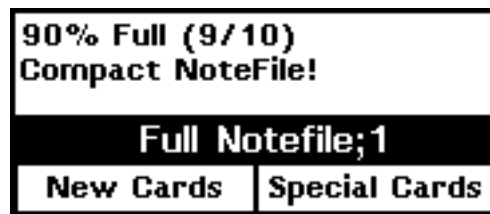


Figure 9-8. Notefile Banner with file capacity window warning to compact notefile.

If you try to create a new card when the notefile is 100% full, the system will display a message that the notefile is full and you will be asked if it is ok to checkpoint the notefile and make room for some number of new cards.

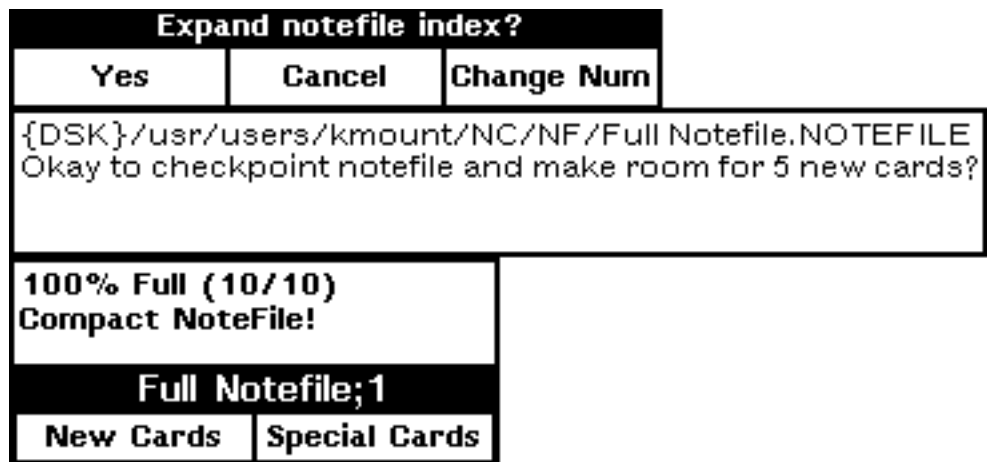


Figure 9-9. "Expand notefile index?" menu and prompt window.

If you select **Yes**, the system will save all the open cards, close the notefile, expand the notefile index by the amount you indicated, reopen the notefile, and create the card you requested. Note that the notefile is not compacted in this operation.

Cancel stops the operation. **Change Num** allows you to change the number of cards to expand the notefile by, after which you can choose the **Yes** option from the "Expand notefile index?" menu and proceed to save and expand the notefile.

Special Cards

Old notecards are accessed from the notefile Banner under the option **Special Cards**, (see Figure 9-1). Clicking the left mouse button in the **Special Cards** option brings up the "Table of Contents" FileBox card for the named notefile.

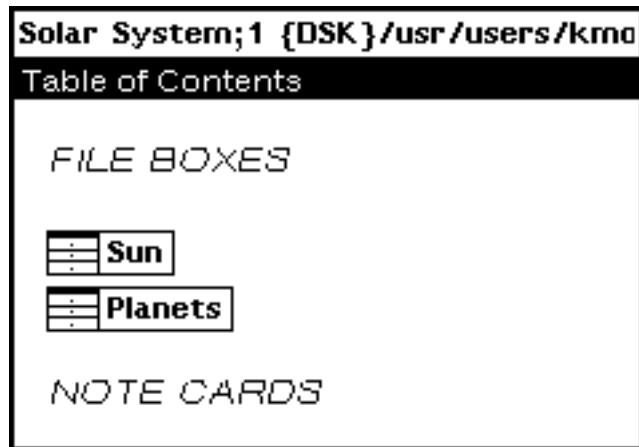


Figure 9-10. "Table of Contents" FileBox card for the "Solar System" notefile.

Holding down the middle mouse button in the **Special Cards** option brings up the "Special Cards" menu.



Figure 9-11. The "Special Cards" menu.

This menu allows you to access all of the top-level FileBoxes. Ultimately, all accessible cards are linked to one of these three FileBoxes.

Table of Contents FileBox

This is the top-level FileBox of each notefile. It is intended for storage of links to FileBoxes and other cards at the highest level of the information hierarchy.

To Be Filed FileBox

A temporary FileBox for cards that are not filed in any FileBoxes. If you close a card without specifying its FileBox, or if you close a notefile, without having specified the FileBoxes for every card, the system will place the cards without assigned FileBoxes in the "To Be Filed" FileBox.

If you later designate a FileBox for an unfiled card using the **Designate FileBoxes** or **Title/FileBoxes** commands from the card menu, this does not remove the card from the "To Be Filed" FileBox. To remove a card from the "To Be Filed" FileBox either delete its link or use the **Unfile From FileBoxes** suboption on the card menu of the newly filed card.



Figure 9-12. A card menu showing the **Unfile from FileBoxes** suboption selected.

As a general rule, it is a good idea to check the "To Be Filed" FileBox at the end of every session to verify that you have properly filed every card where you want it. You might also keep a copy of this FileBox on the screen so that you can tell at a glance if any cards have been filed there.

Orphans FileBox

A FileBox for cards whose last link from another FileBox has been removed.

Removing a link icon from the "Orphans" FileBox severs the card from the FileBox hierarchy but does not delete it from the notefile. Cards in this state are lost cards. The only means of retrieving a lost card is with a search card.

As a general rule, it is a good idea to check the "Orphans" FileBox at the end of every session to verify that you have properly filed every card where you want it. You might also keep a copy of this FileBox on the screen so that you can tell at a glance if any cards have been filed there.

New Cards

New cards are created from the notefile Banner under the option **NewCards**, (see Figure 9-1).

Clicking the left mouse button on the **NewCards** option immediately creates a specific type of card. The type of card created is determined by the value of the **Default Card Type** System Parameter. See Chapter 12, The MenuBox Icon, which describes how to set this parameter. The default value is "Text."

Pressing the middle mouse button over the **NewCards** option brings up the "Card Types" menu.

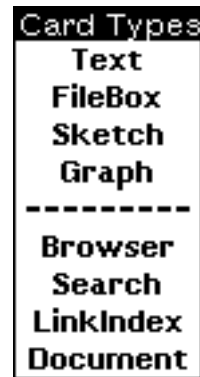


Figure 9-13. The "Card Types" menu.

Selecting one of these options creates a card of that type.

Figure 9-14 graphically represents the notecard types and their range of use. Text, Graph, and Sketch cards help you to collect and express your ideas. FileBoxes allow you to express the relationships between your ideas and organize them into coherent structures. Browser cards provide a broad range of functionality, and are a means of representing as well as retrieving information. LinkIndex cards build sorted lists of cards. Search cards perform searches on card titles. And, Document cards create a linear document from your linked cards.

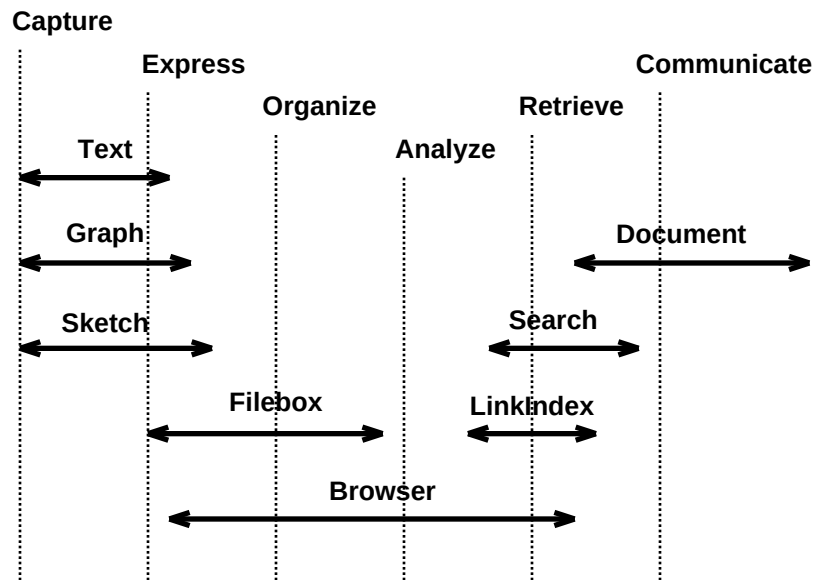


Figure 9-14. Card types and their range of use.

User Cards and System Cards

Cards can be broadly divided into two categories. User cards are those cards for which you create the contents. System cards are cards where the contents are built by the system for you. Chapter 10 discusses user cards; Chapter 11, system cards.

In Figure 9-13, the top four card types (Text, FileBox, Sketch, and Graph) are user notecards. The bottom four card types (Browser, Search, LinkIndex, and Document) are system notecards.

Text-, Sketch-, and Graph-Based Cards

All cards are based on one of three editors, the text editor TEdit, the Sketch editor Sketch, or the Graph editor Grapher.

Text, FileBox, Search, LinkIndex, and Document cards are all based on the text editor TEdit. Hence they all behave in essentially the same way and are referred to as text-based cards.

The Sketch card is the only card based on the Sketch editor. This card is referred to as a sketch-based card.

Graph and Browser cards are both based on the graph editor Grapher. These cards behave in basically the same way and are referred to as graph-based cards.

The Card Menu

All cards have a left-button title-bar menu called the card menu. This menu is the same for all cards except FileBox cards. Each card also has a middle-button title-bar menu which is frequently card-specific. These menus are discussed under their respective card types in Chapter 10, User Cards, and Chapter 11, System Cards.

The Standard Card Menu



Figure 9-15. The note card menu

This menu is found on all cards except the FileBox card which has a slightly different version.

Edit Property List

The property list editor allows you to associate property-value pairs with cards. For example, you might want to attach a Certainty Value property (and value) to every card. The value of this property

would indicate the degree to which you believe the information contained on the card is true.

You bring up the "Edit Property List" window by selecting the **Edit property List** option off the card menu.



Figure 9-16. "Edit Property List Window"

You can only change the property list of a card interactively, from the menu shown in Figure 9-17. Holding down the left or middle mouse button in the "Edit Property List" window's title bar brings up the following edit menu for property lists. Select commands from this menu before releasing the button.

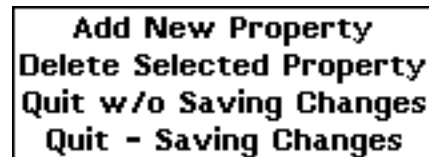


Figure 9-17. Menu from "Edit Property List" window title bar.

Add New Property

Adds user-defined properties to the property list of this card. Type in the property name and the value when prompted. Next, select one of the properties already in the list. The new property will be inserted in front of the selected property. Properties are displayed in bold type and values are displayed between brackets in regular type. You can abort the add process by hitting the Stop key.

In Figure 9-16, **Source** and **Certainty Factor** are the properties and "John Smith" and ".9" are the values. You can edit the values directly with the mouse and keyboard as you would any text string.

Delete Selected Property

Deletes a property from the property list of this card. After choosing this option click on the property to be deleted with the left mouse button. Properties are displayed in bold type. Once you have started the delete process there is no way to abort it, but you can hit the Undo key to undo the deletion.

Quit w/o Saving Changes

Closes the display without saving any of the current changes made using **Add New Property** or **Delete Selected Property**.

Quit - Saving Changes

Closes the display saving all current changes made using **Add New Property** or **Delete Selected Property**.

Show Links

Displays, in the "List of Links" window above the card, a list of all links to and from other cards. Links are represented by link icons. Selecting an icon in this window with the left mouse button displays the card referenced by that link icon.

To close this display, place the cursor in the title bar of the "List of Links" window, depress the left button, and select **Quit** from the single item menu.

For more on the **Show Links** option, see the section "Viewing Local and Global Links" in Chapter 8, Links.

Show Info

Brings up the "Card Attributes" window. This window displays the card's type, the dates each of the card parts was last changed, and a list of dates chronicling when the card was updated.

Card attributes	
Type	Text
ItemDate	" 5-Jan-89 18:00:19"
TitleDate	" 4-Jan-89 09:55:45"
LinksDate	" 6-Jan-89 18:04:26"
PropsDate	" 5-Jan-89 18:00:20"
Updates	((MOUNTFORD,ENVOS " 5-

Figure 9-18. The "Card attributes" window.

The **Show Info** option also automatically brings up the notefile indicator on the card.

Indicate NoteFile

This pull-across subitem on the **Show Info** option brings up the notefile indicator without bringing up the "Card Attributes" window.

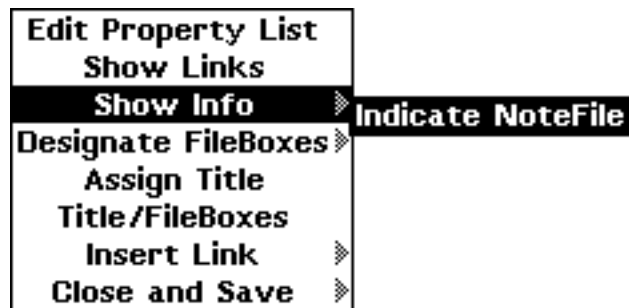


Figure 9-19. The **Indicate Notefile** submenu off **Show Info**.

Solar System;1 {DSK}/usr/users/kmount/NC	
Table of Contents	
FILE BOXES	
	Sun
	Planets
NOTE CARDS	

Figure 9-20 A card with its notefile indicator window open on the top of the card.

To close the "Card Attributes" window, just choose the **Close** item on the standard window title bar menu of the "Card Attributes" window. Closing the "Card Attributes" window does not close the notefile indicator window, which must be closed separately using its standard window title bar menu.

Designate FileBoxes

Pops up a prompt window above the card asking the user to file the card in one or more FileBox cards.

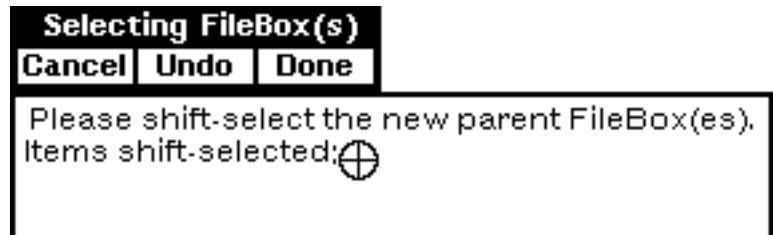


Figure 9-21. The "Selecting FileBox(es)" prompt window.

Associated with the prompt window is a three-item menu. Select **Done** from this menu or type a carriage return after shift-selecting the FileBoxes to file the card in. Select **Cancel** to abort the filing operation. Choose **Undo** to remove the last selected FileBox from the list of new parent FileBoxes.



Figure 9-22. A card menu showing the **Unfile from FileBoxes** suboption selected.

Unfile from FileBoxes

Unfiles the card from the FileBoxes you select. You unfile the card by shift-selecting the FileBox card IDs from their title bars. The system tells you when you have made an invalid selection. The prompt window menu options are the same as those for **Designate FileBoxes**, shown in Figure 9-21.

Assign Title

Assign Title allows you to assign a title to a card or edit an existing title.

You can reposition the cursor in the edit string with the mouse. You can also use the mouse to delete pieces of the string by using the left mouse button to position the cursor at the beginning of the piece you want to delete and sweeping out the portion to delete while holding down the right mouse button. When you release the right mouse button the selected text will be deleted.

To cancel the delete operation, still holding the right mouse button down, move the mouse cursor outside the prompt window and release the right mouse button.

You can undo the delete operation by hitting the Undo key.

Title/FileBoxes

Selecting **Title/FileBoxes** is identical to selecting **Assign Title** and **Designate FileBoxes** in sequence.

Insert Link

Insert Link inserts a user-specified link to another card inside the body of the current card. The link is represented by a link icon.

Before execution of this command, select the point, in the body of the card, where you want the link icon to appear.

When you select this menu option, a menu pops up displaying a list of link types currently available in the notefile. Link types are notefile specific. Specify the type of link by selecting one from this list or select **--New Link Type--** to create a new type. This new type of link is added to the notefile and becomes a choice in its list of link types. It is not possible to assign a system-reserved type of link to a user-specified link. Select ****CANCEL**** from the menu to abort the **Insert Link** command or just click outside the menu.

After a type has been designated, a prompt window and menu pops up above the card asking the user to choose the destination card for the link. The user has the option of selecting an existing card or of creating a new card as the destination card by selecting **New Card** from the menu with the left mouse button. If this option is chosen, a menu of card types pops up from which the user selects the desired type of card. Again, selecting **Cancel** from the "Selecting Note Card" menu aborts this command.

At this point, the link icon will be inserted at the flashing caret in the body of the source card.

For more on **Insert Link** and its submenu options **Insert Links**, **Add Global Link**, and **Add Global Links**, see Chapter 8, Links.

Close and Save

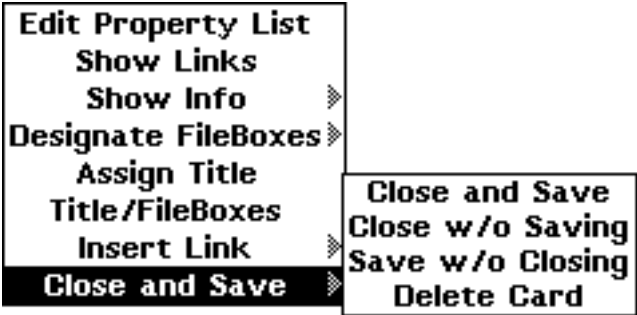


Figure 9-23. The submenu off the **Close and Save** option

Close and Save

Saves the card in the notefile before closing the card. If the **Force Titles** and **Force Filing** system parameters are set to "Yes," NoteCards will request that you provide a title and a FileBox for the card before closing it. If you do not wish to provide this information,

simply type a carriage return to both requests. In this case, the card will be titled "Untitled," and be filed in the "To Be Filed" FileBox.

Close w/o Saving

Closes the card without saving it, in the notefile. Any changes you have made to the card since the last save are lost. This option is useful if the card contents are mistakenly lost or scrambled while you are editing. This command may ask for confirmation. Type a carriage return or click on the **Yes** option to confirm. Type "n" or "N" and a carriage return or click on the **No** option to cancel.

Save w/o Closing

Saves, to the notefile, all changes you have made to the card without closing the card. Saving updates the card in the notefile but does not update the notefile index. If there is a system crash you will have to perform an **Inspect & Repair** operation to recover the card contents.

Delete Card

Permanently deletes the card from the notefile and all its links to and from other cards. Because this deletion is irreversible, the user is asked to confirm before the delete command is executed. Type a carriage return or click on the **Yes** option to confirm. Type "n" or "N" and a carriage return or click on the **No** option to cancel. Note that a deleted card cannot be retrieved with a search card as a lost card can.

The FileBox Card Menu

The FileBox card menu differs in only two items, **Add Global Link**, and **Put Cards Here**.

Add Global Link

The **Add Global Link** and **Add Global Links** options are the same global link options found on the submenu of the **Insert Link** option. **Insert Link** appears in this position on this menu for all other cards. These two options are discussed extensively in Chapter 8, Links.



Figure 9-24. The FileBox card menu with the **Add Global Link** submenu.

Put Cards Here

The **Put Cards Here** menu option is a specialization of the **Insert Links** option for FileBox cards. It allows you to file more than one card in a FileBox in a single operation. Selecting this option brings up the "Selecting cards to file" prompt window into which you can shift select the card IDs from the title bars of all the cards you want to file in this FileBox card.

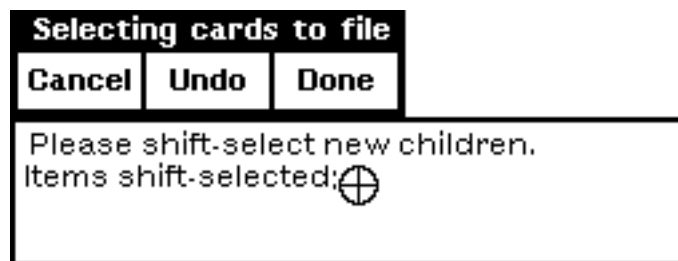


Figure 9-25. The "Selecting cards to file" prompt window.

[This page intentionally left blank]

Cards can be broadly divided into two categories. User cards are those cards for which you create the contents. System cards are cards where the contents are built by the system for you. This chapter discusses user cards.

This chapter explains :

How to use Text cards.

How to use FileBox cards.

How to use Sketch cards.

How to use Graph cards and the graph editor.

How to use the bit map editor.

Text Cards

The Text card is based on TEdit, a versatile editor and text formatter. This entire document was produced using TEdit. For a detailed discussion of how this editor works, see *A User's Guide to TEdit*. Text cards allow you to include sketches, graphs, and bit maps. To learn how to manipulate sketches, see *A User's Guide to Sketch*. For graphs and bit maps see the sections below on the Graph card and the bit map editor.

The Text-Card Menu

The text-card menu is the same as the TEdit menu with one additional option separated from the others by a dashed line.

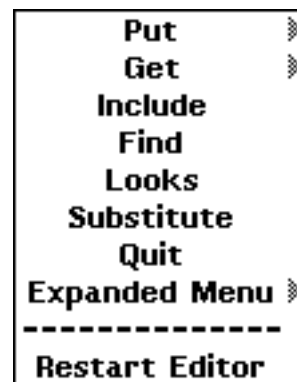


Figure 10-1. The text-card menu.

You use the **Restart Editor** command when the contents of the Text card are incorrectly displayed on the screen.

For all the other menu items, see *A User's Guide to TEdit*.

FileBox Cards

The FileBox card, like the Text card, is based on TEdit. For a detailed discussion of how this editor works, see *A User's Guide to TEdit*. Text-based cards allow you to include sketches, graphs, and bit maps. To learn how to manipulate sketches, see *A User's Guide to Sketch*. For graphs and bit maps see the sections below on the Graph card and the bit map editor.

A FileBox is a card that contains links to other cards including other FileBox cards. All cards, including FileBox cards, can be filed in one or more FileBoxes. Every card, including FileBox cards, except the top level Special FileBox Cards, is contained in at least one other FileBox. Whereas other cards may be linked together to form an arbitrary network, the set of FileBoxes forms a strict hierarchy. This is to say that no child FileBox is allowed to have its parent FileBox as a child. In short, no circular linkages.

FileBoxes are meant to hold all cards relating to some given topic. A FileBox typically contains both links to subFileBoxes, which contain any cards relevant to the subtopics of the main topic, and links to other card types, which contain information relevant to the main topic. For example, the screen image below shows a FileBox containing both FileBoxes and other note cards.

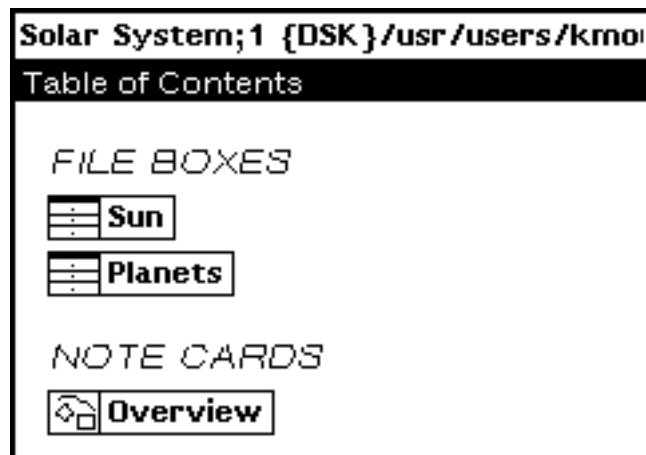


Figure 10-2. The Solar System FileBox containing subFileBoxes for its subtopics and a Sketch card dealing with the main topic, the Solar System.

The FileBox structure provides a way of keeping track of sets of cards on a common topic. In contrast, the links between individual cards allow you to represent the interconnections between various ideas or pieces of information, independent of any categorization into topic areas.

The markers FILE BOXES and NOTE CARDS help differentiate what kinds of cards are filed in the FileBox. In addition, since FileBoxes are text-based cards, anything you can do with a text card, you can also do with a FileBox. This means that you can, for example, insert your own labels or short lines of commentary to break up the links into subgroups.

Suggested FileBoxes and Note Cards

You may find it helpful to create the following types of general FileBoxes

Bibliography

A FileBox for the collection of sources used in the notefile.

Index

A FileBox listing keywords from the notefile, may be helpful when using Search cards.

Read Me

A note card in the top level FileBox giving global information about the notefile for first time browsers.

Active Cards

A FileBox kept at the top level of the FileBox hierarchy containing FileBoxes and note cards that represent work in progress and are thus frequently accessed. A Sketch card containing links to these FileBoxes and note cards is another method of organizing active cards, using spatial cues as a way of representing structure.

The FileBox-Card Menu

This menu is the same as the Text-card menu. See the section immediately above on Text cards as well as *A User's Guide to TEdit*.

System Parameters Affecting FileBoxes

FileBoxes have two system parameters associated with them, **Markers In FileBoxes** and **Alphabetized FileBox Children**. For a complete discussion of these parameters see Chapter 13, System Parameters.

Sketch Cards

The Sketch card is based on Sketch, a sophisticated graphics package. For a detailed discussion of how Sketch works, see *A User's Guide to Sketch*. Sketch cards allow you to include graphs, and bit maps. To learn how to manipulate graphs and bit maps see the sections below on the Graph card and the bit map editor.

The Sketch-Card Menu

This menu is the same as the Sketch editor menu. For a detailed discussion of this menu's functionality, see *A User's Guide to Sketch*.



Figure 10-3. The Sketch card menu.

System Parameters Affecting Sketch Cards

Sketch cards have one system parameter associated with them, **Attach Sketch Menu**. For a complete discussion of this parameter see Chapter 13, System Parameters.

Graph Cards

The Graph card is designed to allow you to construct a layout of user-defined words or phrases, called nodes, which may be connected together with lines to indicate some structure. Each node may be easily moved about the card without losing its connections.

The Graph-Card Menu

The Graph card, like the Browser card, is based on the graph editor Grapher. To make the graph-card menu and terminology more consistent with the browser-card menu and terminology the

Graph card presents a slightly different menu to you than Grapher does. However, since the functionalities are virtually identical, both the graph-card and Grapher menus are discussed in parallel below. The graph-card menu-option titles are left justified while the Grapher menu-option titles are right justified.

You display the graph-card menu by depressing the right mouse button in the body of the card. Select the desired command before releasing the button.

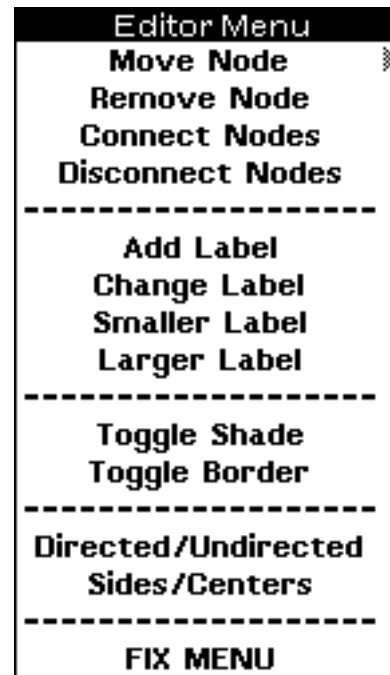


Figure 10-4. The graph-card menu.

The Grapher Menu

You will probably encounter the Grapher menu only if you save your document cards to TEdit files and edit a graph-card graph from within the TEdit document. You can safely skip over this section and still understand the Graph card.

To edit a graph from a TEdit document you must first select the **Edit graph** option from the one-item menu which appears when you hold any mouse button down in the graph region.

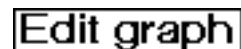


Figure 10-5. The "Edit graph" menu.

When you select this option, Grapher opens a window containing the graph. Hold the middle mouse button down in this window to make the menu shown in Figure 10-6 appear. Use the left mouse button to move nodes. When you are done editing the graph, select the **STOP** option.

While you are editing a graph, Grapher captures the type-in process and does not allow you to do anything other than mouse operations. To free the type-in process, select the **STOP** option.

There is no simple way to abort out of the Grapher editor and throw away all the changes you have made to the graph. For this reason, if you are going to edit the graph extensively, we recommend that you use shift-select to copy the original graph in place and call the Grapher editor on the copy. In a worst case scenario, you can try hitting the STOP key or typing CONTROL-E.

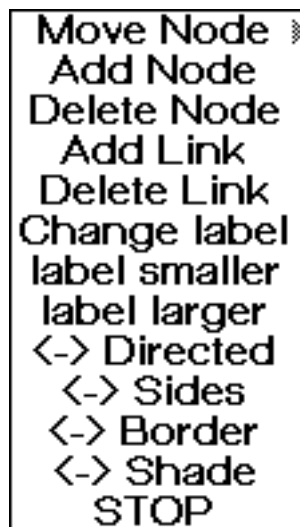


Figure 10-6. The Grapher menu.

Prompts for information or confirmation are given in the card prompt window for Graph cards and appear in the system prompt window for Grapher. General information is printed to the system prompt window for both Graph cards and Grapher.

The Graph-Card and Grapher Menu Options

Move Node

Move Node

Moves a node and connections to a new position. After selecting this option, point to the node you want to move, press and hold the left mouse button, move the node to its new position, and release the mouse button.

Move Node has three options on a submenu. These same three options appear on the Grapher **Move Node** submenu.



Figure 10-7. The **Move Node** submenu.

Move Single Node

Functions exactly as **Move Node** does.

Move Node & SubTree

Moves a selected node and all subnodes which it is connected to. This operation does not move any super nodes of the selected node. That is, nodes which are connected to the selected node as opposed to nodes which the selected node is connected to.

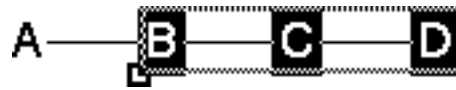


Figure 10-8. A node and subtree selected to be moved.

Figure 10-8 shows what happens, when you select **Move Node & SubTree** and then hold the left mouse button down on node B. Nodes B, C, and D are selected to be moved, but not node A. Grapher keeps track of the nodes where each connection starts and ends, and in this case the connections run from A to B to C to D. So node A is not in the subtree of node B and hence is not moved. See the **Directed/Undirected** option below for more information.

Move Region

Allows you to sketch out a region of the graph which you want to move. **Move Region** does not pay attention to the graph hierarchy, it only pays attention to the area you sweep out in the Graph card.

Remove Node

Delete Node

Removes a node from the graph. Select the node to be deleted with the left mouse button. The card prompt window will prompt you for confirmation.

Connect Nodes

Add Link

Draws a connection between two nodes. Select the "from" node and then the "to" node with the left mouse button when prompted. If a second overlapping connection is made running in the opposite direction between the same two nodes, the lines representing the connections, between those two nodes, will not be visible. You can make them visible by choosing the directed display option.



Figure 10-9. A graph, with connections from A to B to C and a third connection from C to B, displayed using the **Undirected** option.

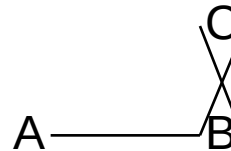


Figure 10-10. A graph, with connections from A to B to C and a third connection from C to B, displayed using the **Directed** option.

Disconnect Nodes

Delete Link

Removes a connection from between two nodes. Select the "from" node and then the "to" node with the left mouse button when prompted.

Add Label**Add Node**

Pops up a window prompting you to type in a label name followed by a carriage return. The label will appear next to the cursor within the graph card. Position the new label by moving the cursor to where you want the label to appear. Plant the label by clicking any mouse button. This operation can be cancelled by typing a carriage return before typing any other characters to the prompt window.

Change Label**Change Label**

Allows you to change a label. **Change Label** first waits for you to select, with the left mouse button, the label you want to change. It then pops open a window prompting you to type in a new label name followed by a carriage return. The new label immediately replaces the old label, preserving font, position, and connections. You can cancel this operation by clicking outside a node, or by typing a carriage return before typing any other characters to the prompt window.

Smaller Label**label smaller**

Decreases the font size of the selected node. Repeat this command as many times as necessary to achieve the font size you want.

Larger Label**label larger**

Increases the font size of the selected node. Repeat this command as many times as necessary to achieve the font size you want.

Toggle Shade**<-> Shade**

Inverts the shade around the selected node. For example, a black label on a white background becomes a white label on a black rectangular background. Select the node to be inverted with the left mouse button. To change the shade back, re-apply this option.

Toggle Border**<-> Border**

Draws a rectangular border around the selected node. Select the node to have a border drawn around it with the left mouse button. To remove a border, re-apply this option.

Directed/Undirected**<-> Directed**

A graph is stored as a directed lattice. Connections always run from one node to some other node.

The **Directed** option makes the flow of the connections explicit in the presentation of the graph. When you select the **Directed** option, connections prefer to run from the left side of the parent node to the right side of the child node when you have **Sides** selected. When you have the **Centers** option selected, connections prefer to run from the bottom center of the parent node to the top center of the child node.

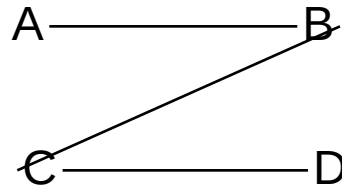


Figure 10-11. A **Directed** graph which explicitly shows the flow of connections from A to B to C to D, with the **Sides** option selected.

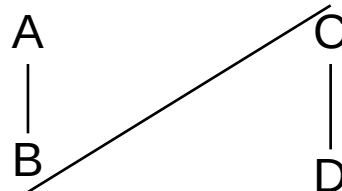


Figure 10-12. A **Directed** graph which explicitly shows the flow of connections from A to B to C to D, with the **Centers** option selected.

The **Undirected** option draws the graph without regard to the flow of connections. The lines are drawn starting and ending on the sides of the nodes closest to each other.

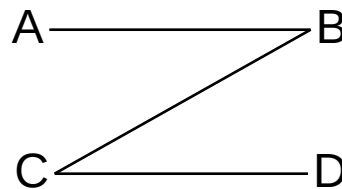


Figure 10-13. An **Undirected** version of Figure 10-11.

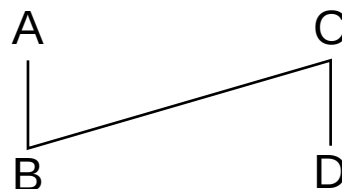


Figure 10-14. An **Undirected** version of Figure 10-12.

Sides/Centers

<-> Sides

The sides mode predisposes the graph editor to make the left and right sides of the nodes the connection points for lines.

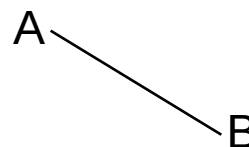


Figure 10-15. A graph drawn favoring sides.

The centers mode predisposes the graph editor to make the top and bottom centers of the nodes the connection points for lines.

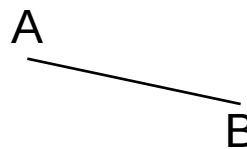


Figure 10-16. A graph drawn favoring centers.

FIX MENU

Attaches the graph-card menu to the right side of the Graph card.

STOP

Only appears on the Grapher menu, not on the Graph card menu.
STOP exits the Grapher editor saving all your changes.

The Bit Map Editor

The bit map editor allows you to manipulate bit maps that have been inserted in Text or Sketch cards as well as TEdit and Sketch files. It is automatically invoked when the bit map area is selected.

Inserting Bit Maps into Cards

The method for inserting bit maps into text-based cards differs slightly from that for inserting them into sketch-based cards. Each procedure is discussed below.

You cannot insert bit maps into graph-based cards.

Text-based Cards

Inserting a bit map into a text-based card involves several steps. First, position the caret cursor where you want the bit map to appear in the destination card, or TEdit window, by clicking at that position with the left mouse button. Second, depressing the Copy key or either of the Shift keys, hold the right mouse button down somewhere in the background and select the **Snap** option from the single item menu which will appear.



Figure 10-17. The single-item "Snap" menu.

At this point the mouse cursor changes to look like this,  This is the prompt asking you to sweep out an area of the screen to be made into a bit map. Third, press and hold the left mouse button while you sweep out a region of the screen. When you release the left mouse button, the bit map will be transferred to the designated card or edit window.

If you need to adjust the area you are sweeping out, do the following. Hold down the right mouse button, in addition to the left mouse button, to bring up the forceps prompt,  This prompt

allows you to change corners so that you can adjust the size of your bit map in all directions.

Sketch-based Cards

The procedure for inserting bit maps into sketch-based cards differs in only one respect from that for text-based cards. The sketch-based card must be the active card, which is to say you must click in it so that it has the type-in process to mark it as the destination for the bit map. However, when you do this, you are not indicating the insertion point for the bit map. In sketch-based cards, positioning the bit map is done last. After you have swept out a region of the screen to include as a bit map in the sketch, move the mouse cursor back into the Sketch card. When you enter the Sketch card, the snapped bit map will appear attached to your mouse cursor, and you can position it by clicking the left mouse button.

Bit Map Operations

Moving the mouse cursor into a bit map and holding down the left or middle mouse button brings up the "Operations on bitmaps" menu.



Figure 10-18. The "Operations on bitmaps" menu.

Change Scale

Changes the scale or size of the bit map. Giving a scale of 2 doubles the size of the bit map; a scale of .5 halves the size of the bit map. You achieve the best results shrinking or enlarging a bit map when you change the scale by evenly divisible amounts. For example, 4, 2, 1, .5, or .25.

Hand Edit

Invokes the bit map editor on the bit map. The bit map editor is described in detail below.

10. USER CARDS

Trim

Trims the white columns and rows from all four edges of the bitmap. This is a very useful operation to remove any extraneous white space from around the bit map. Position the image that you are taking a snap of on a white background to take the greatest advantage of this option.

Reflect Left-to-right

Flips the bitmap about its vertical centerline.

Reflect Top-to-bottom

Flips the bitmap about its horizontal centerline.

Reflect Diagonally

Flips the bitmap about its X=Y diagonal so that the resulting bit map is reversed and lying on its right side. The same effect can be achieved by performing a **Reflect Left-to-Right** followed by a **Rotate Right**.

Rotate Left

Rotates the bit map by 90 degrees in a counterclockwise direction so that the resulting bit map is lying on its left side.

Rotate Right

Rotates the bit map by 90 degrees in a clockwise direction so that the resulting bit map is lying on its right side.

Expand on Right

Adds white space to the right of the bit map. You specify the width of the white space in pixels using the number pad. Select **ok** when you are done.

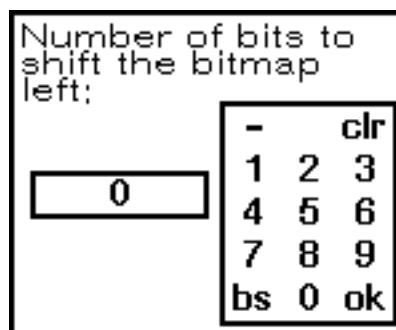


Figure 10-19. The number pad.

The number pad is used much like a simple calculator to enter numbers. The **ok** button returns the number to the system. **bs** deletes the last digit you entered. **clr** resets the input to "0." You abort the operation by setting the input value to "0" and selecting **ok**.

You enter a negative number by first entering the digits and then selecting the minus sign. Entering a negative number removes that many pixels from the right side of the bit map.

Expand on Left

Adds white space to the left of the bit map. You specify the width of the white space in pixels using the number pad. Select **ok** when you are done. See the **Expand on Right** option for more detail.

Expand on Bottom

Adds white space to the bottom of the bit map. You specify the width of the white space in pixels using the number pad. Select **ok** when you are done. See the **Expand on Right** option for more detail.

Expand on Top

Adds white space to the top of the bit map. You specify the width of the white space in pixels using the number pad. Select **ok** when you are done. See the **Expand on Right** option for more detail.

Switch Black & White

Inverts all of the pixels in the bit map; exchanges black for white and white for black.

Add Border

Adds a border to the bit map. The system prompts you for the width of the border using the number pad described above. It then prompts you for the texture of the border with the texture bit map editor.

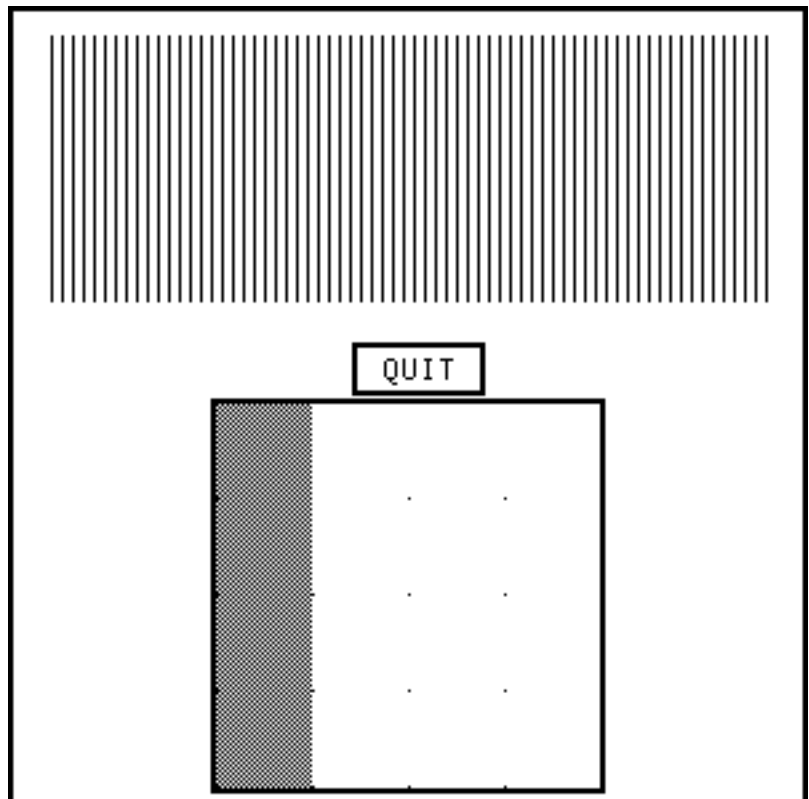


Figure 10-20 The texture bit map editor.

The area at the top of the window shows what the texture looks like in true screen scale and the bottom area contains a four-by-four

edit array. Clicking the left mouse button in the edit grid turns a pixel on; clicking the middle button turns a pixel off. Select the **Quit** option when the texture looks the way you want it to. The texture will then appear as the border around the bit map.

To abort this operation, select **clr** followed by **ok**, on the number pad.

There is no simple way to abort this operation once you have brought up the texture bit map editor. However, you can turn all the pixels off (set them to white space) and select **Quit**. Then selecting **Trim** from the "Operations on bitmaps" menu should return the bit map to its previous condition.

The Bitmap Editor

The editing window has three active areas, a grid edit area in the lower part of the window, a display area in the upper left part, and a gray bar in the upper right.

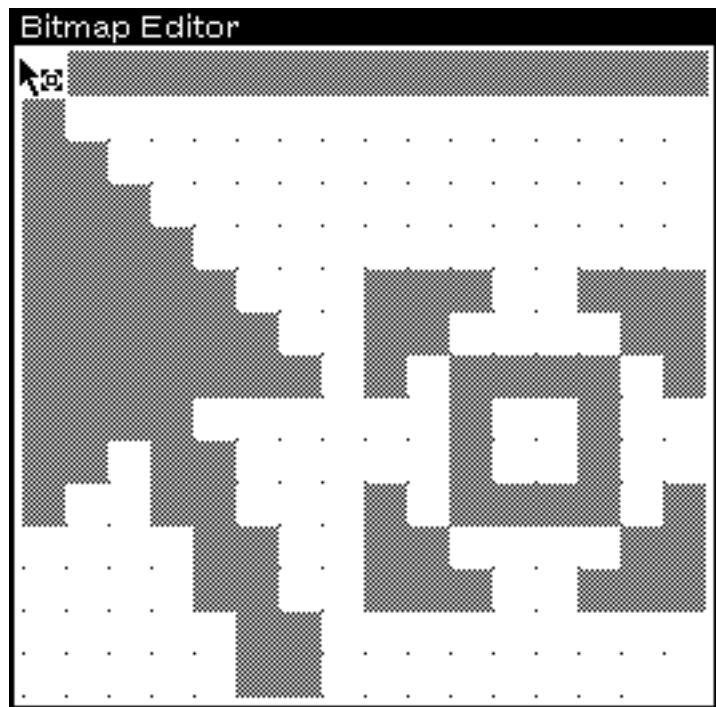


Figure 10-21. The "Bitmap Editor" display.

In the edit area, the left button adds points and the middle button erases points. The display area shows the actual size and form of the bit map. The gray bar provides access to the "Bitmap Editor" menu.

The right mouse button brings up the normal window menu in all areas of the window.

If the bit map is too large to fit in the edit area, you can change the portion which can be edited by scrolling up and down in the left margin, and left and right in the bottom margin. Pressing the middle mouse button while in the display area brings up a menu that allows you to make a global placement of the portion of the bit map which can be edited. If you want to see more of the bit map

you are editing, you can reshape the window to make it larger, or you can use the **GridSize** command, described below, to reduce the bit size in the edit area.

Whenever you press the left or middle mouse button down with the cursor inside the display area or the gray bar, the section of the bit map that is currently in the edit area is shown in reverse video. Pressing the left button while in the gray bar puts the lower left 16 x 16 bit section of the bit map into the mouse cursor for as long as the left button is held down.

Pressing the middle button while in the grey bar or in the title bar brings up the "Bitmap Editor" menu.

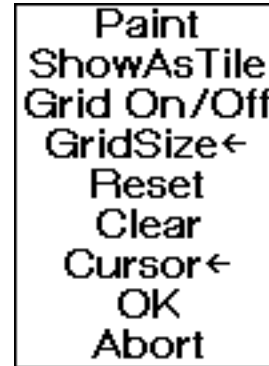


Figure 10-22. The "Bitmap Editor" menu.

Holding the middle button down over a command results in an explanatory message being printed in the system prompt window.

Paint

Puts the current bit map into a window and calls the paint command on the bit map. You use the left mouse button for drawing and the right for erasing. The paint command implements drawing with various brush sizes and shapes but only on an actual sized bit map. You set brush characteristics and exit paint by pressing the right mouse button and selecting the appropriate command from the paint command menu. When you exit, you will be asked whether or not the changes you made while in Paint mode should be placed in the current bit map. **Paint** is particularly useful for erasing or filling in large regions in bit maps. See the section "The Window Menu" in Chapter 7, The User Interface for a detailed discussion of all the paint menu options.



Figure 10-23. Paint command menu.

ShowAsTile

Tesselates the current bit map in the gray bar. This is useful for determining how a bit map will look if it were made the display background. The tiled display does not automatically change as

the bit map changes. To update it, use the ShowAsTile command again.

Grid On/Off

Turns the editing grid display on or off.

GridSize

Allows you to specify the size of the editing grid. When you select this option, a number menu appears, giving you a choice of several point sizes for the grid.

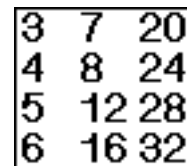


Figure 10-24. The number menu.

When you select a size, the editing portion of the bit map editor is redrawn. A smaller size allows you to edit more of the bit map without scrolling, while a larger size makes it easier for you to turn individual bits on and off. The original size is chosen heuristically. It is typically about 8. Clicking outside the number menu aborts this operation.

Reset

Sets all or part of the bit map to the contents it had when you originally called the bit map editor. When you select this option, a second menu appears giving you a choice between resetting the entire bit map or just the portion that is in the edit area.



Figure 10-25. The "RESET how much?" menu.

This second menu also acts as a confirmation, since clicking outside of this menu results in no action being taken. Note that if the entire bit map appears in the edit area the menu only has the **WholeBitmap** option.

Clear

Sets all or part of the bit map to white space. As with the Reset command, a second menu gives you a choice between clearing the entire bit map or just the portion that is in the edit area.



Figure 10-26. The "CLEAR how much?" menu.

Cursor

Sets the cursor to the contents of the lower left part of the bit map. This operation next prompts you to specify the new cursor's active pixel. You do this by clicking somewhere in the lower left 16 x 16 portion of the grid. Cursors created this way are typically very short lived. This option is intended for people extending the NoteCards environment. We recommend that non-programmers do not use this option.

OK

Copies the edited bit map image into the original bit map, exits the bit map editor, and closes its edit window. The image you modify using the editor is a copy of the original bit map. Unless you exit the bit map editor via **OK**, no changes are made to the original bit map.

Abort

Exits the bit map editor without making any changes to the original bit map. Contrast with **OK**.

[This page intentionally left blank]

12. THE MENUBOX ICON

The MenuBox Icon is the main interface to NoteCards. Through this icon you access all the system NoteCards menus.

This Chapter explains:

How to create notefiles.

How to perform basic operations on notefiles.

How to recover notefiles when they are damaged.

How to perform structure operations on notefiles.



Figure 12-1. The MenuBox Icon.

Notefile Options

It is through the Notefile-Operations menu that you interact with notefiles. The word "Operations" is usually abbreviated as "Ops" and, as a result, this menu is usually referred to as the "Notefile Ops" menu.

You access the "Notefile Ops" menu by holding the left mouse button down in the **NoteFile** option of the MenuBox Icon. The menu has three regions, separated by dashed lines. The top region contains operations you can only apply to closed notefiles. The middle region contains commands you can only apply to open notefiles. The bottom item, NC FileBrowser, does not share these limitations. It provides you with a notefile interface.

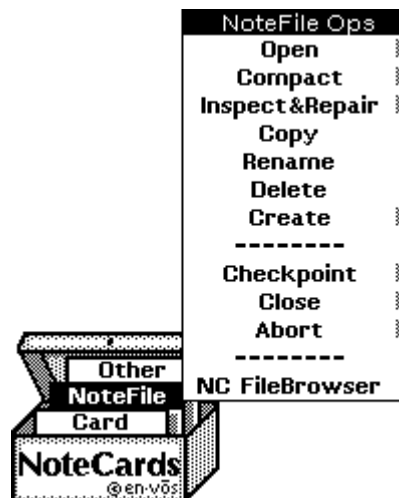


Figure 12-2. The MenuBox Icon's "notefile Ops" menu.

Open

Opens an existing notefile. If you have not previously opened any files, the system prompts you to type in a file name. The file name is not case sensitive. If you provide just a file name with no path, the system looks in the connected directory for the notefile. If NoteCards does not find the file there, it asks you if you want to create the file.

The system maintains a list of all notefiles you open. Subsequently, when you select **Open**, these files will be presented to you in menu format. If the file you want to open is not on the menu, select the **--Other Notefile--** option.

There are two submenu options for Open, **Open Read/Write** and **Open Read-only**.



Figure 12-3. The **Open** submenu.

Open Read/Write

Is the same as selecting **Open** from the top-level menu.

Open Read-only

Opens the file but does not allow you to make any changes to the file. When you open a file read-only, the file name appears in its Banner with a read-only prefix (RO:).

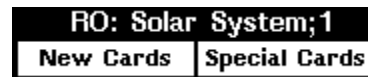


Figure 12-4. A notefile opened read-only with its Banner showing the read-only prefix "RO:"

If the notefile is damaged the following menu appears.

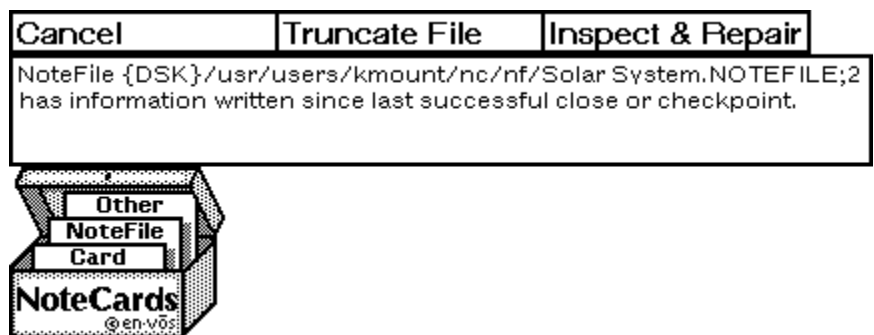


Figure 12-5. The menu which appears when you open a damaged notefile.

Cancel causes the open notefile operation to be aborted.

Truncate File deletes everything after the checkpoint pointer.

Inspect & Repair causes the notefile inspector to be called on your notefile. This is an easy, but slow, way to incorporate these post-checkpoint changes into the notefile. For more details, see the

Inspect & Repair and Appendix A, Notefile Concepts and Appendix B, The Notefile Inspector.

Compact

Compresses a notefile by deleting old information. NoteCards never actually overwrites any information in the data area of a notefile. When cards are revised, old information remains in the notefile. Compacting creates a new, compressed version of the notefile by copying into the new version only the latest information from the old version. To understand the structure of notefiles and compacting, see Appendix A, Notefile Concepts.

Compacting can take 5 to 20 minutes, depending on the size of the notefile.

There are two submenu options off **Compact**, **Compact To New File** and **Compact in Place**.



Figure 12-6. The **Compact** submenu.

Compact To New File

Is the same as selecting **Compact** from the main menu. NoteCards prompts you to type in a new file name. The default is the same file name with a higher version number.

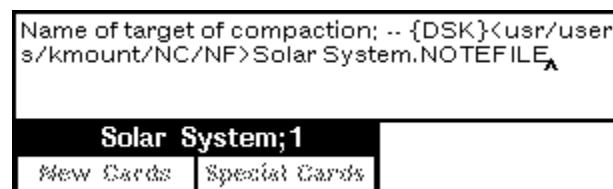


Figure 12-7. NoteCards prompting for the output file name for the compact operation.

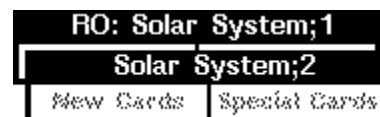


Figure 12-8. The notefile Banner after compacting to the same file name. The old version is "Solar System;1." The new version is "Solar System;2."

Compact In Place

Does not build a new copy of your notefile, rather the contents of the old file are rearranged in such a way that old versions of cards are written over. This is useful when your notefile is large and there isn't room to store another version. However, be aware that when you use this option you are not able to back up cards to previous versions by using the **Inspect & Repair** utility. We recommend

compacting to a target file and saving the uncompact version until you are sure that the older versions of cards are not needed.

Inspect & Repair

Reads the data area of your notefile, looking for good card parts, including outdated and deleted versions. It then allows you to delete or back up card parts to earlier versions through its menu interface. Only when the notefile is considered healthy are you allowed to perform the link rebuilding.

Inspect & Repair has two submenu options, **From Links** and **From Contents**.

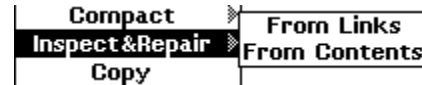


Figure 12-9. The **Inspect & Repair** submenu.

From Links

Rebuilds the notefile index using only the links part of the notefile.

From Contents

Rebuilds the notefile index using the links part of the notefile as well as the links stored in the bodies of individual note cards. **From Contents** takes longer to rebuild a notefile index.

For a detailed explanation of the **Inspect & Repair** utility, see Appendix B, The Notefile Inspector.

Copy

Allows you to copy a notefile.

Prompts you for a new name and path for the notefile to be copied and copies it. This is the same as doing a file copy.

Rename

Allows you to rename a notefile.

Prompts you for a new name and path for the notefile to be renamed and renames it. This is the same as executing a file rename.

Delete

Deletes a notefile.

NoteCards asks you to confirm the deletion by typing a carriage return for "No" or a **Y** and a carriage return for "Yes." If your response is "Yes," a second confirmation is required a second time to insure against unintentional deletion.

Create

Creates a new empty notefile. When you use **Create**, it is possible to assign an already existing name to the new notefile, thereby creating more than one version of the notefile.

Create has two submenu options, **Create without Open** and **Create and Open**.



Figure 12-10. The **Create** submenu.

Create without Open

Is the default action. It prompts you for a new notefile name, creates a new notefile, and place the new notefile's Banner on the screen. If you supply only a name and no path, the notefile is created in your connected directory.

Create and Open

Does all of the above, and opens the new notefile.

Checkpoint

Saves the contents of any active cards to the notefile without closing the cards or the notefile. The index array is written out to the file and the checkpoint pointer is reset to the end of the file.

A notefile consists of two parts, an index area and a data area. For each card, the index contains four pointers into the data area. There are pointers for the notecard's contents, title, property list, and list of links. When, for example, you change a card's title, NoteCards writes the new title at the end of the file, and the card's title pointer is changed to point to the new title. To increase access speed, index modifications are written in memory and are not written to the file till checkpoint, or close time. There is a checkpoint pointer that is updated to point to the end of file at checkpoint or close. New data, such as a new title, is still written to the file, but always at the end of the file. Thus if a crash occurs, due to a power failure or some other mishap, and later you reopen the notefile, NoteCards can notice the extra data beyond the checkpoint pointer. Each time a notefile is opened, the checkpoint pointer is compared with the end of file pointer. If they don't agree, then a message that something is wrong is printed out. At this point, you must choose one of three options from a menu: **Cancel**, **Truncate File**, or **Inspect & Repair**.

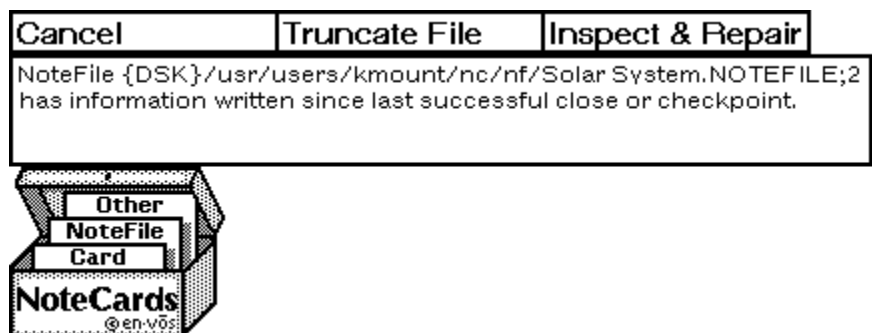


Figure 12-11. The menu which appears when you open a damaged notefile.

Cancel causes the open notefile operation to be aborted.

Truncate File deletes everything after the checkpoint pointer.

Inspect & Repair causes the notefile inspector to be called on your notefile. This is an easy, but slow, way to incorporate these post-checkpoint changes into the notefile. For more details, see Appendix A, Notefile Concepts and Appendix B, The Notefile Inspector.



Figure 12-12. The **Checkpoint** submenu.

Checkpoint and **Checkpoint Notefile** pop up a menu of known notefiles which you can checkpoint.

NoteFiles	
Solar System;2	{dsk}/usr/users/kmount/nc/nf/solar system,notefile;2
o Solar System;4	{dsk}/usr/users/kmount/nc/nf/solar system,notefile;4
o demo;1	{dsk}/usr/users/kmount/nc/nf/demo,notefile;1
test;1	{dsk}/usr/users/kmount/nc/nf/test,notefile;1

Figure 12-13. A menu of the list on known notefiles.

The grayed over notefiles are closed and cannot be checkpointed. Selecting one of the open notefiles causes that notefile to be checkpointed.

Checkpoint All checkpoints all open notefiles.

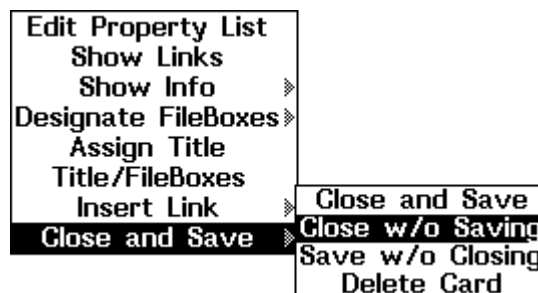


Figure 12-14. The card menu with the **Close w/o Saving** menu option selected.

In contrast to **Checkpoint**, when you choose **Close and Save** or **Save w/o Closing** from the card menu, the card's contents are written to the notefile but the notefile index is not written to the notefile. If you are anticipating a crash, you should checkpoint you notefile frequently.

Close

Allows you to close a notefile.

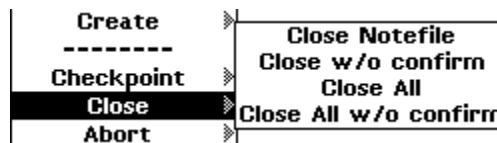


Figure 12-15. The **Close** submenu.

Close and Close Notefile

Pop up a menu of known notefiles like that shown in Figure 12-13. Selecting one of these notefiles closes that notefile. The grayed over notefiles are already closed. Selecting one of these has no effect.

Close w/o confirm

Closes a notefile without asking you about open cards on the screen. Open cards are closed and saved to either their designated FileBox or the "To Be Filed" FileBox.

Close All

Closes all open notefiles.

Close All w/o confirm

Closes all open notefiles without asking you about open cards on the screen. Open cards are closed and saved to either their designated FileBox or the "To Be Filed" FileBox.

Abort

Truncates and closes a notefile.

If you discover that you have made changes to a notefile that you don't want to keep, choose this option to truncate your file. This should only be done if you feel that the post-checkpoint work is not worth saving.



Figure 12-16. The **Abort** submenu.

Abort and Abort Notefile

Pop up a menu of known notefiles like that shown in Figure 12-13. Selecting one of these notefiles aborts that notefile. The grayed over notefiles are already closed. Selecting one of these has no effect.

Abort All

Aborts all open notefiles.

NC FileBrowser

Opens up a FileBrowser specialized for NoteCards.

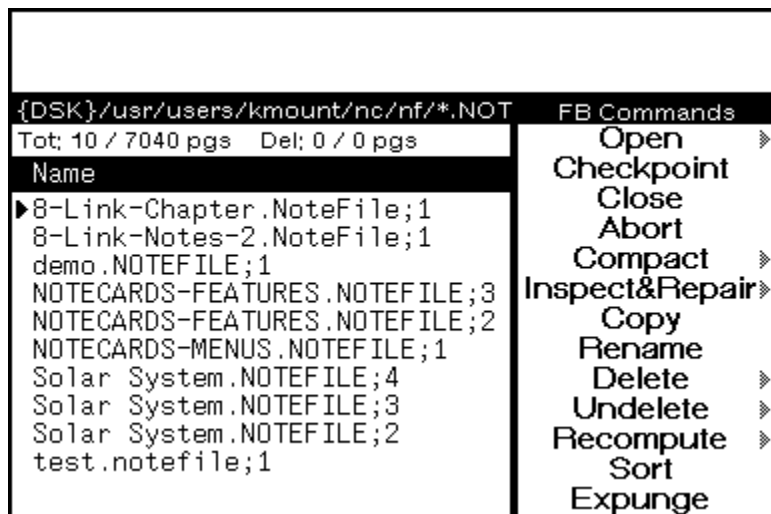


Figure 12-17. The NC FileBrowser.

The commands: **Open**, **Checkpoint**, **Close**, **Abort**, **Compact**, and **Inspect & Repair** can be looked up in this chapter.

The commands: **Copy**, **Rename**, **Delete**, **Undelete**, **Recompute**, **Sort**, and **Expunge** should be looked up in Chapter 14, The FileBrowser.

Card Options

The Card options allow you to perform a **Close**, **Delete**, **Copy**, or **Move**, on a series of cards or a card structure.

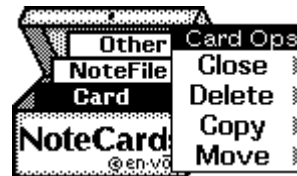


Figure 12-18. The "Card Ops" menu.

Close

Enables you to close any number of currently open cards by simply shift-selecting each one's card-ID into the "Selecting Cards" prompt window.

After you have specified your choices, select **Done** from the menu on top of the "Selecting Cards" prompt window, or type a carriage return. **Cancel** aborts this command. **Undo** removes the last selected card from the list.

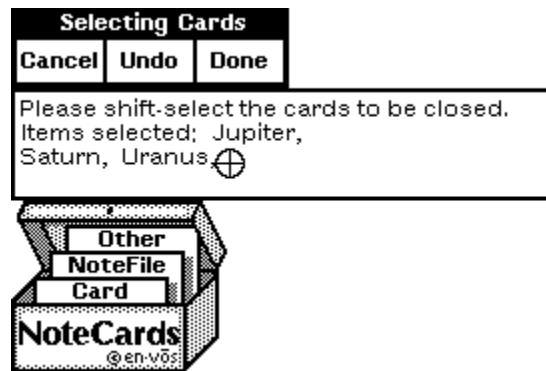


Figure 12-19. The "Selecting Cards" prompt window and menu.

The **Close** option has two submenu options.

Close and Close NoteCards

Perform the same action as **Close**.

Close Structure

Closes a linked list of cards. This operation follows both local and global links.



Figure 12-20. The **Close** submenu.

Once you have selected the cards from which to start the **Close Structure** operation, a menu opens asking you what types of links you want to follow. When you are done, select **Done**. **Reset** sets the menu to the values it had when it came up. **Abort**, terminates the operation.

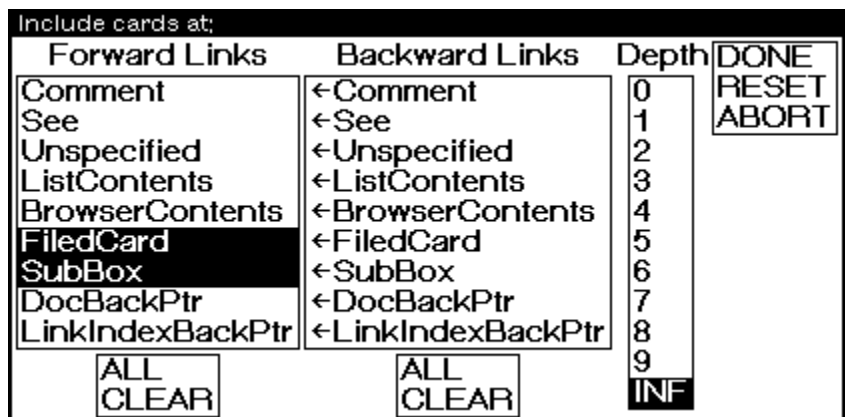


Figure 12-21. The "Include cards at:" menu.

For an explanation of Forward and Backward links see Chapter 8, Links. The **Depth** variable determines how many link levels are traversed in carrying out the operation. If set to a depth of 2, only cards two levels down are closed. For example, if you had cards 1,

2, 3, and 4 on the screen, each forming a sequence (card 1 only pointing to card 2, and card 2 only pointing to card 3, etc.), then if the depth were set to two and you started the close operation from card 1, only cards 1, 2, and 3 would be closed, because only two levels of links would be traversed. Card 4 would remain open on the screen after the **Close Structure** operation.

Delete

Works just like **Close**.

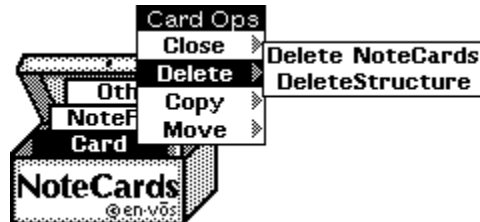


Figure 12-22. The **Delete** submenu.

We caution you to be very careful with the **Delete Structure** command as it is very easy to unintentionally delete large numbers of cards. This command is most useful in conjunction with **Copy Structure** as a way of performing a safe **Move Structure**. First you copy the structure you think you want and then once you have verified that you have gotten the structure you wanted, you use **Delete Structure** to remove it.

Copy

Works just like **Close**. Allows you to copy a group of cards or a card structure to another FileBox which may be in a different notefile.

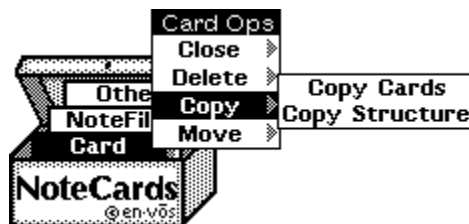


Figure 12-23. The **Copy** submenu.

Move

Works just like **Close**. Allows you to move a group of cards or a card structure to another FileBox which may be in a different notefile.

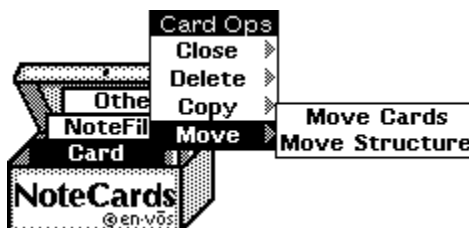


Figure 12-24. The **Move** submenu.

Other Options



Figure 12-25. The "Other Ops" menu.

Edit Parameters

The **Edit Parameters** option brings up the System Parameters menu. This menu is discussed in detail in Chapter 13, System Parameters.

NF Indicators On

NF Indicators On and the submenu option **Indicators On** turn on notefile indicators for all the cards on the screen.

Indicators Off

Turns off all notefile indicator windows for all the cards on the screen. Note that whether or not cards come up with notefile indicators is determined by the system parameter **Show Notefile On Cards**. To learn more about this parameter and changing system parameters, see Chapter 13, System Parameters.

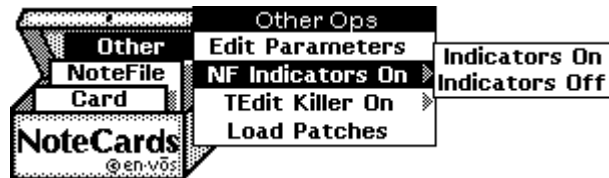


Figure 12-26. The "NF Indicators On" submenu.

Notefile indicators are small windows on the tops of cards which show the name and path of the card's notefile.

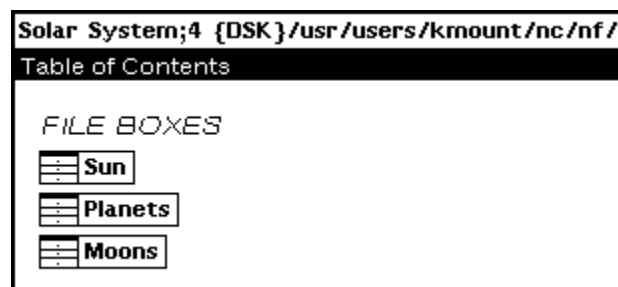


Figure 12-27. A notefile indicator window attached to the "Table of Contents" card from the Solar System notefile.

TEdit Killer On

Kills TEdit processes after they have been around for a specified period of time.

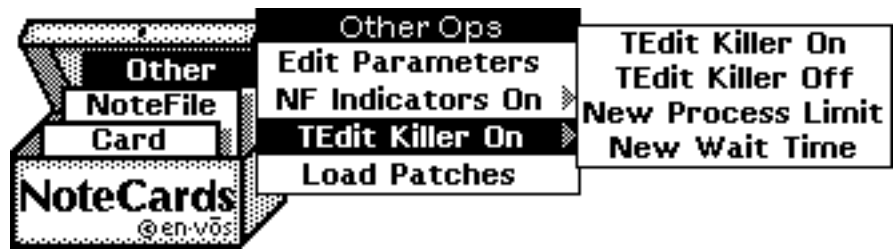


Figure 12-28. The "TEdit Killer On" submenu.

Since every text-based card and every TEdit window, including TEdit command menus, is a separate TEdit process it is very easy to get twenty or thirty TEdit processes all vying for machine time. This can greatly slow down your machine's response time. If you like to work with many cards and windows open on the screen at one time turning the TEdit killer on will help improve your machine's response time. Note that TEdit killer only affects text-based cards. The processes for sketch-based cards are unaffected. To restart a TEdit process that has been killed in this manner, you only have to click in the TEdit window.

TEdit Killer On

Turns on the TEdit killer process.

TEdit Killer Off

Turns off the TEdit killer process.

New Process Limit

Sets the number of TEdit process which are allowed to exist. Selecting this option brings up the following menu.

```

Old TEdit
process limit: 5

  0

  - AB CLE
    OR AR
  1 2 3
  4 5 6
  7 8 9
bs 0 ok
  
```

Figure 12-29. The TEdit process limit number pad.

A response of "0" allows you to have one TEdit process running and not none.

The default value is "5."

New Wait Time

Sets the number of second which a TEdit process can be idle before it will be killed. Selecting this option brings up the following menu.

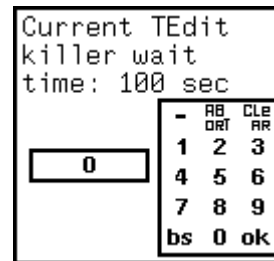


Figure 12-30. The TEdit killer wait time number pad.

Note: If both the TEdit process limit and the TEdit killer wait time are set to "0," it can freeze your system.

The default value is "100."

Load Patches

Forces the loading of any patches which were distributed with the NoteCards sysout.

[This page intentionally left blank]

13. SYSTEM PARAMETERS

System parameters allow you to customize the NoteCards environment to your particular needs and tastes.

This chapter explains:

- How to set system parameters.

- What each system parameter does.

Accessing System Parameters

System parameters are set from the "NoteCards Parameters" menu. To access this menu, hold the left mouse button down on the **Other** option on the MenuBox Icon. When the "Other Ops" menu comes up, still holding the mouse button down, slide off and select the **Edit Parameters** option. At this point, you release the mouse button. You can also execute a single rapid left-button click on the Other menu option and the "Other Ops" menu will come up and stay up. To select the option you want click on it.

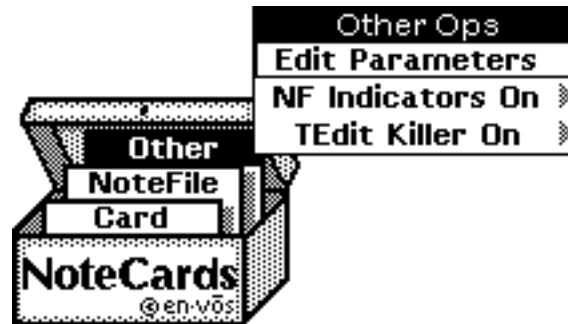


Figure 13-1. The MenuBox Icon with the **Edit Parameters** option on the **Other** menu option selected.

This action brings up the "NoteCards Parameters" permanent menu, shown below in Figure 13-2.

Each parameter is preset to a default value. Placing the cursor over the name of the parameter and depressing the left mouse button will toggle the value, listed to the right, between "Yes" and "No" if binary, and allow selection from an appropriate menu otherwise.

NoteCards Parameters	
NoteCards System Parameters	
Extra TEdit Props	None
Include Card Object In ShowInfo	No
Del TEdit Process When Shrinking	No
Fonts	
Menu Font	(HELVETICA 10 BOLD)
Default Font	(HELVETICA 12 STANDARD)
Link Icon Font	(HELVETICA 10 BOLD)
NoteFile Indicator Font	(HELVETICA 10 BOLD)
NoteFiles	
Show NoteFile On Cards	Yes
New NoteFile Initial Size	1000
Menu Lingers After NoteFile Close	Yes
Show Table of Contenes on Open	Yes
Cards	
Force Filing	Yes
Force Titles	No
Default Card Type	Text
Close Cards Off Screen	No
Bring Up Cards At Previous Pos	No
FileBox Cards	
Markers In FileBoxes	Yes
Alphabetized FileBox Children	No
Browser Cards	
Special Browser Specs	No
Arrow Heads In Browsers	None
Link Dashing In Browsers	No
Sketch Cards	
Attach Sketch Menu	No
Links	
Link Icon Border Width	1
Link Icon Multi Line Mode	No
Link Icon Max Width In Pixels	1024
Cross File Link Mode	ASK
Link Icon Show Title Default	Yes
Link Icon Attach Bitmap Default	Yes
Link Icon Show Link Type Default	No
Use Deleted Link Icon Indicators	Yes

Figure 13-2. The "NoteCards Parameters" menu.

NoteCards System Parameters

Extra TEdit Props

Does not initially appear on the list of parameters for the user version of NoteCards. It is obtained by going into the **Low Level Tools** option on the background menu. This parameter is to be used by people extending the NoteCards environment. Users of NoteCards should not add this parameter to their "NoteCards Parameters" menu.

The value of this parameter is a property list that is appended to the property list passed as the PROPS argument to every call to TEdit. This parameter allows you to set up the initial TEdit props for all new text-based cards.

Selecting this property brings up a type-in window above the parameter editor window. To set the parameter value, type in a sequence of zero or more attribute/value pairs followed by carriage return. The possibilities for attributes and values are described in *A User's Guide to TEdit*, Chapter 8, The Programmer's Interface to Tedit, in the section "Using the Top-level TEdit function".

For example, to set the default line leading to 2 you would type:

PARALOOKS (LINELEADING 2)

You should be very careful when setting other than LOOKS-related properties. In particular, NoteCards sets properties like QUITFN and PROMPTWINDOW itself, overriding any alternative values for these properties that you may have typed in.

Special note: If for some reason, you need to have an expression EVAL'ed in the typein window, precede it with ctrl-y.

The default value is "None."

Include Card Object In ShowInfo

When the value of this parameter is Yes, the Show Info inspector will include the Card Object. This option is for people extending or modifying the NoteCards system.

The default value is "No."

Del TEdit Process When Shrinking

If the value of this parameter is "Yes," when you shrink any text-based card, NoteCards will kill the TEdit process behind it. If the value is "No," the TEdit process remains running even when the card is shrunk to an icon.

Expanding an icon for a card whose TEdit process has been killed will not automatically restart the TEdit process. Instead, the user has to click the MIDDLE button in the substance of the card, and select the **New Edit Process** item from the singleton menu that appears.

Running TEdit processes use up limited resources. As a result, only about 25 to 30 TEdit processes can be active simultaneously. Setting the value of this parameter to "Yes," eliminates

unnecessary use of these limited resources when text-based cards are shrunk to icons. The cost is the extra effort needed to restart the process when the card is expanded.

The default value is "No."

Font Parameters

When you select any of the parameters in the Fonts section the system displays the "Please select a font:" menu.

Family	Size	Face	
TIMESROMAN	8	STANDARD	DONE RESET ABORT
HELVETICA	9	BOLD	
GACHA	10	ITALIC	
CLASSIC	11	BOLDITALIC	
MODERN	12		
TERMINAL	14		
TITAN	18		
CREAM	24		
OLDENGLISH	30		
	36		

Figure 13-3. The "Please select a font" menu.

To change the link icon font, select the values you want for each of the parameters with the mouse and click on **DONE**. **RESET** returns the values to their original settings. **ABORT** returns the values to their original settings and exits.

The value for each font parameter is three-element list which includes the font family, the font size in points and the font face.

Menu Font

The value of this parameter is a description of the font in which the text in all menus will be printed.

The default value is "(HELVETICA 10 BOLD)."

Default Font

This parameter controls the font all new text-based cards will use.

The default value is "(HELVETICA 12 STANDARD)."

Link Icon Font

The value of this parameter is a description of the font which is used to print the card name and link type that appear inside the links.

The default value is "(HELVETICA 10 BOLD)."

Notefile Parameters

Show Notefile On Cards

In order to associate a card with its notefile, NoteCards provides a notefile indicator window which attaches to the top of the card's title bar.

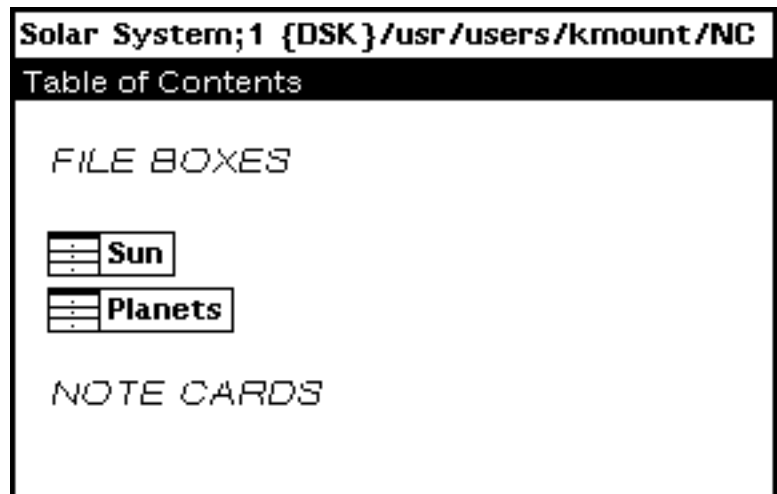


Figure 13-4. A card with its notefile indicator window attached to the top of the card.

The notefile indicator is constructed by concatenating the root file name, the version number, and the full file name. The indicator is left justified. If the card is too narrow to show the entire Indicator string, the full file name is truncated before the root name and version number as shown.

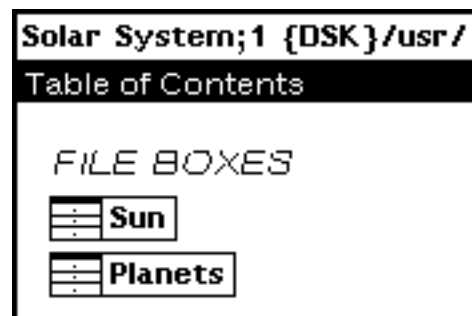


Figure 13-5. A card with its notefile indicator window attached to the top of the card, showing a truncated full file name.

If the NoteCards system parameter **Show Notefile On Cards** has the value "Yes," then a notefile indicator is attached to every card as it is brought up on the screen.

Occasionally, after you have reshaped a card, a piece of the text in the notefile indicator window will be garbled or broken. To redisplay the text in this window, place the point of the mouse cursor in the notefile indicator window, hold down the right mouse button, and select the **Redisplay** option from the window menu which pops up.

The default value is "No."

New Notefile Initial Size

When you create a new notefile, the system consults the value of this parameter to determine the initial size of the notefile index. The notefile-index size is equal to the number of cards the notefile can hold. Larger notefiles take a little longer to create. Notefiles can be enlarged later on, using the **Inspect & Repair** option.

The default value is "1000."

Menu Lingers After Notefile Close

If the value of this parameter is "No," the notefile Banner is removed from the screen when the notefile is closed. If the value is set to "Yes," then the Banner remains on the screen with its menu items greyed.

The default value is "Yes."

Card Parameters

Force Filing

If you set **Force Filing** to "Yes," the system will require you to file every card in a FileBox before the card can be closed. Setting the value to "No" eliminates this forced filing, but the system will file the card in the "To Be Filed" FileBox if the card has no parent FileBox when you close it.

The default value is "Yes."

Force Titles

If you set **Force Titles** to "Yes," this parameter will require you to title a card before it can be closed. Setting the value to "No" eliminates forced titling. When a card is not titled, it is named "Untitled" by default. This may cause confusion if several cards are left untitled. The system, however, has a unique identifier for each card and will continue to regard them as unambiguous entities.

The default value is "Yes."

Default Card Type

When you select **the New Cards** option from the notefile Banner with the left mouse button, a card of the default type is immediately created and displayed. This parameter allows you to change this default to suit your needs. Select the new default card type from the list of choices provided.

The default value is "Text."

Close Cards Off Screen

If set to "Yes," then when a card is closed, it is first moved off the screen so that the close happens invisibly.

The default value is "No."

Bring Up Cards At Previous Pos

If set to "Yes," cards are brought up on the screen in the position they occupied when you last closed them. You are not asked to position a frame image.

The default value is "No."

FileBox Card Parameters

Markers In FileBoxes

If set to "Yes," then new fileboxes will contain the markers "FILE BOXES" and "NOTE CARDS." New child boxes are inserted under the FILE BOXES marker and new child cards under the NOTE CARDS marker. If set to "No," then new fileboxes come up without markers and new children are inserted at the current cursor position . Note that regardless of the **Markers In FileBoxes** setting, if a filebox has no markers (because you've deleted them) then new children are inserted at the cursor position.

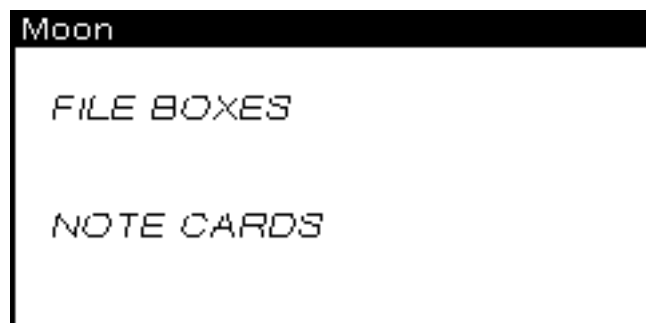


Figure 13-6. A FileBox card showing the markers "FILE BOXES" and "NOTE CARDS."

The default value is "Yes."

Alphabetized FileBox Children

If set to "Yes," then FileBoxes created while this parameter is set to "Yes" will automatically alphabetize any new cards filed in them.

The default value is "No."

Browser Card Parameters

Special Browser Specs

If set to "Yes", this parameter causes a sequence of five prompts to appear above the Browser NoteCard during the Browser creation process. These prompts mainly concern the node-link graph properties documented in the "Browser Cards" section of Chapter 11, System Cards.

For example, this option may be used to create a vertical browser instead of the default horizontal presentation. Note that a carriage return given as a response to any of the prompts will cause the Special Browser Spec to retain its default value.

The default value is "No."

Arrow Heads In Browsers

This dictates whether arrow heads are drawn on browser links. The variable can be set to either AtMidpoint, AtEndpoint, or None. See the Browser card section of Chapter 11, System Cards for details.

The default value is "None."

Link Dashing In Browser

If set to "Yes", this parameter builds a Browser with dashed links. Each link in a Browser refers to the type of link being followed. With link dashing, each link included in the Browser is assigned a different style of dashing (up to six styles). A "No" setting means links will be presented as solid lines in a Browser. A Browser with link dashing will take a little longer to create than one with solid lines.

There may be problems printing Browsers with dashed lines.

The default value is "No."

Sketch Card Parameters

Attach Sketch Menu

If set to "Yes," the Sketch menu is automatically attached to the right-hand side of each Sketch card when it is opened.

The default value is "No."

Link Parameters

Link Icon Border Width

Changes the border size of a link icon.



Figure 13-7. A link icon with a border size of 3.

The link icon border width can also be set to zero for link icons that blend more smoothly with text.

This is a  **Link Icon** with no border.

Figure 13-8. A link icon with a border size of 0.

The default value is "1."

Link Icon Multi Line Mode

This parameter works in conjunction with **Link Icon Max Width in Pixels**. Multi line mode determines whether the link icon will allow multiple text lines or whether all the text will be forced to appear on a single line. If **Link Icon Multi Line Mode** is set to "Yes," all link icons will display in multiple lines, with each line fitting within the width specified by **Link Icon Max Width in Pixels**. Line breaks occur at word boundaries.

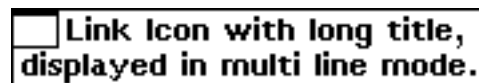


Figure 13-9. A link icon in multi line mode.

The default value is "No."

Cross File Link Mode

All links in NoteCards are bidirectional or two-way with one possible exception. When you are creating cross-file links, these links can be unidirectional or one-way. A one-way link means that a source card contains a to-link pointing to the destination card, but the destination card does not contain a corresponding from-link pointing back to the source card. This allows you to update a source-card file without being forced to open any corresponding destination-card files.

One-Way	Creates unidirectional links between cards in different notefiles.
Two-Way	Creates the usual bidirectional links between cards in different notefiles.
Ask	The system asks you whether you want One-Way or Two-Way links each time you create a cross-file link. The default value is "Ask."

Link Icon Max Width in Pixels

Sets the maximum width in pixels of a link icon. This value includes the card-type bitmap, if it is displayed. All links are truncated at the width specified by this parameter.



Figure 13-10. A truncated link icon in single line mode.

The default value is "1024" pixels.

Link Icon Show Title Default

This parameter and the following two determine the manner in which link icons are displayed if these parameters are not currently specified in the icon. Which is to say, if the link icon display mode values are set to "System Default."

If the title field in a link icon is set to "System Default" then the **Link Icon Show Title Default** parameter is consulted to determine whether the title will be shown inside the link icon. If it is set to "Yes" then the title will be shown.

The default value is "Yes."

Link Icon Attach Bitmap Default

If the bitmap field in a link icon is set to "System Default" then the **Link Icon Attach Bitmap Default** parameter is consulted to determine whether the card-type bitmap is shown next to the link icon. If it is set to "Yes" then the card-type bitmap is shown.

The default value is "Yes."

Link Icon Show Link Type Default

If the link type field in a link icon is set to "System Default" then the **Link Icon Show Link Type Default** parameter is consulted to determine whether the link type will be shown inside the link icon. If it is set to "Yes" then the link type will be shown.

The default value is "No."

Use Deleted Link Icon Indicators

If the value of this parameter is "Yes," when you delete a card, all link icons which point to that card are replaced with a deleted icon.



Figure 13-11. A deleted icon.

If the value of this parameter is "No," the links to the card are simply removed, leaving no trace behind.

Delete icons are useful when you may be deleting cards where you have link icons pointing to them and these link icons are referred to in the text.

Eg. "This link  takes you to the next card in the sequence." If the "Next" card were subsequently deleted with this parameter set to "Yes," this would become, "This link  takes you to the next card in the sequence." If this parameter were set to "No," the sentence would become the following more obscure version, "This link takes you to the next card in the sequence." with no visible link.

The default value is "Yes."

[This page intentionally left blank]

The FileBrowser provides a convenient user interface for manipulating files stored on a workstation or file server. It enables you to see, edit, delete, print, load, copy, move, rename, compile, sort, and get several types of information about groups of files.

User Interface

Starting FileBrowser

To open a browser on a set of files select the FileBrowser command from the background menu.

FileBrowser will prompt you to create a window by presenting you with a dashed rectangle with the mouse cursor and a small geometric design at the lower right corner.

1. Move your mouse until the upper left corner of the rectangle is where you want it on the screen.
2. Hold down the left mouse button and move your mouse down and to the right, thus expanding the window diagonally, until the window is the right size.
3. Release the mouse button. This creates a window group on your screen in the outlined area.

Next, FileBrowser prompts you for a file pattern. Type a pattern, as described in the section "Specifying What Files to Browse," below.

FileBrowser enumerates the set of files matching the pattern you requested to see. While the enumeration is in progress, the **Recompute** command is grayed out. When the enumeration is finished, you may select files and issue commands. You can scroll the window at any time, even while the browser is busy.

If FileBrowser can't find any files matching the pattern you specified, or you decide you specified the wrong pattern and want to try again, you can specify a new file name pattern from within the browser using the **New Pattern** command; see **Recompute** in the section "FileBrowser Commands," below.

You can have as many active FileBrowsers open at once as you like.

Specifying What Files to Browse

A full file name in NoteCards consists of a device or host (such as your local disk, a file server), a principal directory and zero or more subdirectories, a file name (possibly including an extension), and a version number. These fields are put together in the form

```
{HOST}<DIRECTORY>SUBDIRECTORY>FILENAME.EXTENSION;VERSION
```

A file name pattern, as specified to FileBrowser, consists of a file name with one or more pieces omitted or filled with wild cards (*).

All the files matching the pattern are listed by FileBrowser. Thus, you can browse all the files in a particular directory, all the files in a subdirectory of that directory, all the files in a directory with a particular extension, and so forth. The wild card `*` can be used to stand for zero or more consecutive characters in the file name. You can use as many wild cards in a pattern as you wish.

If you leave out some of the fields in a file name pattern, the missing fields are defaulted by the system. Omitted fields in the front of the pattern, i.e., the host, device, or directory fields, are filled in by consulting your connected directory. Other omitted fields are filled in with wild cards unless they are explicitly omitted; i.e., the field is empty, but the preceding punctuation is still present. In more detail, some of the cases are as follows:

If you leave out the name of the host/device, specifying `<DIRECTORY>FILENAME`, FileBrowser will use the name of the host/device for the directory to which you are currently connected.

If you leave out both the device and directory names, specifying `FILENAME`, FileBrowser will use the device and directory to which you are currently connected.

If you do not specify a file name, FileBrowser lists all the files in the specified directory (or the connected directory if you also omitted the host and directory).

If you leave out the extension of a file name, FileBrowser lists all the files with the specified file name and any extension. If you omit the extension but include the period that usually precedes the extension, FileBrowser lists only the files with the specified name and *no* extension.

If you omit the version number of the file name, FileBrowser lists all versions of the matching files. If you omit the version number but include the semicolon that usually precedes the version, FileBrowser lists only the highest version of the matching files.

Thus, the minimal pattern you can type is `*` (asterisk—enumerate all files in the connected directory) or `;` (semicolon—enumerate just the highest version of all files). If you press the RETURN key without giving a pattern, FileBrowser aborts the prompt for a pattern, leaving you with an empty browser in which the only things you can do are change some FileBrowser parameters (see the subcommands of **Recompute** in the section "FileBrowser Commands," below) and then use the **Recompute** command to be prompted for a pattern again.

Examples

The pattern `*` specifies all files in the connected directory. It is equivalent to `*.*` or `*.*.*`.

The pattern `<NoteCards>Demo` specifies all files in directory NoteCards with name Demo and any extension. It is equivalent to `<NoteCards>Demo.*.*`.

The pattern `<NoteCards>Demo.` specifies all files in directory NoteCards with name Demo and *no* extension. It is equivalent to `<NoteCards>Demo.*`.

The pattern *.TEdit specifies all files in the connected directory with the extension TEdit. It is equivalent to *.TEdit;.*.

The pattern *.TEdit; specifies only the newest version of all files in the connected directory with the extension .TEdit.

The pattern <NoteCards>A*E specifies all files in directory NoteCards whose names begin with A and end with E and have any extension.

The pattern {DSK}<NoteCards>*MY* specifies all files in directory {DSK}<NoteCards> whose names contain the substring MY and any extension.

Using the FileBrowser Window

The FileBrowser window has six major subwindows, which from top to bottom are as follows.

Enumerating {DSK}/usr/users/kmount/NC/DOC/ENVOS/1.1/WORKING/*.*;* ...done			
{DSK}/usr/users/kmount/NC/DOC/ENVOS/1.1/WORKING/*.*;* at 12:26			
Total: 32		Deleted: 0	
Name		Created	
000-OUTLINE.TEDIT;13		4-Apr-89 11:28:15	
001-TITLE-PAGE.TEDIT;7		6-Mar-89 14:50:11	
002-PREFACE.TEDIT;15		17-Feb-89 09:52:39	
003-TOC.TEDIT;2		8-Mar-89 14:00:07	
01-INTRODUCTION.TEDIT;10		23-Feb-89 13:32:25	
02-SYSTEM-REQUIREMENTS.TEDIT;21		23-Feb-89 11:38:09	
03-SOFTWARE-INSTALLATION.TEDIT;22		23-Feb-89 13:40:36	
04-SYSTEM-USE-ISSUES.TEDIT;21		7-Mar-89 12:05:51	
05-NOTECARDS-BASICS.TEDIT;48		5-Apr-89 16:20:08	
06-BUILDING-NC-STRUCTURES.TEDIT;13		4-Apr-89 10:40:25	
Info Options			
Length	Pages	Created	Read
ByteSize	Type	Written	Author

Delete	»
Undelete	»
Copy	
Rename	
Hardcopy	»
See	»
Edit	
Load	»
Compile	»
Expunge	
Recompute	»
Sort	

Figure 14-1. The FileBrowser display.

Prompt window This topmost subwindow is where FileBrowser prints messages about what it is doing and receives input from you. Its contents are cleared before every command.

Tally window This subwindow immediately below the prompt window keeps a running tally of the total number of files listed in the window and the number of files that you have marked for deletion. In addition, if one of the attributes you are displaying is a size attribute (Pages or Length, as in the INFO menu, described below), this window maintains a tally of the total number of pages in the files listed and the files marked for deletion.

This window also has a title bar across the top identifying the pattern you specified and the time at which the directory enumeration was performed.

The window is blank while the files are being enumerated.

Browser window This is the principal subwindow, in which the files matching the specified pattern are listed. Each file's name appears at the left, and various attributes of the file are displayed in columns to the right. A title bar across the top of the browser window identifies the contents of each column (e.g., Name, Pages, Created). The files are listed in alphabetical order, with multiple versions of the same

file listed in decreasing version order; i.e., the newest version appears first. The width of the column listing the file names is initially chosen to be appropriate for average-sized file names. If the files you asked to browse have particularly long names, then when FileBrowser has finished listing all the files it may choose to redraw the browser window with the attribute columns moved farther to the right to accommodate the longer file names.

Command menu

This menu appears vertically along the right side of FileBrowser window (under a title bar "FB Commands") and lists the commands that you may select to perform operations on the files in the browser, or to change the appearance of the browser. Most of the commands operate on the set of currently selected files (see the section "Selecting Files" below). Some commands have subcommands, as indicated by the small triangle alongside them, which can be selected by holding down the left mouse button and sliding the mouse to the right over the triangle.

Info menu

An additional subwindow, the Info menu, is not normally displayed. It is used to change the set of file information (attributes) displayed in the browser (see the section "Getting Information About Files," below).

Scroll bar

If there are more files in the listing than fit at one time in the browser window, you can scroll the browser window to view more files. Slide the mouse cursor out the left side of the browser window to get the scroll bar and press the left mouse button to scroll the region up and the right mouse button to scroll it down. Pressing a mouse button when the cursor is near the bottom of the scroll bar will scroll the region by larger increments than when the cursor is at the top.

You can also press the middle mouse button in the scroll bar to move the listing to the place that corresponds to that position in the scroll bar. For example, pressing the middle mouse button when the cursor is at the bottom of the scroll bar will display the end of the listing. This quick-scrolling technique is called thumbing. The gray box in the scroll region indicates where the currently displayed contents are, relative to the entire contents of the browser.

Similarly, if there is more attribute information than fits in the browser window, you can scroll the browser window horizontally to view the rest of the attribute information. To do this, slide the mouse out the bottom of the browser window to get the horizontal scroll bar. The left button scrolls to the left, the right button to the right.

Selecting Files

Most FileBrowser operations are performed by selecting a single file or set of files, then giving a command that specifies what you want to do with the selected files. The current selection is indicated by a small triangle in the left margin of the browser next to each selected file.

```
| 003-TOC.TEDIT;2  
| ▶01-INTRODUCTION.TEDIT;10  
| 02-SYSTEM-REQUIREMENTS.TEDIT;21  
| ▶03-SOFTWARE-INSTALLATION.TEDIT;22  
| ▶04-SYSTEM-USE-ISSUES.TEDIT;21  
| ▶05-NOTECARDS-BASICS.TEDIT;48  
| 06-BUILDING-NC-STRUCTURES.TEDIT;13
```

Figure 14-2. Four selected files marked by triangles.

To select one file, point to any part of the line (which lists the file name and its attributes) and press the left mouse button. If other files are already selected, this unselects them; thus, a file selected with the left mouse button is always the only selection.

To add a single file to the current selection, press the middle mouse button at any place in the line. The file is selected without unselecting any other file.

To remove a single file from the current selection, hold down the control key and press the middle mouse button at any place in the line. The file is unselected without affecting any other file.

To extend the selection to include a group of contiguous files, that is, to select all the files between a file and the nearest already selected file, press the right mouse button on any part of the line. You can only extend the selection from the first selected file upward, or the last selected file downward. In addition, files marked for deletion are not normally selected when you extend.

If you want to include all files, both deleted and undeleted, hold down the control key while extending the selection.

Some lines in a FileBrowser display are directory-only lines. These lines are slightly indented and name the directory and subdirectory to which the files listed below that line belong. You cannot select in these lines, though you can copy-select them (see the section "Copy-Selecting Files," below).

Commands that Require Input

Some FileBrowser commands require input from you. For example, the **Copy** command requires that you supply a destination file name. When a command requires input, FileBrowser prints a prompt message in its prompt window. This is usually followed by a default answer. If you want the default answer, you can just press the carriage return to finish the input. If you want to specify a different answer, simply start typing it; the default answer is erased and your answer replaces it.

Alternatively, you can modify the default answer by backspacing over individual letters, or typing control-W to back up over complete words. Typing CONTROL-Q erases the entire answer. You can also use the mouse to edit your answer, using the same rules as followed by the Executive (see the documentation of TTYIN). Briefly, the left mouse button positions the caret at a character boundary; the middle mouse button positions the caret at the nearest word boundary; and the right mouse button deletes the characters between the caret and the mouse.

When you have finished, position the caret at the end of your answer, if it isn't there already, and press the carriage return. You can also type CONTROL-X to finish your answer even if the caret isn't at the end.

If you change your mind and want to abort the command, supply an empty input; i.e., if there is an answer in progress, backspace over it or type CONTROL-Q to erase it, then press <RETURN>. FileBrowser prints "aborted" and aborts the command. In most situations, the CONTROL-E interrupt can also be used to abort your answer.

While you are typing an answer, you can copy-select file names out of the browser (or any other browser), as described below in the section "Copy-Selecting Files". This can be useful, for example, if you wish to rename a file to a similar name in the same directory, or move a file into a subdirectory listed in the browser.

Aborting Commands

During commands of indefinite duration, such as **Recompute** or **Copy**, FileBrowser adds another command to the browser, **Abort**.

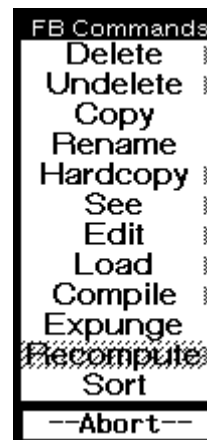


Figure 14-3. The FileBrowser menu with the Abort option attached.

Clicking on the **Abort** command will immediately abort the current operation. Aborting some commands can take a little while, as FileBrowser may need to do some cleaning up, so the **Abort** command is greyed out during this time to show you that it is doing something.

Quitting the FileBrowser

To quit a FileBrowser, simply close the browser window. If any files have been deleted but not expunged, a small menu will pop up listing two options: **Expunge deleted files** and **Don't Expunge**. If you choose **Expunge deleted files**, the files will be expunged before the window closes. If you choose **Don't Expunge**, your deletions are ignored. If you click outside the menu, no action is taken, and the **Close** command is aborted.

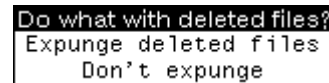


Figure 14-4. The "Do what with deleted files?" menu.

If you have finished with FileBrowser only temporarily and want to put it aside to work on later, you can shrink the browser by selecting the **Shrink** command from the right-button background menu. The browser shrinks to an icon which displays the file pattern inside the browser. If any files are marked for deletion, you will be prompted with the same menu of expunge options as when you close a browser.



Figure 14-5. A FileBrowser shrink icon.

Copy-Selecting Files

You can copy-select file names from a FileBrowser into other windows, such as TEdit windows, by holding down the COPY or SHIFT key while selecting a name in the window. The full name of the file is inserted as if you had typed it where the input caret is flashing. You can also copy-select in a directory-only line, in which case the full directory name is inserted in your type-in.

Getting Hardcopy Directory Listings

You can get a hardcopy listing of the directory displayed in a FileBrowser by using the regular window **Hardcopy** command. Press the right button in FileBrowser's prompt window or tally window and select **Hardcopy** from the menu. FileBrowser will produce a hardcopy listing of the files and the attributes displayed in the browser.

If the browser displays a large number of attributes, or your default printer font is too large, the listing may not accommodate all the attributes on one line, making the listing less readable. You may want to make the listing with fewer attributes, or use a smaller font for the listing.

FileBrowser Commands

Delete, Undelete

Removing a file from the file system using FileBrowser is a two-step process. First, you mark the file or files for deletion. Then you issue the **Expunge** command. Any time between the deletion and the expunge you can change your mind and undelete any of the files.

To mark a file or files for deletion, select them, then choose the **Delete** command. FileBrowser draws a line through the deleted files. It also adjusts the numbers in the tally window to show how many files are marked deleted and how many pages they contain. It is thus easy to see how much file space you will regain when you issue the **Expunge** command.

```

11-SYSTEM-CARDS.TEDIT;5
12-MENUBOX-ICON.TEDIT;11
13-PARAMETERS.TEDIT;9
14-FILEBROWSER.TEDIT;10
▶14-FILEBROWSER.TEDIT;9——
14-FILEBROWSER.TEDIT;8——
15-OTHER-TOOLS.TEDIT;27
16-PRINTING.TEDIT;52

```

Figure 14.6 Two files marked for deletion.

To undelete a file or files (i.e., to remove the deletion mark), select them, then choose the **Undelete** command. The lines through the files are removed, and the tally of deleted files is updated. The **Undelete** command has a single subcommand, **Undelete ALL Files**, which undeletes all the files in the browser, independently of whether they are selected. This is useful if you completely change your mind about deleting any files.

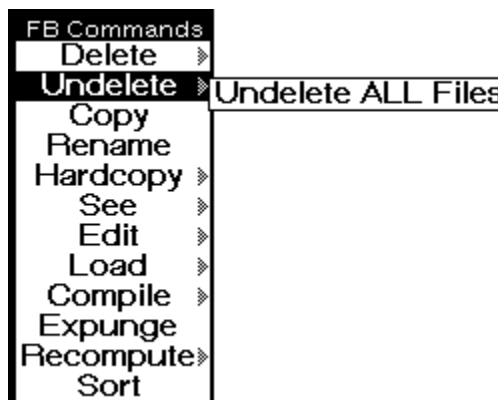
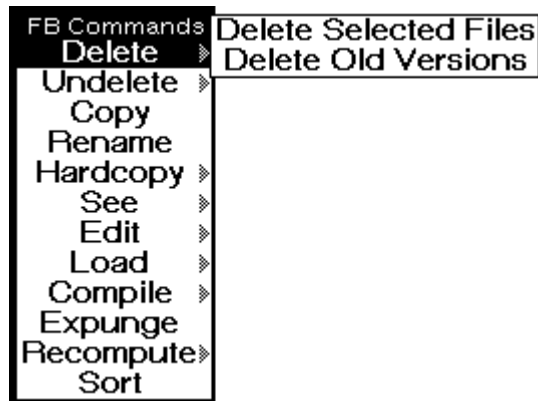


Figure 14-7. The **Undelete** submenu.

The **Delete** command has a useful subcommand, **Delete Old Versions**. When you have been editing a file in the text editor and performing repeated **Put** commands, multiple versions of the file accumulate, each more recent version denoted by a higher version number. The **Delete Old Versions** command is used to delete excess versions of the files displayed in the browser.

Figure 14-8. The **Delete** submenu.

To use this command, press the mouse down on the **Delete** command and slide the cursor out to the right, choosing the **Delete Old Versions** command. Unlike the **Delete** command (or **Delete Selected Files**, the equivalent subcommand), the **Delete Old Versions** command operates on all the files in the browser. FileBrowser prompts you for the number of versions of each file that you wish to retain. It offers the default of one version. You can accept the default or you can type a different number of your choosing, followed by a carriage return. FileBrowser then marks for deletion all but the most recent *n* versions of all the files in the browser, where *n* is the number you specified. Before issuing the **Expunge** command, you can, if you wish, scroll through the browser, undeleting any particular files for which you wish to retain more versions than you specified.

The **Delete Old Versions** command is sometimes useful even when you are not planning to actually expunge the files. This is because of the way extending the selection avoids deleted files (see the section "Selecting Files," above).

For example, if you wanted to copy only the most recent version of all the files in the browser to another location, you could do the following:

1. Use the **Delete Old Versions** command, retaining just one version. This marks deleted all files but the newest version of each.
2. Go to the start of the browser and select the first file, then scroll to the end of the browser and press the right mouse button to extend the selection to the end of the browser. You have selected exactly the newest version of each file.
3. Use the **Copy** command to copy those files.
4. Finally, use the **Undelete ALL Files** command to undelete all the old versions.

Copy

The **Copy** command is used to copy an entire file or set of files to another file system location; for example, from your disk to a file

server. Select the file(s) you wish to copy, then select the **Copy** command. FileBrowser prompts you to supply a destination.

If you selected just one file, FileBrowser prints the old name and offers a default, which consists of the same file name and either the same directory that was last used in a **Copy** or **Rename** in this FileBrowser, or the connected directory if this is the first use of **Copy** in this FileBrowser. You can accept the default or supply your own destination file name. If you supply just a directory specification, e.g., `{SERVER}<DIRECTORY>`, the file is copied to that directory under its current name. If you supply a complete name, the file is copied to that exact name.

Copy file {DSK}/usr/users/kmount/NC/DOC/ENVOS/1.1/WORKING/14-FILEBROWSER.TEDIT;10 to new file name: {Eris}<NoteCards>Doc>ENVOS>1.1>WORKING>14-FILEBROWSER.TEDIT				
{DSK}/usr/users/kmount/NC/DOC/ENVOS/1.1/WORKING/.*;* at 13:18 Thu 6-Apr FB Commands				
Total: 34 / 4734 pages		Deleted: 2 / 652 pages		
Name	Pages	Created		
000-OUTLINE.TEDIT;13	31	4-Apr-89 11:28:15		Delete
001-TITLE-PAGE.TEDIT;7	12	6-Mar-89 14:50:11		Undelete
002-PREFACE.TEDIT;15	24	17-Feb-89 09:52:39		Copy
003-TOC.TEDIT;2	47	8-Mar-89 14:00:07		Rename
				Hardcopy
				See

Figure 14-9. The Copy command confirming the name of the file to copy to.

Note: Unless you specify a version number in the destination file name, the version number of the new file will be 1, or one higher than the highest existing version of the file in the destination directory, independent of the version number of the old name.

Even files marked for deletion can be copied.

If you selected several files, FileBrowser notes how many files you wish to copy and offers as a default destination the connected directory. You can accept the default or supply a different directory. All the files are copied to that directory under the names they currently have.

You must supply a directory specification, e.g., `{SERVER}<YOURDIRECTORY>`, rather than a complete file name, since you can't copy multiple files to the same name. If you mistakenly type a file name, rather than a directory specification, FileBrowser will complain and abort the command.

If you want to copy files from different subdirectories, FileBrowser will ask, via a message in its prompt window, if you want to preserve the subdirectory structure at the destination. If you answer **Yes**, then the names at the destination will include not just the root name of each source file, but also all the subdirectory names below the greatest subdirectory prefix common to all the selected files (this common prefix is displayed as part of the question). If you answer **No**, then the names at the destination are formed solely from the root name of each file (the name displayed in the browser), ignoring any directory information each name might have. This can cause multiple files with the same root name to be copied into the same destination name (but with different version numbers, of course).

When copying (or renaming) multiple versions of the same file, FileBrowser does the copying in order of increasing version number, so that the versions at the destination are in the same relative order as at the source.

As each file is copied, FileBrowser prints a message giving the full name of the new file. If a file with the chosen name already exists, the new file's version number will be one higher; otherwise it will be version 1 (one). The new file will have the same creation date as the original file. If the destination file happens to be one that matches the pattern of the files in the browser, the new file is inserted in the appropriate place in the browser display. However, if it matches the pattern of some other FileBrowser, it is not inserted in that other browser's display (in other words, FileBrowsers do not know about each other). You would have to **Recompute** the destination FileBrowser to see that the file was copied into it.

Rename

The **Rename** command is used for changing the name of a file or group of files, or for moving a file or group of files to a different directory.

The **Rename** command is used in exactly the same way as the **Copy** command. If you rename a single file, you can supply a complete new name or just a directory; if you rename several files, you must specify a directory. As each file is renamed, FileBrowser prints a message giving the file's new name and removes the file from the browser display. If the new name belongs in the same browser, it is inserted in the appropriate place. If for some reason a file could not be renamed, this is noted in the FileBrowser prompt window. The reasons for the failure of a renaming operation are roughly the same as for the failure of an **Expunge**; the file is open, or you do not have the access rights needed to rename the file.

Note: If the destination of the rename is on a different file system than the original file, changing its name is equivalent to copying the file to its new name and then deleting the original file.

Hardcopy

You can print text files, TEdit, Sketch, Interpress, and PostScript files from FileBrowser. Select the appropriate file or files, then select the **Hardcopy** command. The **Hardcopy** command will determine what type of file you are printing and call the appropriate function for printing that file. Then the files will be printed one at a time on your default printer. The prompt window will display status messages telling you when files are being printed and when they are done (if your printer is one that provides this status service).

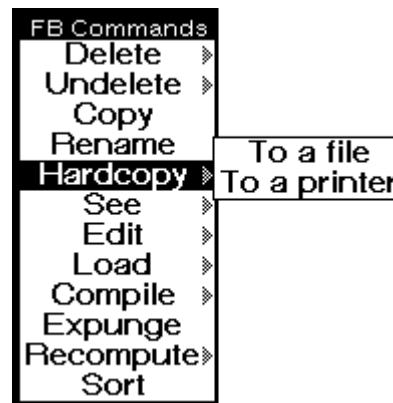


Figure 14-10. The **Hardcopy** submenu.

You may specify printing to a file or to a printer other than the default printer by means of a submenu from the **Hardcopy** command. This menu is the same as the one on the **Hardcopy** command in the background menu. Selecting **To a printer** presents you with a choice of printers from a menu. Selecting **To a file** prompts you to supply a file name. If you selected a single file, you must specify a single hardcopy file name (or accept the offered default). If you selected multiple files, then you must specify a pattern with a single asterisk somewhere in the "name" field, for example, *.INTERPRESS or Hardcopy-*.IP. The output file names are constructed by merging the pattern with each selected file name. If the name includes an extension that implies the type of print format (e.g., .IP or .INTERPRESS implies the Interpress print format), then a file of the specified type is made automatically. Otherwise, you are prompted to supply a print format type.

Note: For files stored on servers not supporting random access, FileBrowser is currently unable to determine that a file is in TEdit format unless the file has the extension .TEDIT. Therefore you should use TEdit to hardcopy TEdit files with other extensions. Use FileBrowser's **Edit** command (to call TEdit), then the **Hardcopy** command either from the TEdit Expanded Menu or from the right-button menu.

Note: To obtain a hardcopy of the directory itself, use the **Hardcopy** command from the right-button window menu. See the section "Getting Hardcopy Directory Listings".

See

When you browse a directory you sometimes want to see a file before printing or performing some other operation on it. To do this, select the file, then select the **See** command from the command menu. FileBrowser will prompt you to open a window by presenting you with a dashed rectangle and printing a message in the system prompt window. The window will be blank until FileBrowser starts printing the contents of the file in it.

There are actually four different **See** commands, as shown in the submenu for **See**. The **Fast SEE Pretty** and **Fast SEE Unformatted** commands are provided to let you quickly see the contents of a file, but not do anything fancy, such as scroll around

at random in the file. The slower **Scrollable & Pretty** command does let you scroll, and if the file contains formatting information of a kind that FileBrowser knows about (via the editors you have loaded), you will see the file formatted. However, this command does much more work, and may take a bit longer to show you even the first line of the file. The **Filebrowse** command is for use on files that are directories; it is described in the next section.

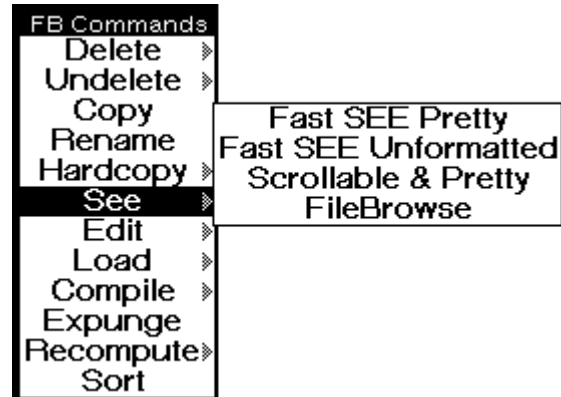


Figure 14-11. The **See** submenu.

The **Fast SEE Pretty** and **Fast SEE Unformatted** commands display the selected file in the display window one windowfull at a time. When the file fills the window, a small menu appears at the bottom-left corner of the window (or top-left if your display window is at the bottom of the screen) giving you the option of seeing more of the file or aborting the **See** command. If you issued the command with more than one file selected, you also have the choice of aborting just the display of this file or the entire **See** command.

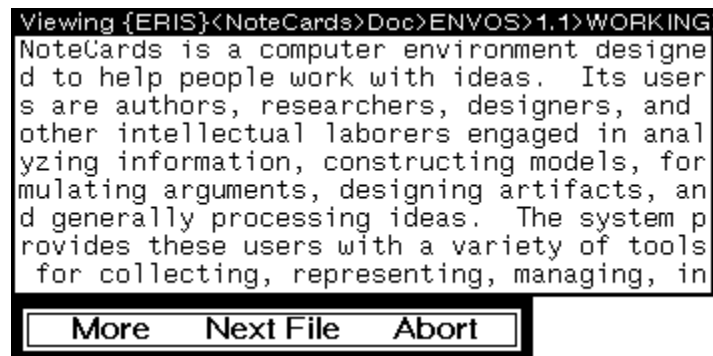


Figure 14-12. The **See** window control menu.

If you select **More**, the **See** command displays another windowful of the file. If you select **Next File**, the **See** command closes this file and goes on to display the next file in the current selection. If you select **Abort**, the entire **See** command is aborted. You can also abort the **See** command by closing the display window.

The next time you give a **Fast See Pretty** or **Fast SEE Unformatted** command, the same window will be reused.

The only difference between the **Fast See Pretty** and the **Fast SEE Unformatted** commands is the manner in which the characters of the file are processed as they are displayed.

The pretty (formatted) version interprets certain control characters found in source files to be font change commands, and interprets certain multibyte sequences as representing characters in the Xerox extended character set. It also squeezes out blank lines and shrinks the indentation of indented lines in order to better fit the text in a window that is generally much narrower than the standard file width. The formatted version is thus most appropriate for viewing source files and files containing plain text.

The unformatted version of the **See** command does no special processing on the characters whatsoever. It simply displays each eight-bit byte as a single character, uninterpreted. This means that bytes that do not represent normal printing characters may be displayed as black boxes, in the form ^x or #x, or as a flashing of the window (for the byte that represents the ASCII "bell" character). The Unformatted version is thus most appropriate for viewing binary files that also contain text portions that might be worth seeing; e.g., compiled files (those with extension .LCOM) or Interpress masters (extension .IP or .INTERPRESS).

The **Scrollable & Pretty** command views a file in a different way. This command brings up a new read-only TEdit window for viewing a file (only if TEdit is loaded in your system; otherwise, **Scrollable & Pretty** reverts to fast format). You can scroll and copy-select the file's contents at will, as with any TEdit window. If the file is a NoteCards source file, its contents are first formatted into a TEdit document, so that all the font information is retained. This formatting, however, can take a long time for a large file. For other kinds of files, the **Scrollable & Pretty** command is exactly like viewing the file in a regular TEdit window, except that you can't edit it. If you want to edit a file, use the Edit command instead of the See command.

You can keep the display window used by the **Scrollable & Pretty** command open as long as you like. The command uses a different window for each file you select. Simply close the window with the standard right-button window menu when you are finished with it.

Filebrowse

The **Filebrowse** command is a subcommand of **See** used to view a subdirectory in its own FileBrowser window. The selected file must be a (sub)directory. Subdirectory files appear in browsers on XNS file servers when the depth is finite (see the **Set Depth** command), and their names always end in ">". On Sun servers subdirectory file names end in "/".

```
medley/  
►nc/  
NOTES/  
PROGS/  
setup/  
sysouts/
```

Figure 14-13. Some subdirectories on a Sun server, indicated by trailing slashes.

The **Filebrowse** command prompts you for a region for a new FileBrowser window group, in which it proceeds to enumerate the contents of the selected subdirectory, to the same depth as the main browser used, if any.

Edit

The **Edit** command invokes an editor on the selected file. To specify an editor explicitly, use one of the commands on the submenu.

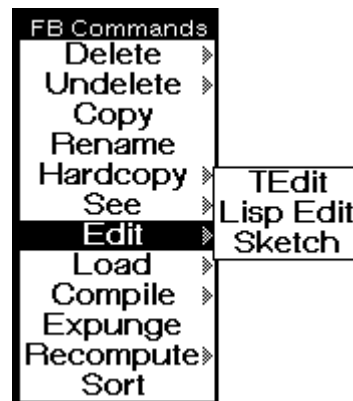


Figure 14-14. The **Edit** submenu.

To start up a TEdit editor on a selected text file, select **Edit** with the left mouse button. If you have recently closed a TEdit window, then TEdit will probably reuse that window; otherwise, you will be prompted to create an editor window. TEdit only remembers the most recently abandoned window, however, so if you issue the **Edit** command when you have several files selected, you will be prompted to create windows for all but the first file.

The subcommand **Lisp Edit** is for people extending the NoteCards environment. Do not use this command unless you are extending the NoteCards system.

If you select the main **Edit** command, without sliding off to the submenu, then FileBrowser's default editor, TEdit, is called.

Load

The **Load** command is for people extending the NoteCards system. Do not use this command unless you are extending the NoteCards system.

Compile

The **Compile** command is for people extending the NoteCards system. Do not use this command unless you are extending the NoteCards system.

Expunge

If you are sure you want to delete files permanently, choose the **Expunge** command. The **Expunge** command is grayed while FileBrowser expunges the files that were marked for deletion by the

Delete command. As each file is removed from the system, it is removed as well from the browser display, and the tally of total number of files and number of deleted files is updated, so you can see the progress of the command.

If for some reason a file can not be expunged, FileBrowser prints a message saying so in its prompt window, but continues to expunge the other files. The main reasons that prevent a file from being expunged are its being opened, either by you or some other user, or your not having the access rights required to delete it (if it is on a file server). See the section "Troubleshooting Problems with FileBrowser," below.

Note: The **Expunge** command is not affected by the current selection; it operates only on files marked for deletion, whether currently selected or not.

Recompute

FileBrowser's display shows those files that existed and matched the specified pattern at the time you created the browser. If you want the browser to reflect the latest state of the file system, use the **Recompute** command.

For example, if you open a FileBrowser on your directory, then save several versions of a TEdit file on that directory, the file listing will not display the new versions until you **Recompute**.

The **Recompute** command operates exactly as when you started up FileBrowser initially: it clears the display and tally windows, then enumerates the files matching the pattern. The **Recompute** command in the menu is grayed until the enumeration is finished. During this time you cannot scroll or perform any other operations on the browser. However, you can close the window if you want to abort the command and throw away the browser.

If any files are marked for deletion at the time you request a **Recompute**, FileBrowser will present the choice of expunging or undeleting the files, just as it does when you want to quit the browser (see the section "Quitting the FileBrowser," above).

The **Recompute** command also has a menu of subcommands that allow you to list different files or different information for the same set of files.

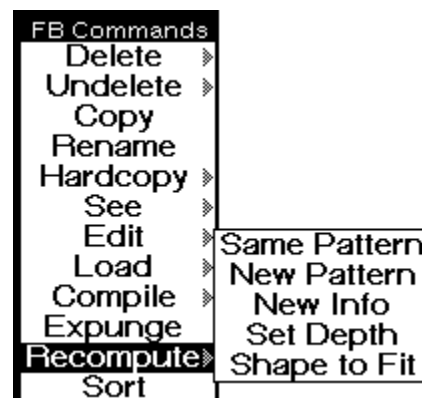


Figure 14-15. The **Recompute** submenu.

Same Pattern	Is the same as the main Recompute command, i.e., it enumerates the files matching the same pattern as before.
New Pattern	Lets you change the pattern, i.e., browse a new set of files. FileBrowser prompts you to supply a new file name pattern and offers the old pattern as an initial default. You can either type an entirely new pattern, replacing the one offered, or delete the old pattern one character at a time by backspacing. Press the carriage return when you have finished specifying the pattern. FileBrowser then enumerates the files matching this pattern, just as with the Recompute command. You can abort the command with the Abort button, or by erasing the whole pattern (by backspacing or using CONTROL-Q) and then pressing the <RETURN>.
New Info	Lets you change which attributes the browser displays. It is described in the next section.
Set Depth	Lets you change the depth to which FileBrowser enumerates a directory on an XNS file server. It is described in the section "Set Depth."
Shape to Fit	Reshapes the FileBrowser window so that all the attributes in the display are visible at once, eliminating the need to horizontally scroll the window to get at all the information.

New Info

FileBrowser displays some file attributes, or information about the file, alongside each file in the browser display. Ordinarily, the attributes displayed are the size of the file in pages, its creation date, and its author. You can change the attributes displayed in a particular browser window by using the **New Info** command.

To use the **New Info** command, select it from the submenu of the **Recompute** command. FileBrowser opens up an additional subwindow, the "Info Options" window, below the display window. This subwindow contains a menu of attributes, with the current defaults shaded. Selecting a shaded item unshades it; selecting an unshaded item shades it. When you have selected all the attributes you wish to see displayed, issue either the **Recompute** or the **New Pattern** command. The files will be listed with the new information you requested. The "Info Options" window stays open—you can close it at any time.

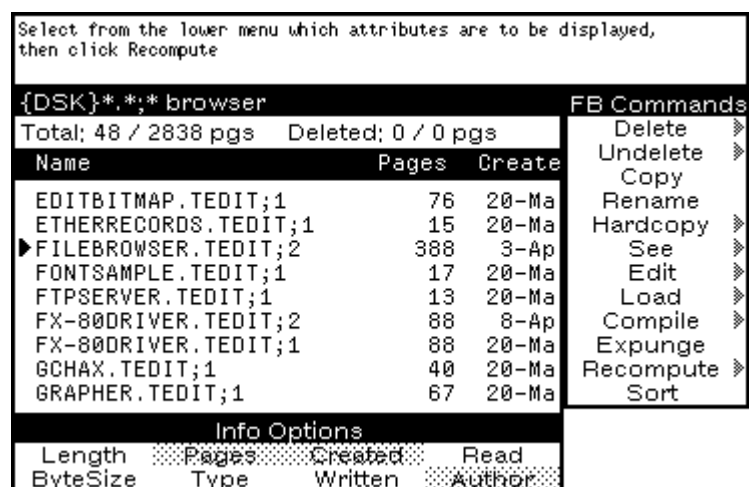


Figure 14-16. A FileBrowser showing the "Info Options" window.

The Info Options items have the following meanings:

Created	The date and time that the content of the file was created. This date changes whenever the file is modified, but does not change when a file is copied or renamed.
Written	The date and time the file was last written to the file system. This date is never older than the Created date, but it can be newer if the file is copied, unmodified, from one file system to another.
Read	The date and time the file was last read. This attribute may be blank if the file has never been read.
Author	The login name of the person who wrote the file, or last modified it.
Length	The length of the file in (usually eight-bit) bytes.
Pages	The number of 512-byte pages in the file. On some servers, this attribute is blank if the file is empty.
ByteSize	The size (in bits) of the bytes in the file.
Type	A value indicating what kind of data the file contains. The usual values of this attribute are TEXT, meaning the file contains just characters, or BINARY, meaning the file contains arbitrary data. Some servers have additional types, such as INTERPRESS for files in Interpress format.

Set Depth

XNS file servers support a feature that allows enumerating a directory to a user-specifiable depth. The "depth" of a file reflects the number of subdirectories between it and the root of the enumeration, i.e., the directory or subdirectory you gave in the pattern to FileBrowser, not counting any containing wildcards (asterisks). The immediate descendants of the root are at depth 1, files in subdirectories of depth 1 are at depth 2, and so on.

Ordinarily, FileBrowser enumerates a directory to the default depth, which is usually unlimited. To enumerate a directory to a different default, use the FB command with argument :DEPTH *n*, for some positive integer *n*, or T for unlimited depth. To change the depth in an existing FileBrowser, use the **Set Depth** command, a subcommand under the **Recompute** command. The command offers you a menu of choices:



Figure 14-17. The **Set Depth** menu.

Global default means use the default depth, overriding the depth at which this browser was last enumerated. **Infinite** means use no depth limit (same as depth T). **1** and **2** are common depth choices;

to choose some other numeric value, select **Other** and enter the value via the displayed keypad.

The **Set Depth** command does not affect the current display. It takes effect the next time you use the **Recompute** or **FileBrowse** commands from the same browser.

During a **Recompute**, if a subdirectory appears at the specified maximum depth, its descendants are not enumerated; rather, the subdirectory itself appears as an entry in the browser display. This entry can be selected, just like a file, but only a small number of commands can be used on it: you can **Rename** it, you can **Delete** it if it has no descendants, and you can **FileBrowse** it. It has attributes, just as ordinary files do. Its page size is the size of the entire subtree rooted at the subdirectory.

Note: Depth currently affects only XNS servers; all other devices ignore it and enumerate to their own default depths. In addition, due to a bug in XNS Services 10, depth is ignored for nontrivial patterns, i.e., anything but `"*.*"`.

Sort

The **Sort** command allows you to sort the files in the browser by any attribute of the files displayed in the browser. Selecting **Sort** brings up a menu of attributes by which to sort. This menu includes all the attributes currently displayed in the browser (such as Creation Date, Author), plus the choice Name. For some attributes you can sort forward or backwards; the choice is on a submenu, and the default is generally in the order of numerically greatest (e.g., size) or most recent (e.g., creation date) first.

If the attribute you select is not Name, then the file names displayed in the browser will be reformatted to include their directory portion (if there are any subdirectories below the browser's main pattern), as the subdirectory information is no longer implicit in a file's position in the browser.

The sort order Name, Decreasing Version is the default order in which browsers initially are created.

Troubleshooting Problems with FileBrowser

When FileBrowser returns the message: No files in group *FILENAMEPATTERN* when you know those files exist, the file server is probably down or rejecting connections. If this is so, your only option is to wait until the server is functioning again, and then give the **Recompute** command. In the case of an Xerox file server, the enumeration of files can also fail if you do not have sufficient access privileges; this condition is usually noted by a message in the system prompt window.

When you try to expunge a file and FileBrowser displays the message: Can't expunge *FILENAME*, it may be because you

don't have write access to the file, or someone else is reading the file. However, the most common reason is that the file is still open. Be sure to close any TEdit windows in which you may still be viewing the file. If you have recently issued a **Hardcopy** command for the file, a background process may still be working on the file.

[This page intentionally left blank]

Digital Clock

Introduction

The digital clock allows you to keep track of the time in multiple time zones. The clock updates itself once a minute. Clicking the left mouse button inside any of the clock windows will cause the digital clock to update itself.

17 Feb 89 11:39am (Fri)	
Washington DC:	17 Feb 89 2:39pm (Fri)
Tokyo, Japan:	18 Feb 89 5:39am (Sat)

Figure 15-1. The Digital Clock with two auxilliary regional time-zone windows showing the time in Washington and Tokyo.

Starting the Digital Clock

To put a digital clock on the screen, bring up the background menu by holding down the right mouse button, position the mouse over the **Clocks** option and slide off to the right to bring up the Clocks submenu. Slide the mouse cursor over the **Digital Clock** choice and release the right mouse button.



Figure 15-2. The **Digital Clock** option from the **Clocks** submenu from the background menu.

Selecting the digital clock option will delete any existing digital clock and start a new one.

When the clock comes up, it will prompt you for an auxilliary time zone. If you click outside the "Enter Time Zone" menu, the clock will come up with just the local time zone.



Figure 15-3. The "Enter Time Zone" menu.

Stopping the Digital Clock

To remove the clock from your screen and kill the clock process, simply close the clock window using the clock window's right mouse-button **Close** option.

Changing the Digital Clock

You can specialize the digital clock to suit your needs. You modify the digital clock by holding the middle mouse button down in the clock window. Holding this mouse button down in the main clock window brings up the menu shown in Figure 15-4. Holding the middle mouse button down in one of the regional time-zone windows brings up the menu shown in Figure 15-5.



Figure 15-4. The main-window middle-button menu.



Figure 15-5. The auxilliary-window middle-button menu.

Set Font

Brings up a series of menus which allows you to change the font in the main clock window.



Figure 15-6. The font menu.

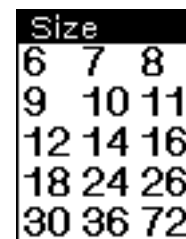


Figure 15-7. The font size menu.



Figure 15-8. The font face menu.

Select each option you want in sequence. If you want to keep any particular aspect of a font, simply click outside that particular menu. Clicking outside all the menus, leaves the font exactly the way it was to begin with.

While the system is searching for a font in the font files, it displays the message, "Fetching Font."

Not all possible combinations of every font exist. When the clock cannot find a particular font, it displays the message "Font Not Found."

Set Time

Brings up a menu which allows you to set the time.

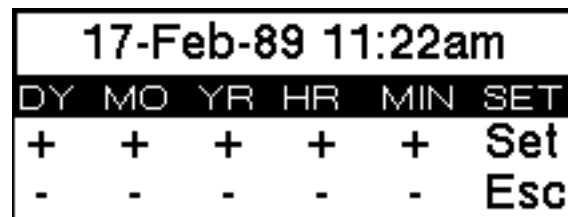


Figure 15-9. The set-time menu.

Each click on a + advances the counter for that category; each click on a - decreases the counter for that category. Selecting the **Set** option sets the time and closes the menu. Selecting **Esc** (Escape) closes the menu without setting the time.

Set Alarm

Brings up the set-time menu shown in Figure 15-9. The menu functions the same way it does for the **Set Time** option. After you have set the time, the clock will ask you for a message to associate with the alarm time. Type in your message followed by a carriage return.



Figure 15-10. The Digital Clock with the auxilliary window prompting for a message to associate with the alarm time.

Quiet Alarm/Loud Alarm

Toggles the clock back and forth between, a loud/auditory alarm and a quiet/visual alarm.



Figure 15-11. The main window middle-button menu with the **Loud Alarm** option showing instead of the **Quiet Alarm** option.

The loud alarm causes the monitor to beep once a minute when the alarm is ringing. The quiet alarm causes the screen to flash once a minute when the alarm is ringing.

Delete Alarm Setting

Turns the alarm off when it is ringing. This option, shown in Figure 15-11, only appears when the alarm is set.

Shape to Fit

Resizes the window to the minimum size necessary for the font you are using.

The window should resize automatically whenever you change the font. If ever the window and font size get out of synch, you can force the window to resize itself by selecting this option.

12-Hour Clock/24-Hour Clock

Toggles the digital clock back and forth between 12- and 24-hour modes. Compare Figures 15-4 and 15-11 to see how these options appear in the menu.

Set Local Time Zone

Allows you to change the time zone shown in the main digital clock window. Selecting this option brings up the "Enter Time Zone" menu, shown in Figure 15-3, from which you can choose the appropriate time zone. Clicking outside this menu leaves the time zone unchanged.

Add New Regional Time Zone

Allows you to add an additional regional time zone to the digital clock display. Selecting this option opens a new auxiliary window on the bottom of the clock and then pops up the "Enter Time Zone"

menu, shown in Figure 15-3, for you to select a time zone for the auxilliary window.

Delete This Window

Deletes the auxilliary window the mouse cursor was in when you pressed the middle mouse button to bring up the auxilliary-window menu.

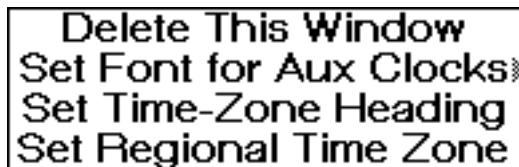


Figure 15-12. The auxilliary-window menu.

Set Font for Aux Clocks

Changes the font for all the auxilliary clocks indepent of the main clock. Selecting the option brings up the series of menus shown in Figures 15-6, 15-7, and 15-8. Clicking outside of any menu leaves that aspect of the font the way it was originally.

Set Aux Clock Font In Just This Window

Changes the font for the auxilliary window the mouse cursor was in when you pressed the middle mouse button to bring up the auxilliary-window menu. Selecting the option brings up the series of menus shown in Figures 15-6, 15-7, and 15-8. Clicking outside of any menu leaves that aspect of the font the way it was originally.



Figure 15-13. The submenu for setting the font in just one auxilliary window.

Set Time-Zone Heading

Prompts you for a new heading for the auxilliary window you were in when you pressed the middle mouse button to bring up the auxilliary-window menu. Type in the name followed by a carriage return.

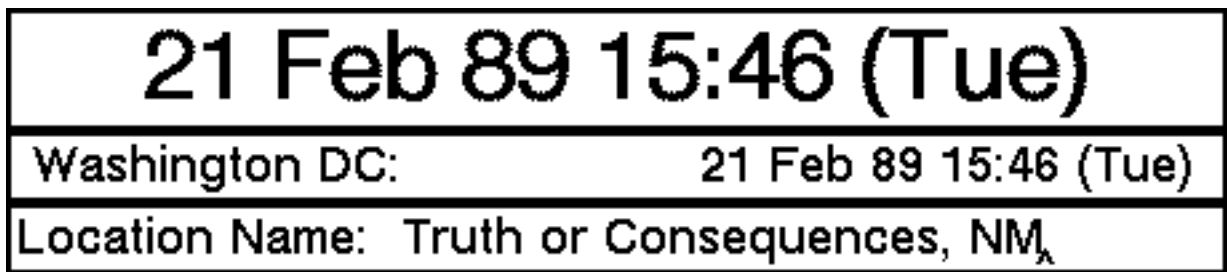


Figure 15-14. An auxilliary window prompting for a new heading.

Occasionally, the window isn't wide enough to display the whole name. In this case choose the option **Shape to Fit** from the main window middle-button menu.

Set Regional Time Zone

Brings up the "Enter Time Zone" menu shown in Figure 15-3, which allows you to change the time zone region. **Set Regional Time Zone** does not allow other than hour increments.

Analog Clock

Introduction

The **Analog Clock** suboption off the **Clock** option on the background menu sets up an analog clock on your screen. The clock is updated once a minute.



Figure 15-15 The analog clock.

Starting the Analog Clock

To put an analog clock on the screen, bring up the background menu by holding down the right mouse button, position the mouse over the **Clocks** option and slide off to the right to bring up the Clock submenu. Slide the mouse cursor over the **Analog Clock** choice and release the right mouse button.



Figure 15-16. The Analog Clock option from the Clock submenu from the background menu.

The first time you bring the analog clock up in a sysout, it will prompt you to sweep out a window. Once you have done this this clock will always appear in its last size. If you want to resize the analog clock, choose the **Shape** option from the right button window menu, and sweep out the new shape for the clock.

Stopping the Analog Clock

To remove the clock from your screen and kill the clock process, simply close the clock window using the clock window's right mouse-button **Close** option.

Changing the Analog Clock

Holding the middle mouse button down inside the analog clock pops up the following menu.



Figure 15-17. The analog clock menu.

There are four independent properties which the user may control: the hands of the clock, time digits printed where the hands end, rings on the clock face, and 12 numbers around the outside of the clock face.

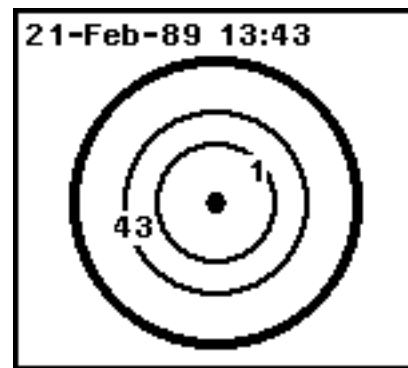


Figure 15-18. An analog clock with no points or numbers, rings, no hands, and times.

NUMBERS

Removes points and adds numbers to the clock face.

POINTS

Removes numbers and adds points to the clock face.

NO.NUMBERS

Removes both numbers and points from the clock face.

RINGS

Adds rings to the clock face.

NO.RINGS

Removes rings from the clock face.

HANDS

Adds hands to the clock face.

NO.HANDS

Removes hands from the clock face.

TIMES

Adds time digits to the clock face.

NO.TIMES

Removes time digits from the clock face.

SHOW.STYLE

Prints the current style to the system prompt window.

SET.TO.DEFAULT

Resets the clock face to the default settings.

CHANGE.DEFAULT

Sets the default to the current style.

SETTIME

Forces the clock to go to the machine to ask it the time.

Known Problems

The clock will occasionally not respond to style changes. To force it to change, close the clock window and restart the clock from the background menu.

Directory Connector

The directory connector allows you to keep track of your currently connected directory and allows you to change the directory. It updates itself about once every ten seconds.



Dir: {DSK}/usr/users/kmount/nc/nf/

Figure 15-19. The directory connector window.

Starting the Directory Connector

To put a directory connector on the screen, bring up the background menu by holding down the right mouse button, position the mouse over the **Directory Connector** option and release the right mouse button.

Stopping the Directory Connector

To remove the directory connector from your screen and kill its process, simply close the directory connector window using the window's right mouse-button **Close** option.

Changing the Directory Connector Fonts

To change the directory connector font, bring up the background menu by holding down the right mouse button, position the mouse over the **Directory Connector** option, slide off the menu to the right, position the mouse over the **Change Font** option, and release the right mouse button.



Directory Connector » **Change Font**

Figure 15-20. The **Directory Connector** option on the background menu showing the **Change Font** submenu option.

When you select the Change Font option you will be presented with a series of two menus asking you for a font and a size.



Figure 15-21. The font menu.

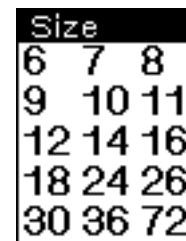


Figure 15-22. The font size menu.

While the directory connector is looking for a font, it prints the message "Fetching Font..." to the system prompt window.

Not all possible combinations of fonts are possible. If a font is not found the directory connector prints the message "Font Not Found." in the system prompt window.

Using the Directory Connector

The directory connector can be used in two ways.

Left Mouse Button

Clicking the left mouse button in the directory connector window causes the window to update itself.

You can shift-select the connected directory name out of the directory connector window with the left mouse button. You do this by pointing to the position in the window where you want the name of the directory to appear and clicking the left mouse button to position the caret cursor there, a FileBrowser prompt window for instance. Next you hold a SHIFT key down and click the left mouse button in the directory connector window. The name of the directory will appear where you positioned the caret cursor.

Middle Mouse Button

Clicking the middle mouse button in the directory connector window brings up a menu of directories to connect to.



Figure 15-23. The "New Directory?" menu.

If the directory you want to connect to is on this list, click the left mouse button on it to connect to it.

If you change your mind and want to abort the operation, click outside the "New Directory?" menu.

If the directory you want to connect to is not on this list, click the left mouse button on * * Connect to Other Directory * * and type the directory name into the prompt window which will appear.

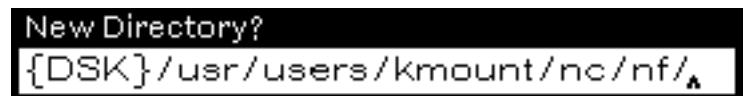


Figure 15-24. The "New Directory?" prompt window.

The prompt window will come up with your currently connected directory. The moment you start typing the system will erase the window and show what you are typing.

If you want to modify the name of your currently connected directory, backspace over as much of the name as you need to and type the name of the new subdirectory. Be sure to end it with a slash.

If you type in an invalid directory name, directory connector will beep and print the message "Not a valid directory name." to the system prompt window.

[This page intentionally left blank]

NoteCards includes facilities for printing in the PostScript language by Adobe, and the Interpress, language by Xerox.

This chapter explains:

How to setup your printers.

How to specify where your font directories are.

How to deal with some common printer problems.

How to customize your postscript printer.

Setting Up Printers

You can set up printers from either the background menu or the initialization file. For how to set up your printers with the initialization file, see Appendix C.

Adding Default Printers

To set up a new default printer from the background menu, press and hold the right mouse button in the background, place the mouse cursor on **Set Default Printer** then slide off to the right and select **Add Default Printer**.

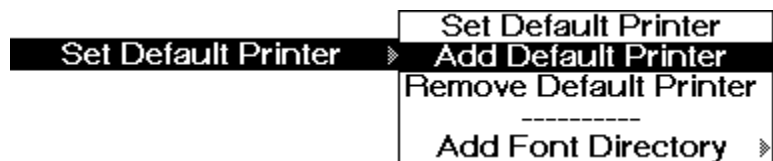


Figure 16-1. The **Set Default Printer** background menu option displaying its submenu.

A prompt window will appear showing you the name of the current default printer and asking for the name of a new default printer.

Figure 16-2. A PostScript printer name being entered as a new default printer.

Figure 16-3. An Interpress printer name being entered as a new default printer.

Enter the name of the new default printer. The moment you type the first character of the new printer name, the old name will be deleted and replaced with what you type. If you don't want to make any changes, strike the carriage return key. If you want to modify the name that is there, you can backspace over the name to erase

part of it. If you type a carriage return without a entering printer name, this operation is aborted.

When you type a carriage return to end the name, a menu will pop up requesting the printer type. Select one of the options with the left mouse button.



Figure 16-4. The system prompting you for the printer type.

If you do not select any option (by clicking outside the menu) and the printer's type was previously defined, it will remain the same. If the printer's type was not previously defined, and you do not select any option, the printer's type will be left undefined which will cause problems when you try to print. If you leave the printer type undefined the system will warn you.



Figure 16-5. The system prompt window giving the new printer status and warning you that the printer type for Oahu is undefined.



Figure 16-6. The system prompt window giving the new printer status.

At this point, if you have already specified the font directories, you should be able to print to any default printer.

Setting the Default Printer

The system maintains a list of printers, one of which is the default printer--the printer that files are sent to when no printer is explicitly specified.

To set the default printer, select the **Set Default Printer** option from the background menu.



Figure 16-7. The **Set Default Printer** option on the background menu.

This will cause the system to open a menu listing all the known printers.



Figure 16-8. The "Choose Default Printer." menu

Selecting one of these options will make that printer the default printer. The current default printer is the topmost option.

Once you have selected a new default printer, the system will confirm your choice by printing the printer name and type to the system prompt window.



Figure 16-9. The system prompt window giving the new printer status.

If the printer type is undefined the system beeps and issues a warning.



Figure 16-10. The system prompt window giving the new printer status and warning you that the printer type for Oahu is undefined.

Removing a Default Printer

If you make a mistake specifying a printer or want to remove a printer for some reason, you can use the **Remove Default Printer** submenu option on the background menu.

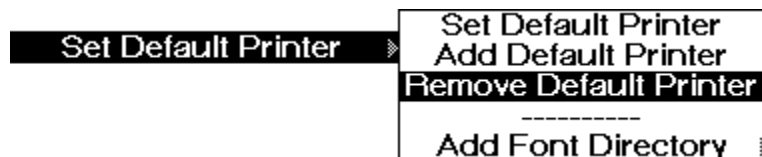


Figure 16-11. The **Remove Default Printer** submenu option off the background menu.

Selecting this option brings up a menu of known printers. Selecting one of these deletes it from the known-printers list and makes the topmost remaining printer the default printer.



Figure 16-12. The "Choose Printer to Remove." menu.

Setting Font Directories

As with printers, it is possible to set your font directories from either the background menu or your initialization file. We recommend that you set your directories from the initialization file. There are two reasons for this. Setting the font directories involves a lot of typing, and these directories are set correctly for you by the standard initialization file unless you have moved the font files. See Appendix C for how to set the font directories from the initialization file.

Adding a New Font Directory

To add a font directory from the background menu, press and hold the right mouse button in the background to bring up the background menu. Position the mouse cursor on the **Set Default Printer** option and slide off to bring up the submenu containing **Add Font Directory**.



Figure 16-13. The **Add Font Directory** submenu options on the background menu.

Selecting **Add Font Directory** brings up a menu asking you to specify the font group you want to add a search directory to.



Figure 16-14. The "Add to which directory group?" menu.

The NoteCards system supports one display and three hard copy modes. For each of these there is a specialized set of fonts. The fonts for the screen display are stored in the display font directories, the fonts for PostScript printers are stored in the PostScript font directories, those for Interpress printers are stored in the Interpress font directories, etc. When you add a new directory to one of these groups, you are providing the system with another place to search for fonts. For example, the display font directories typically include the following directories:

```
{DSK}/usr/local/lde/fonts/display/presentation/
{DSK}/usr/local/lde/fonts/display/presentation/
{DSK}/usr/local/lde/fonts/display/publishing/
{DSK}/usr/local/lde/fonts/display/printwheel/
{DSK}/usr/local/lde/fonts/display/miscellaneous/
```

When the system is looking for a new display font, it will search through each of these directories in sequence for the font. The fonts are broken up into separate directories for convenience and efficiency reasons.

Once you have specified the font directory group you want to work with, the system will prompt you for the name of a new directory to search for fonts of that type.

Figure 16-15. The system prompting for a new font directory to search for Interpress fonts.

Enter the name of the new font directory. The moment you type the first character of the font directory name, the old name will be deleted and replaced with what you type. If you don't want to make any changes, strike the carriage return key. If you want to modify the name that is there, you can backspace over the name to erase part of it. If you type a carriage return without a printer name, this operation is aborted.

Note that for Sun machines, the directory name will always start with {DSK}, that subdirectories in the path name are separated by slashes, and that the directory path name must end in a slash (/).

Showing the Font Directories

In order to view a particular set of font directories, use the **Show Font Directories** submenu option shown in Figure 16-13.

Like **Add Font Directory**, this first brings up menu asking you for the directory group you want to show.



Figure 16-16. The "Show which directory group?" menu.

Once you have selected the directory group, the system pops up a window which displays the font directories in that group. You can close this window using the standard window-menu close option or the window will close of its own accord after about two minutes.



Figure 16-17. A window showing the list of display font directories.

Removing a Font Directory

If you make a mistake entering a font directory name or for some other reason want to remove a font directory, use the **Remove Font Directory** option shown in Figure 16-13.

Like the **Add Font Directory** this first brings up menu asking you for the directory group you want to modify.

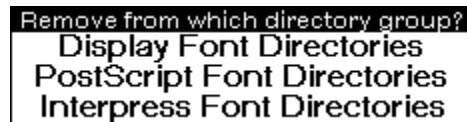


Figure 16-18. The "Remove from which directory group?" menu.

Once you have selected the directory group, the system pops up a menu of all the directories currently in that group.



Figure 16-19. The "Remove which directory?" menu.

Selecting one of these options deletes that directory. Clicking outside the menu aborts this operation.

Trouble Shooting Common Printer Problems

This section discusses some of the more common breaks which occur when your printers are not properly set up.

Can't determine IMAGETYPE for {LPT}

This break occurs when you do not have any printers specified on the default printer list, or if you have not specified the printer type.

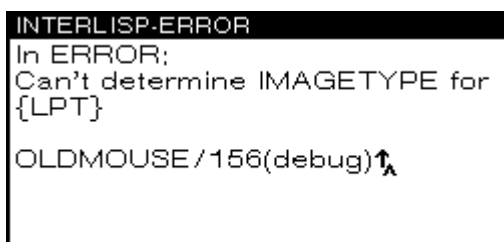


Figure 16-20. A break window indicating that the system cannot find a valid printer.

Type an up-arrow to exit the break and make sure that the printer and its type are properly set.

You can get into this situation by choosing the hardcopy **To a printer** suboption off the standard window menu and then selecting the **Other...** option from the "Which Printer?" menu.

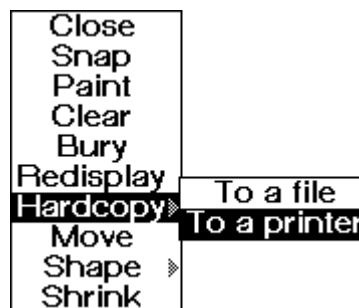


Figure 16-21. The standard right button window menu, showing the hardcopy **To a printer** submenu option selected.

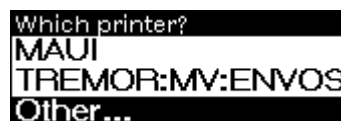
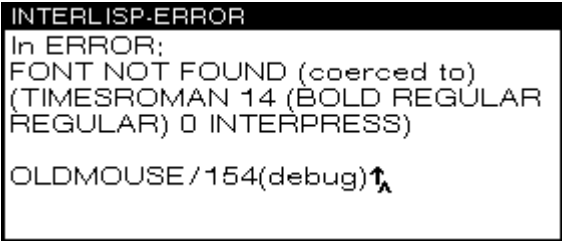


Figure 16-22. The "Which printer?" menu showing the **Other...** option selected.

The **Other...** option does not automatically prompt you for a printer type. Therefore, we recommend that you always add new printers using the background menu **Add Default Printer** submenu option.

Font not Found

This break occurs when you do not have one of your font directories set properly.



```
INTERLISP-ERROR
In ERROR:
FONT NOT FOUND (coerced to)
(TIMESROMAN 14 (BOLD REGULAR
REGULAR) 0 INTERPRESS)

OLDMOUSE/154(debug)↑
```

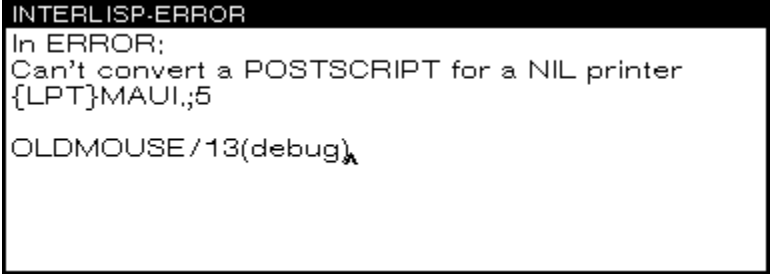
Figure 16-21. A break window indicating that the system cannot find the appropriate font.

In this particular case, the system was unable to find an Interpress font. Note that the font description contains the word Interpress. This implies that either there are no Interpress font directories set or that the given Interpress font directories do not contain that font.

Type an up-arrow to exit the break and make sure that the appropriate font directory is properly set. You can do this by selecting the **Show Font Directories** suboption off the **Set Default Printer** option on the background menu. If directories are not correctly set use the **Add Font Directory** and **Remove Font Directory** options to correct the errors.

Can't Convert a POSTSCRIPT for a NIL printer

This break can occur if you have incorrectly typed the case of the printer name.



```
INTERLISP-ERROR
In ERROR:
Can't convert a POSTSCRIPT for a NIL printer
{LPT}MAUI.;5

OLDMOUSE/13(debug)\
```

Figure 16-22. A break window indicating that the system cannot find the appropriate printer.

Type an up-arrow to exit the break and if you have entered a printer name in lower case, try entering the same name in upper case by selecting the **Add Default Printer** option off the **Set Default Printer** submenu on the background menu.

Customizing Your PostScript Printer

The PostScript driver allows you to set a number of parameters to modify the output to suit your needs.

Portrait vs. Landscape

There are two terms used to describe how an image is printed on a piece of paper which you need to know.

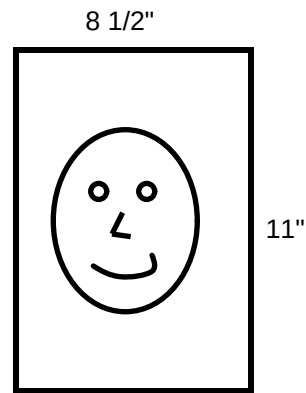


Figure 16-23. Portrait describes the way we normally hold a piece of paper.

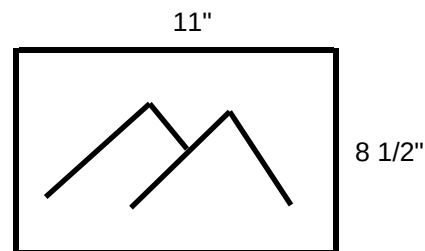


Figure 16-24. Landscape describes a piece of paper held on its side.

Maximum Wild Font Size

Indicates the maximum point size that the system will indicate as available when looking for fonts.

The default value is "72."

Font Association Menu

Maps display font names onto PostScript font names. To change the mapping click the left mouse button on a display font name in the left-hand column then select a PostScript font from the a menu of PostScript fonts names which will appear.

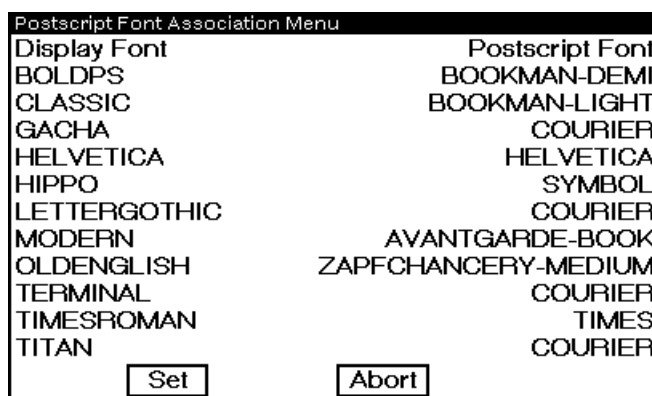


Figure 16-25. The "Postscript Font Association Menu" showing the display fonts on the left and the PostScript printer fonts on the right.

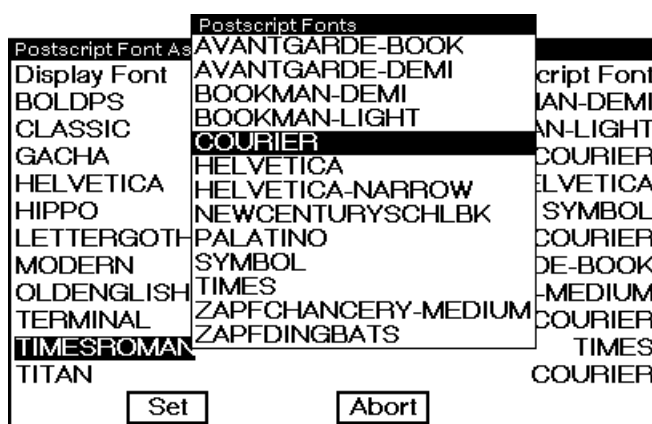


Figure 16-26. The "Postscript Font Association Menu" with the TIMESROMAN display font selected and about to be set to correspond to the COURIER PostScript printer font.

PostScript Short Edge Shift

Is the distance (in points) to shift the image perpendicular to the short edge of the paper. A positive value gives a shift upward in portrait mode, and to the right in landscape mode. A negative value shifts the image down in portrait and to the left in landscaped mode.

The default value is "0."

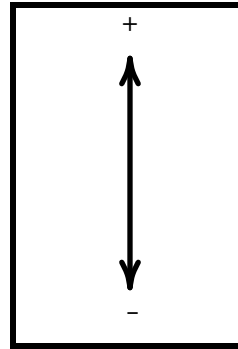


Figure 16-27. The short edge shift in portrait mode.

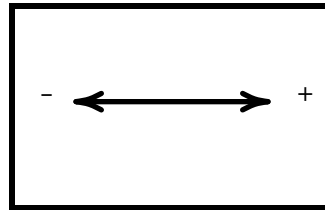


Figure 16-28. The short edge shift in landscape mode.

PostScript Short Edge Points

Indicates the printable region of the page, in points, along the short edge of the paper. It should be adjusted to allow for any short edge shifts of the image.

The default value is "576" points or 8 inches.

PostScript Long Edge Shift

Is the corresponding variable for shifts perpendicular to the long edge of the paper. A positive value here gives a shift to the right in portrait mode and downward in landscape mode.

The default value is "0."

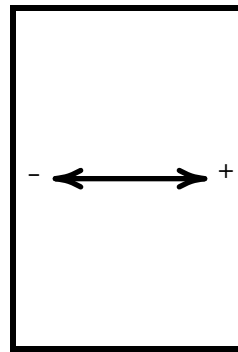


Figure 16-29. The long edge shift in portrait mode.

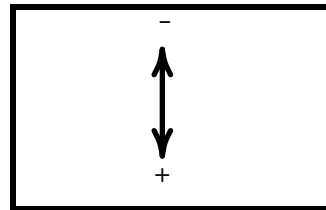


Figure 16-30. The long edge shift in landscape mode.

PostScript Long Edge Points

Indicates the printable region of the page, in points, along the long edge of the paper. It should be adjusted to allow for any long edge shifts of the image.

The default value is "786.24" or "10.92" inches.

Note:

The AST TurboLaser PS has an imageable area on the page which is a different size than that of the Apple LaserWriter. The values of **PostScript Short Edge Points** and **PostScript Long Edge Points** for the AST are "575.76" and "767.76," respectively.

Landscape Mode

Indicates whether the default orientation of output files is landscape.

The default value is "No."

Landscape Text Mode

Indicates if the printing of text files should force the orientation of output files to landscape.

The default value is "No."

Bitmap Scale

Specifies an independent scale factor for display of bit map images, window hardcopies, for example. Values less than 1 reduce the image size, i.e. a value of 0.5 will give a half size bit map image. The position of the scaled bit map will still have the same lower-left

corner, i.e., the scaled bit map is not centered in the region of the full size bit map image.

The default value is "1."

Note:

Setting bit map scale to 0.96, instead of 1, will give cleaner bit map images on a 300 dpi printer. This corrects for the 72 ppi imagestream vs. the 75 dpi printer, using 4x4 device dots per bit map pixel. Also, values of 0.24, 0.48 and 0.72, instead of 0.25, 0.5 and 0.75, will also give cleaner images for reduced size output. In general, use integer multiples of 0.24 for a 300 dpi printer.

Texture Scale

Specifies an independent scale for the display of bit map textures. The value represents the number of printer bits per screen pixel. The default value is 4, which represents each pixel of the texture as a 4x4 block, so that textures are approximately the same resolution as on the screen for 300 dpi output devices, such as the Apple Laserwriter.

The PostScript package extends the allowed representations of a texture, beyond 16-bit integers and 16x16 bit maps, to any square bit map. (If the bit map is not square, its longer edge is truncated from the top or right to make it square.) Use this feature with caution, as large bit map textures, or sizes other than multiples of 16 bits square, require large amounts of storage in the PostScript interpreter, and can cause errors when printing.

Image Size Factor

Specifies an independent factor to change the overall size of the printed image. This resizing affects the entire printed output. Values greater than 1 enlarge the printed image, and values less than 1 reduce it. The **Bitmap Scale** parameter does not consider the **Image Size Factor** when determining the scale factor for a bit map.

[This page intentionally left blank]

17. ERROR RECOVERY AND KNOWN PROBLEMS

NoteCards on the Sun Workstation has an error handling system which includes

- the break window error system,
- a diagnostic program, URAID, which handles emulator errors.

Occasionally, you may encounter SunOS error messages. Refer to your Sun documentation set for recovery procedures when these errors occur.

Known problems with the NoteCards system are discussed at the end of this chapter.

System Status, Aborting Operations, and Spawning a New Mouse

NoteCards provides a number of keyboard combinations for enquiring about system status and aborting operations.

System Status

Typing CONTROL-T prints status information on the type-in process. The type-in process is the process which has the flashing caret cursor. This is of minimal usefulness as the type-in process is usually just sitting waiting for you to type characters in, and the messages are aimed at people extending the NoteCards system. However, if you need to know if the system is still active, typing CONTROL-T will force the system to give you the status on one process.

Aborting Operations

Hitting the STOP key will frequently abort the process you want it to. However, as the system runs multiple processes the process which receives the stop command may not be the one you intended to receive it .

Typing CONTROL-G gives you a list of all the currently running processes, in menu format, as shown in Figure 17-1.


```

Interrupt which process?
[Spawn Mouse]
OLDMOUSE *run
TEdit *tty
MOUSE
TEdit#3
DIGI-CLOCK
DIRECTORY CONNECTOR
TEDIT-KILLER#2
ERIS#LEAF
\10MBWATCHER#2
\NSGATELISTENER
\PUPGATELISTENER
\TIMER.PROCESS
BACKGROUND

```

Figure 17-1. The "Interrupt which process?" menu showing all the currently running processes.

Selecting one of these processes will cause a break to occur in that process. You can then uparrow, ^, out of the break to abort the process. If you decide you have broken the wrong process, you can type **ok** to have the process resume where it left off.

```

INTERRUPT
Interrupt below \TEDIT,HARDCOPY,DISPLAYLINE,
OLDMOUSE/118(debug)ok

```

Figure 17-2. A break caused by a CONTROL-G. The OLDMOUSE process is being resumed with the **ok** command.

Because there can be so many processes, TEdit processes in particular (there is one TEdit process for every TEdit window including the expanded-menu command windows), it is important to know which process you want to break to abort an operation. Generally, if you start an operation with the mouse, it either runs under the MOUSE or OLDMOUSE process, which is deleted the moment the operation finishes. If you see an OLDMOUSE process, this is usually the process you want to break.

There are exceptions to the OLDMOUSE rule. Performing operations from the FileBrowser causes a process with an "FB-" prefix to be spawned. All operations on the TEdit expanded menus are run in the main TEdit process associated with that edit window. If you have one TEdit window open on your screen and one TEdit expanded menu open, you will have two TEdit processes running: TEdit, which is the process associated with the main TEdit window, and TEdit#2, which is the process associated with the expanded menu.

***Killing the wrong process can cause you to lose information.
USE WITH CAUTION!***

Spawning a New Mouse

If by some chance you do manage to kill your mouse process off, it is easy to start a new one. Type CONTROL-G and choose the **[Spawn Mouse]** option on the "Interrupt which process?" menu. **[Spawn Mouse]** will always be the very top option. If you create several new mouse processes, the system will remove all but one mouse process.

Break Windows

Occasionally, when performing certain operations like accessing a file which has been moved to another directory, deleted, or renamed, or trying to create a font which does not exist, NoteCards will enter a break. When this happens, NoteCards provides you with information about what caused the break, and asks you what to do.



Figure 17-3. A break window caused by a font-not-found error.

In cases like this, you should first determine if the cause of the break is a simple error. Here for instance, there is no Helvetica 33 bold font. There is, however, a Helvetica 32. The appropriate action in this case would be to up arrow out of the break window by typing a ^, and reexecuting the operation with the correct font.

Another possibility is that the font directories may have been moved and the system may be looking in the wrong place for the fonts. Here again, you would up arrow out of the break window by typing a ^, edit the initialization file (see Appendix C, Initialization Files) to insert the correct values for DISPLAYFONTDIRECTORIES, load the initialization file, and then retry the operation.

If the break is caused by a missing file, you should up arrow out of the break window by typing a ^, find the current location of the file, and retry the operation with the new path and file name.

If you are unable to uparrow out of a break, or the system is behaving strangely, you should save all your work by closing your notefiles, and saving your TEdit and Sketch files, perform a **Flush**

image & Exit, and reload your system. **Flush image & Exit** is a submenu off **Logout** on the background menu.

Errors While Running NoteCards

The following errors may occur running NoteCards on the Sun Workstation.

ERROR MESSAGE	REASON/FUNCTION RESPONSIBLE
File access timed out	Occurs when you try to access a file on a remotely mounted file system or NFS service that is down.
File too large	Self explanatory.
Too-Many-Files-Open	Occurs when 1) you exceed the SunOS open file limit (see Chapter 4, System Use Issues) 2) you exceed system file resources while writing a sysout (using the Logout command from the background menu.)
Nonexistent directory	Occurs when user tries to connect to a nonexistent directory.
No-Such-Directory	Occurs when user tries to connect to a nonexistent directory.
Connection timed out	Self explanatory.
Bad Host Name	Self explanatory.

I/O Errors

These Xerox workstation-specific errors may occur if certain functions are inadvertently used on the Sun Workstation.

ERROR MESSAGE	REASON/FUNCTION RESPONSIBLE
Floppy: No floppy drive on this machine.	Self-explanatory.
Device error: {FLOPPY}	Occurs when you try to access a floppy device while running on the Sun Workstation.
Wrong machinetype	Occurs when functions controlling Xerox disk drive device-specific behavior are entered when running on a Sun.

Virtual Memory Errors

ERROR MESSAGE	REASON/FUNCTION RESPONSIBLE
File-System-Resources-Exceeded	Logout, Save image & Exit, Save VM
Protection-Violation	Logout, Save image & Exit, Save VM
File-Wont-Open	Logout, Save image & Exit, Save VM

URAI

The NoteCards system normally operates as a self-contained environment. In some unusual circumstances NoteCards may encounter a situation from which it cannot recover. In this case, when an unrecoverable emulator error is encountered, the emulator halts and enters into a small debugger called URAID. URAID allows you to inspect memory, or to look inside the sysout file, and attempt to recover from the error.

If you produce the same type of error condition in NoteCards on a Xerox workstation as you did on a Sun Workstation, you get an MP error instead of a URAID error.

Entering URAID

Normally, the emulator automatically enters URAID when an unrecoverable emulator error occurs. If the system freezes and will not let you regain control, you can throw the system into URAID by simultaneously holding down the SHIFT, CONTROL and DELETE (L10) keys or the SHIFT, CONTROL, and NEXT (ALTERNATE) keys.

URAI Commands

URAI has a few simple commands which you can use to attempt error recovery. All URAID commands are case sensitive.

- h** Hard Reset. Attempts to recover by resetting the Lisp stack. Quits URAID and causes NoteCards to resume execution. This command should not be used unless you are sure that execution can be resumed.
- e** Exit to SunOS. NoteCards will end. If you are going to call Customer Support, call *before* you give this command. After giving this command, nothing is recoverable.
- q** Quits URAID and returns to NoteCards.

Note: An error may occur while the NoteCards system is running uninterruptably. The following message signals this error:

Error in uninterruptable system code -- ^N
to continue into error handler

Disregard the ^N command; it is not supported by URAID.
Use the **q** command to continue.

Other Fatal Error Conditions

Occasionally, other emulator, operating system, or system administration errors may occur from which the URAID program cannot recover. Such error conditions include a process dying, the emulator going into an infinite loop, the keyboard being lost, or the system freezing up.

If any of these emulator errors occur, use the UNIX `kill` command to kill the `lde` process.

System Error Conditions

The following are error messages generated by SunOS. For complete information on these error messages, see the *SunOS Reference Manual*, Intro(2).

ERROR MESSAGE	DESCRIPTION
:0 Unused	
:1 EPERM	Not owner
:2 ENOENT	No such file or directory
:3 ESRCH	No such process
:4 EINTR	Interrupted system call
:5 EIO	I/O error
:6 ENXIO	No such device or address
:7 E2BIG	Arg list too long
:8 ENOEXEC	Exec format error
:9 EBADF	Bad file number
:10 ECHILD	No children
:11 EAGAIN	No more processes
:12 ENOMEM	Not enough core
:13 EACCES	Permission denied
:14 EFAULT	Bad address
:15 ENOTBLK	Block device required
:16 EBUSY	Mount device busy
:17 EEXIST	File exists
:18 EXDEV	Cross-device link
:19 ENODEV	No such device
:20 ENODIR	Not a directory
:21 EISDIR	Is a directory
:22 EINVAL	Invalid argument
:23 ENFILE	File table overflow

:24	EMFILE	Too many open files
:25	ENOTTY	Not a typewriter
:26	Unused	
:27	EFBIG	File too large
:28	ENOSPC	No space left on device
:29	ESPIPE	Illegal seek
:30	EROFS	Read-only file system
:31	EMLINK	Too many links
:32	EPIPE	Broken pipe
:33	EDOM	Math argument
:34	ERANGE	Result too large
:35	EWOULDBLOCK	Operation would block
:36	EINPROGRESS	Operation now in progress
:37	EALREADY	Operation already in progress
:38	ENOTSOCK	Socket operation on non-socket
:39	EDESTADDRREQ	Destination address required
:40	EMSGSIZE	Message too long
:41	EPROTOTYPE	Protocol wrong type for socket
:42	ENOPROTOPT	Bad protocol option
:43	EPROTONOSUPPORT	Protocol not supported
:44	ESOCKNOSUPPORT	Socket not supported
:45	EOPNOTSUPP	Operation not supported on socket
:46	EPFNOSUPPORT	Protocol family not supported
:47	EAFNOSUPPORT	Address family not supported by protocol family

ERROR MESSAGE**DESCRIPTION**

:48	EADDRINUSE	Address already in use
:49	EADDRNOTAVAIL	Can't assign requested address
:50	ENETDOWN	Network is down
:51	ENETUNREACH	Network is unreadable
:52	ENETRESET	Network dropped connection on reset
:53	ECONNABORTED	Software caused connection abort
:54	ECONNRESET	Connection reset by peer
:55	ENOBUFS	No buffer space available
:56	EISCONN	Socket is already connected
:57	ENOTCONN	Socket is not connected
:58	ESHUTDOWN	Can't send after socket shutdown
:59	Unused	
:60	ETIMEDOUT	Connection timed out
:61	ECONNREFUSED	Connection refused
:62	ELOOP	Too many levels of symbolic link
:63	ENAMETOOLONG	File name is too long
:64	EHOSTDOWN	Host is down
:65	EHOSTUNREACH	No route to host
:66	ENOTEMPTY	Directory not empty
:67	Unused	
:68	Unused	
:69	EDQUOT	Disc quota exceeded
:70	ESTALE	Stale NFS file handle
:71	EREMOTE	Too many levels of remote in path
:72	ENOSTR	Not a stream device
:73	ETIME	Timer expired
:74	ENOSR	Out of stream resources
:75	ENOMSG	No message of desired type
:76	EBADMSG	Not a data message

Known Problems

Printing Browsers

Printing Browsers which contain dashed lines may cause your printer to hang.

Notefile Indicator Window

Occasionally, after you have reshaped a card, a piece of the text in the notefile indicator window will be garbled or broken. To redisplay the text in this window, place the point of the mouse cursor in the notefile indicator window, hold down the right mouse button, and select the **Redisplay** option from the window menu which pops up.

Fonts

The PostScript Zapfchancery font does not work at this point in time.

TEdit

When a TEdit window is positioned such that part of the display region of the window is off the screen, TEdit will display some lines of text in a garbled fashion. Simply move the window so that the entire extent of the window is on the screen or redisplay the window using the **Redisplay** option on the title-bar right-button mouse menu.

[This page intentionally left blank]

This document provides a brief explanation of the structure of notefiles. It also describes how checkpointing, aborting, and recovering a notefile after a crash work. Finally, it describes the use of **Inspect & Repair** to repair a notefile with inconsistent links. For additional information see Appendix B, The Notefile Inspector.

Note: The information in this appendix was copied whole from the documentation for a prerelease of NoteCards and has not yet been updated for this release. As such, it may be inaccurate in places.

Background Concepts

Unique identifiers: UIDs

All objects in NoteCards (i.e., cards, links, notefiles) are assigned a unique identifier called a UID. Each UID is a 112-bit number that is guaranteed to be unique across all time and space. UIDs are used in many places in NoteCards as keys for indexing and retrieving cards, links, and notefiles.

Card parts

For storage purposes a note card is decomposed into 4 independent parts: contents, title, property list, and links. Each of these parts is stored separately in the data area of the notefile. This is discussed in the Notefile Structure section below. When a card is saved, only those card parts that have changed are rewritten in the notefile.

The contents of the card are stored on the notefile in a manner appropriate to its type. Thus a Text card's content is a text stream and is written on the notefile exactly the way TEdit writes out text streams (i.e., text followed by "looks" information). In contrast, all titles are stored as strings and all property lists as standard Lisp lists. Storage of Links is described in Section 4 below.

Notefile Structure: index and data areas

A notefile consists of three parts, a header, an index, and a data area. The header and the index are fixed in size for each notefile. The data area follows the index area and grows as cards are added to or modified in the notefile.

The notefile header contains the following information about the notefile: its UID, a number identifying its format, the checkpoint pointer, the size of the index, and a pointer to the next available index entry. If the notefile header is destroyed, it cannot be automatically reconstructed. Careful hand manipulation of the

notefile by a NoteCards wizard is required to recover a notefile with a bad header.

The structure of the index and data area is shown in Figure A-1 .

The index contains a fixed number of index entries. Each index entry that is in use, contains information for one of the cards in the notefile. Specifically, an index entry contains 5 fields: a status character, the card's UID, and 4 pointers. The status character specifies whether the index entry is free (not in use) or contains information for an active or a deleted card. If the index entry is not free, the UID field contains the UID of the card referred to by this index entry and the four pointer fields contain the location in the data area of the 4 parts of its card: contents, title, links and property list.

The number of index entries is fixed at notefile creation time. The default is 1000 entries. The number of index entries is automatically doubled by the notefile compactor if 75% of the entries are used. The compactor also frees (i.e., makes unused) all of the index entries that refer to deleted cards. In normal operation, NoteCards prints a warning whenever more than 90% of the index entries in a notefile are used. At this point, the notefile should be compacted to increase the index size.

The data area contains the actual information about the card. Whenever you change, say, the title of a card, the new title is written at the end of the data area. The title pointer in the card's index entry is then updated to point to this new location in the data area. Thus, in general, a notefile's data area grows every time any part of any card is changed.

The old information, now somewhere in the middle of the data area, is not removed. However, it is no longer directly accessible because there is no index entry that points to it. Thus, for most purposes this old information can be considered "dead space" in the notefile. The notefile compactor rewrites the notefile, eliminating all such dead space.

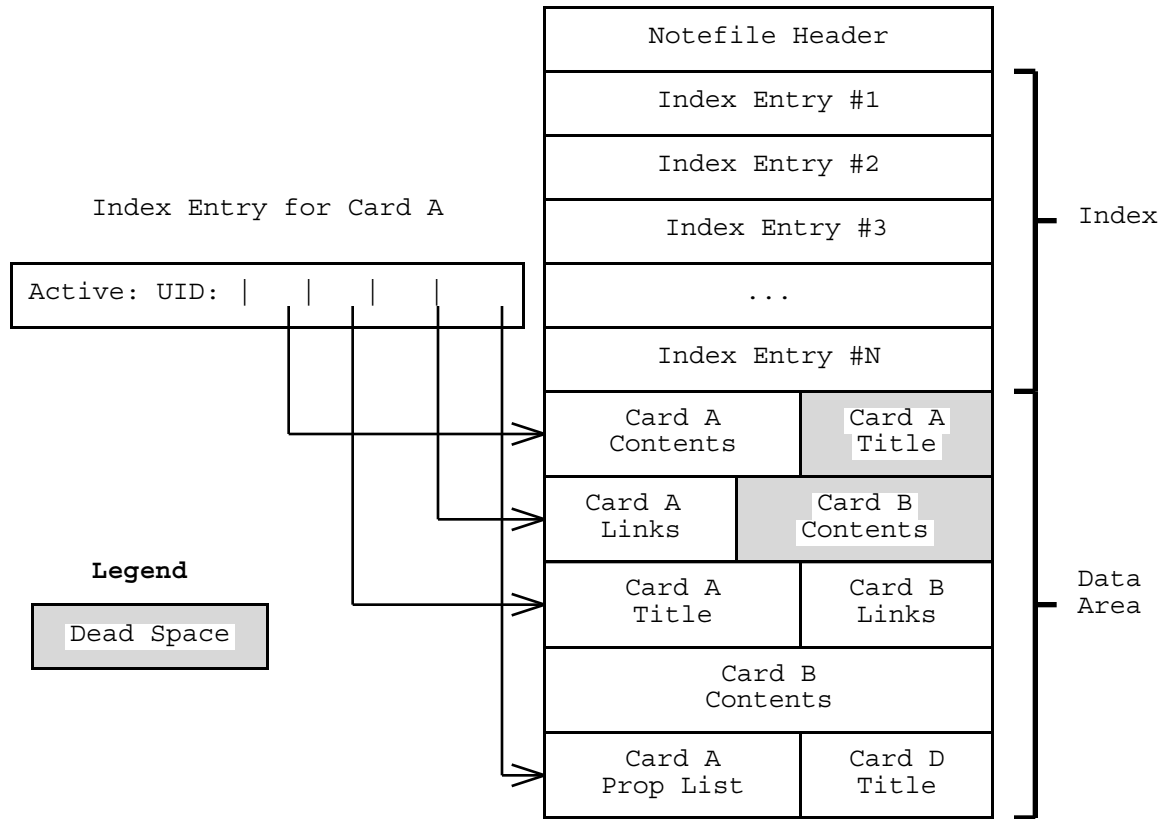


Figure A-1. The structure of a notefile.

As long as the notefile has not been compacted, all the old information can be accessed (and made to be "current") via the notefile Inspect and Repair facility. Inspect and Repair does this by ignoring the index and parsing the entire data area to produce a listing of all the information (both current and old) about a card that is stored on the data area. See the Inspect and Repair manual for more information.

Notefile Checkpointing

As long as a notefile is open, its index area is cached in memory. When a card part is saved, the card part's information is written to the end of the data area but the card's actual index entry on the notefile is not updated with this new location. Instead, the appropriate pointer in the card's in-memory index entry is updated. Thus, the index in the notefile continues to point to the old information in the data area, while the in-memory index points to the new information.

Thus while a notefile is open, its current state is distributed between the actual notefile and information cached in memory. The current index is cached in memory. For cards open on the screen (or cached in memory from the programmer's interface), the "current" card part information is contained in an in-memory cache. For all

other cards, the current information is contained in the data area of the notefile pointed to by the in-memory index entry.

If a machine crashes while a notefile is open, the information cached in memory is lost. A crash not only discards changes made to cards on the screen, but it also leaves the information stored on the notefile in an inconsistent state. For example, the index on the notefile may point to old information in the data area. This occurs because the new information (e.g., a new title) is written to the data area but only the in-memory index pointers (which are lost in the crash) have been updated to point to the new information.

Checkpointing forces all of the in-memory information to be written onto the notefile. Specifically, checkpointing causes all open cards to be saved and the in-memory index to be written to the notefile's index area. Thus, immediately after checkpointing, the notefile itself contains its current (and consistent) state. If the machine were to crash at this point, no information would be lost and the notefile would be consistent.

Checkpointing also writes a **checkpoint pointer** onto the notefile header. The checkpoint pointer contains the location of the end of the data area (i.e., the end of the notefile) at the time the checkpoint is done.

As the notefile is used after the checkpoint, information is written in the data area past the checkpoint pointer but only the in-memory index entries are updated to point to this information. The on-file index entries still point to the information in the data area *before* the location referenced by the checkpoint pointer. Thus, a consistent notefile can be constructed from the index area and all of the information in the data area located before the checkpoint location. This is essentially the notefile as it was at the time of the last checkpoint. (Note: one small exception is that changes to a card's size on the screen are actually written in the middle of the data area rather than at the end. Thus, truncating a notefile to its checkpoint location cannot "undo" the reshaping of a card.)

When opening a notefile after a crash, the system will insure that the notefile is in a consistent state. It does so by truncating the data area to the last checkpoint location, saving the truncated information if requested by the user. This leaves the notefile in the state it was during the last checkpoint before the crash.

Aborting a notefile does the same thing. It truncates the data area to the last checkpoint location, thereby eliminating all changes made to the notefile since the last checkpoint. It also discards the in-memory index. Thus, the notefile is left in the exact state it was after the last checkpoint.

Finally, note that notefile Close forces a checkpoint. Therefore, aborts and recovery after crashes actually restore the NoteFile to its state as of the last user-initiated checkpoint or close.

Storing links and repairing notefiles with inconsistent links

The links card part is divided into three subcomponents: to-links, from-links, and global links. The to-links is a list of all links whose Source is the given card (i.e., that point from the card to some other card). The from-links is a list of links whose Destination is the given card (i.e., that point from some other card to the given card). Finally, the global links is a subset of the to-links that includes only the Global links originating in the given card.

Given this scheme, every link is stored on the notefile in three different places. First, if the link is Local it is stored inside the link icon which in turn is inside the content part of the link's source card. If the link is global, it is stored in the global links subpart of its source card. Second, the link is stored in the to-links list of its source card. Third, the link is stored in the from-links list of its destination card.

These three records of the same link occasionally get out of synch, resulting in an inconsistent notefile. There are a number of symptoms of such inconsistency. For example, the **ShowLinks** display for card may indicate that the card is a destination for a link from some source card X while the ShowLinks display for X does not include a ToLink to that destination card. Occasionally, inconsistent links will also result in link icons that contain the words "DELETED" or "FREE" when displayed on the screen. This usually means that the card at one end of a link was deleted, but somehow the links of the card at the other end were not updated. Such link icons cause NoteCards to break when you try to follow them.

One function of the NoteCards **Inspect & Repair** facility is to resynchronize the three records for all links in the notefile. The inspector's third phase rebuilds the links as follows. First it removes all to- and from-links for every card. Then it reads the contents for each card and recreates to-links and from-links by looking at the links found inside the link icons in the card's content and in its global links list. In addition, the links rebuilder phase of the notefile inspector can rebuild filebox contents from cards pointed to by the filebox, and the set of all link types from the list of all links.

[This page intentionally left blank]

The main purpose of the notefile inspector is to repair notefiles by rebuilding their links. However, the inspector is generally useful for checking the health of notefiles, deleting cards and backing up cards (or more precisely, card parts) to previous versions. Thus, you may want to use the inspector even if your notefile is healthy and doesn't need its links rebuilt.

The notefile inspector has three separate phases: reading the notefile's data area searching for healthy card parts, allowing the user to make modifications, and rebuilding the links. The process can be aborted after phase 1 or 2 if desired. This document describes each of the three phases in turn and concludes with tips, strategies and pitfalls to watch for.

Using the notefile inspector requires some knowledge of the innards of notefile organization. We recommend that you read Appendix A, Notefile Concepts, before continuing.

Note: The information in this appendix was copied whole from the documentation for a prerelease of NoteCards and has not yet been updated for this release. As such, it may be inaccurate in places.

Running the Notefile Inspector: Phase 1: Scouring the Data Area

To start the inspector on a notefile, first be sure that the notefile is not open. Then select the item **Inspect&Repair** from one of the NoteFile Ops menus (i.e. from the notefile Banner, the MenuBox icon or the notefile FileBrowser). There is one option available at this level by "pulling to the side" called **ReadSubstances**. This ensures that substances of all cards pronounced valid by the inspector are readable. If this option is not invoked, then a check is still run on the length of the substance, but not on its contents. Unfortunately, the ReadSubstances option requires MUCH more work by phase 1. We recommend that you only use this option if Phase 3 (link rebuilding) breaks with some error like "Bad Piece Tbl" from TEdit. In that case, up-arrow out of the break and start the Inspect&Repair process over again, this time using the slower but more comprehensive ReadSubstances option.

Selecting **Inspect&Repair** will invoke phase 1 of the inspector, wherein the data area of the notefile is scoured for valid card parts. A record of all such parts is kept and statistics printed out at the end. You'll be asked to position the window in which those statistics as well as later inspector communications will be printed. (Note that closing this window will abort the Inspect&Repair process if you confirm.) You can monitor the progress of phase 1 by watching the prompt window. It will be printing messages like "Processing byte xxxxx of yyyy."

When phase 1 has completed and you've positioned the interaction window, statistics on your notefile will be provided. You'll be told the total number of cards and the number of deleted ones. If all seems well with the world, the next line will read "All non-deleted

cards look okay." If not, there will be various messages outlining the problems. (See Figure 1.)

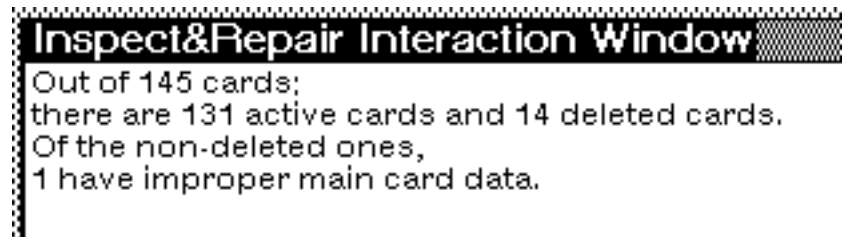


Figure B-1: Snapshot of a sample interaction window

Normally, the messages describing problems are of the form shown in Figure 1, namely a count of the number of cards broken in a given way. Some messages are a bit trickier. If you have fileboxes with bad substances (i.e. main card data), then you're told that if you don't wish to delete or back up such fileboxes to previous versions, then phase 3 will rebuild them. If, however, the filebox is one of the top level boxes (Contents, Orphans, ToBeFiled), then deletion is not an option. (See Section 3.)

If there are cards having user-defined types whose type definition code has not been loaded, then you'll get a message to that effect, something like "<n> cards have unknown card types (FOO BAR)." At this point you should load the lisp files containing the definitions of the unknown card types. If not, then these cards will show up with bad substance in phase 2. If you do load the appropriate files, then run **Recheck Bad Cards** to get the bad cards list recomputed before inspecting any cards.

If you have a card for which no substance versions could be read, then you'll also get unknown card type messages for it (reading something like "<n> cards have unknown card types (NIL)"). This is because the inspector couldn't find a card type on the notefile for that card.

A menu of options appears attached to the upper right corner of the interaction window. (See Figure 2.) The particular options you get in that menu depend on the state of your notefile and are described below. The last three options appear in all cases. The other two may or may not be present in the menu you get. In any case, you should select one of the options before attempting any other NoteCards-related work.



Figure B-2: The Phase 1 options menu.

Continue Repair

This option is only available if the notefile is in fairly good health (i.e. okay except for fileboxes to rebuild or global links to rebuild - see Section 3). Selecting it causes Inspect&Repair to move to phase 3 and rebuild your notefile's links.

End Inspect & Repair

This option is only available if it seems that you don't need to continue to the link rebuilding phase. You will not get this option if you've deleted any cards, or generally if there are problems with the notefile. Choosing this option causes the Inspect&Repair process to end gracefully (via a normal checkpoint and close notefile), thus skipping phase 3 of rebuilding links.

Abort

Choosing this option aborts the Inspect&Repair process entirely, throwing away any changes you might have made (such as card deletions or back ups.) It requires your confirmation in the prompt window.

Recheck Bad Cards

Recompute the bad cards list by running the last part of phase 1 again. This is useful if you've loaded files containing definitions of previously undefined card types.

Inspect Cards

This brings up a menu of titles of active cards with which you can inspect, delete, or back up particular cards. There is a "pull-across" menu item called **Include Deleted Cards**, which if selected will include card titles for deleted cards as well as active ones. Using this option, one can undelete deleted cards and restoring them to some previous version.

Running the Notefile Inspector: Phase 2: Your Chance to Tinker

After selecting **Inspect Cards** in the interaction window's attached menu, a menu containing numbered titles of notecards pops up and is attached to the interaction window's lower left corner. It contains the first 10 characters of titles for all active cards and possibly deleted cards as well if you selected the submenu item **Include Deleted Cards** described above. The menu can hold some 100 card titles. If your notefile has more than that, then the menu is composed of several pages each containing around 100 cards. Rapid switching between pages is possible. Figure 3 below shows a sample card inspector menu.

Attached to the upper right corner of the cards inspector menu is a menu containing at least the three options: **Abort**, **Done** and **Search**. If the menu has multiple pages (there are more than 100 active cards in the notefile), then the attached menu will also include the items **Next Page**, **Previous Page**, and **First Page**. Selecting these causes the current menu to be exchanged for either the next menu, previous menu, or first menu, respectively.

Clicking **Abort** causes the entire Inspect&Repair process to quit, throwing away any changes you've made. It requires your confirmation in the prompt window. (This is equivalent to choosing

Abort from the inspector window as described in the previous section.)

Choosing **Done** from this attached menu indicates that you're done tinkering with card parts and wish to return to the main interaction window. This causes the card inspector menu to close and the phase 1 process outlined in the previous section to be performed again (leaving you looking at something like Figure 1). Note, however, that the scanning of the data area is not repeated (it takes way too long). Rather, your changes are made to the in-core index array and the statistics on bad cards are recomputed. This cycle of compute statistics (phase 1) followed by inspect and tinker (phase 2) can be repeated as often as desired. Eventually, you must either abort, end the Inspect&Repair via **End Inspect&Repair**, or continue to phase 3 via **Continue Repair**.

Choosing **Search** from the attached menu allows you to find those cards having titles matching a given string. This works much like the Search card in Notecards. That is, you specify a string and the list of cards whose titles contain that string is printed out. This is handy, for example, if you want to undelete a card, but only know its title. Note that the search is case *ins*sensitive.

The card inspector menu

In the card inspector menu, those titles corresponding to deleted cards have a line drawn through them. Those having some sort of problem appear shaded. In addition, an upper-case letter suffix is attached to such entries indicating the problem. For example, in the card inspector menu shown in Figure 3, the shaded menu item "10: BNF for Li|S" indicates that the card with title beginning "BNF for Li" has bad substance. The letter codes are S, L, P, and T indicating bad substance, links, property list, and title, respectively. If such a letter code appears in lower case, then the meaning is that the current version of that card part is beyond the last checkpoint pointer. (There may have been a crash, for example, thus preventing the notefile from closing normally.)

Figures 3 and 4 show both pages of a two page card inspector including deleted cards. Sometimes deleted cards (e.g. "110: NIL|LSPT" in Figure 4) show up shaded with LSPT suffixes. This is because they were deleted before any data about the card was written to the notefile. Thus there are no valid card parts to back up to for those cards. But often, deleted cards can be backed up to previous versions. (See Figure 6 and discussion below.)

Card inspector: Page 1 of 2				Abort Done Search Previous Page Next Page First Page
1: Contents	2: Orphans	3: To Be File	4: Link Label	
5: Registry	6: Papers	7: Value Cell	8: Value cell	
9: Introducti	10: BNF for LIS	11: BNF for Co	12: Format of	
13: Recomputin	14: Example fr	15: Implicatio	16: Extensions	
17: Formula ex	18: Collect ex	19: Outline	20: Value cell	
21: People	22: Schatz, Br	23: Sharma	24: Dean, Jeff	
25: Kasif, Sim	26: Krovetz, B	27: Liles, Bil	28: Feuerman,	
29: Cook, Lesl	30: Kondo, Jin	31: Schwartz,	32: Ward, Sand	
33: Gregory, R	34: Schwartz,	35: Delisle, N	36: Chapman, G	
37: Hamlet, Di	38: Ellis, Ski	39: Halasz, Fr	40: Brennan, S	
41: Moore, Lee	42: VanLehn, K	43: Carnese, D	44: Notecards	
45: NC in Loop	46: Review of	47: Pros and C	48: Introducti	
49: Discussion	50: Paper outl	51: Document f	52: Reggia, Ji	
53: Wohltman,	54: junk	55: NoteCards	56: Applicatio	
57: Applicatio	58: Introducti	59: Random not	60: NCAPPLICAT	
61: Mail box	62: Question o	63: The progra	64: ARI Exampl	
65: Second exa	66: Summary of	67: Notes from	68: Mont-Reyna	
69: Kim, Scott	70: Weiser, Ma	71: Places	72: REI	
73: Frescoe's	74: Higgins, R	75: Dieter's	76: Pitts, Wel	
77: Leavitt, M	78: Marshall,	79: Kimball's	80: Deerwester	
81: CPSR	82: Moran, Tom	83: Nelson, Te	84: Informatio	
85: Mail	86: Loops	87: Colab	88: Koncepts C	
89: Wijsman, N	90: Todd, John	91: Draft	92: Notes on K	
93: Stats on K	94: Cards	95: Links	96: Notefile	
97: Notes from	98: Thoughts o	99: Document f	100: Notes from	

Figure B-3: A card inspector menu: Page 1.

Card inspector: Page 2 of 2		Abort Done Search Previous Page Next Page First Page
102: Tailorabil	103: Global par	
104: Interlisp	105: Card regis	
106: PI-based f	107: Event hook	
108: Template c	109: Interlisp	
110: Programmer	111: Types mech	
112: Card types	113: Tailorabil	
114: foo	110: foo	
117: asdf	110: FILE OPT	
119: Table of P	120: Table of c	
121: Table of c	122: Support fo	
123: Outline	124: Techniques	
125: Document f	126: PI-based f	
127: Event hook	128: Document f	
129: Thoughts	130: Talks	
131: Talk on NC	132: Types hier	
134: Title slid	135: Levels of	
136: Global par	138: PI sketch	
139: Use of PI	140: Event hook	
141: Types mech	142: Miscellane	
143: IDE exampl	144: Kinds of c	
145: Estrin, De		

Figure B-4: A card inspector menu: Page 2.

At the top of the first page of any notefile's card inspector, there are entries for the top level boxes and special cards. These are labeled

"1: Contents", "2: Orphans", "3: To Be File", "4: Link Label", and "5: Registry".

Note that if the inspector shows no entry in the card inspector menu for some card, it is because that card has been deleted. If you've asked to show deleted cards and it still doesn't appear, it's because the notefile has been compacted since that card was deleted.

If you button card title entry in the card inspector menu, then a popup menu allows up to two choices **Inspect** and/or **Delete**. If the card is currently deleted (has a line drawn through it), then the Delete option is replaced by **Undelete**. Certain cards cannot be deleted and thus their popup menus only contain the Inspect option. These are the 5 special cards mentioned above.

Choosing Delete or Undelete from this popup menu causes the card to be deleted or undeleted, respectively and the line through the menu item either drawn or undrawn. Note, however, that this action (and all others) can be undone by choosing ABORT from either the card inspector menu or the interaction window menu.

Choosing Inspect from the popup menu for a card entry brings up a card parts inspector for that card.

The card parts inspector

Figures 5 and 6 below show examples of card parts inspectors. A card parts inspector is composed of four attached menus arranged vertically and one attached operations menu atop the stack.

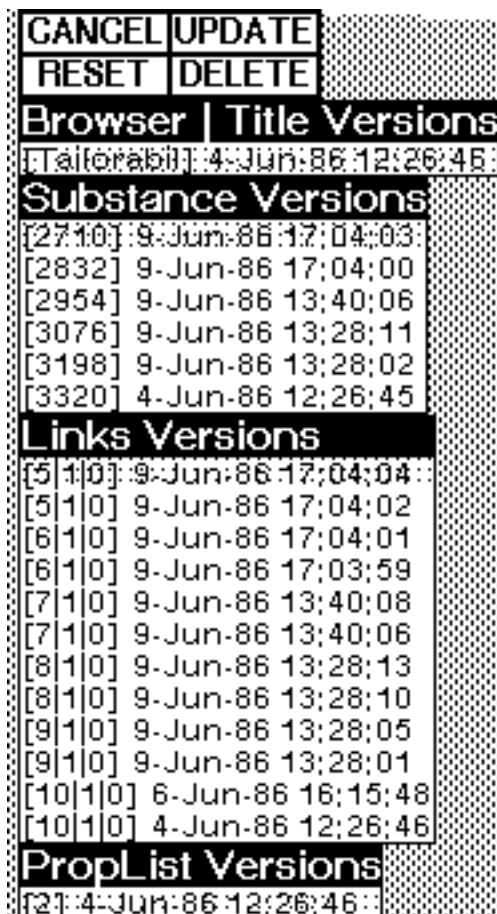


Figure B-5: Card parts inspector for a browser card.

In Figure B-5, the four menus contain entries for every valid version of card parts for the browser card with title "Tailorability". The top menu is for versions of titles and below that are menus for versions of the card's substance, links, and prop list. For example, the Substance submenu contains entries for 6 versions of the substance of this card. The current version of each card part is shaded. Each menu entry gives the date that that version was written. (Very old notefiles may show "NO DATE AVAILABLE" in place of a real date.)

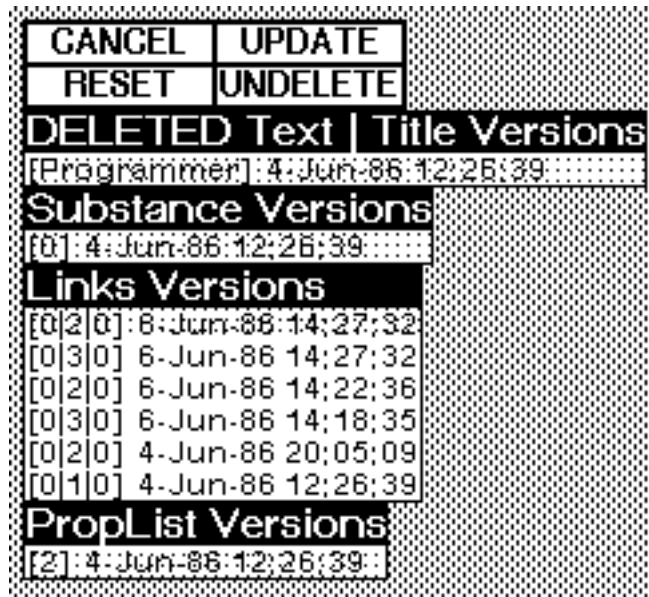


Figure B-6: Card parts inspector for a deleted card.

Figure B-6 shows a card parts inspector for a deleted text card and thus allows the UNDELETE option. By clicking UNDELETE, this card can be backed up to its last version.

If the current version of the card part is bad, then the menu entry will be a string so indicating, for example, "BADSUBSTANCE."

The title of the top menu includes the card's type. In addition, each menu item contains some information, in square brackets, before the date. In the title versions menu, this information is the first few characters of the title. In the substance versions menu, it is the number of bytes in the substance. In the links versions menu, it is a triple of numbers giving the number of to links, from links, and global links for this card. Finally, the proplist versions menu includes the number of entries on the property list for this card (i.e. twice the number of attribute-value pairs).

Atop the stack of menus is an attached menu of operations, described below.

CANCEL

This cancels this card parts inspector, throwing away any changes made.

UPDATE

This closes the card parts inspector, incorporating any changes (backing up to previous versions of card parts) you might have done.

DELETE

This option closes the card parts inspector and deletes the card.

UNDELETE

For cards that have been deleted, this option appears instead of DELETE. Choosing it causes the card to be undeleted.

RESET

This causes the selections in the submenus to be restored to the values they had when the card parts inspector was first brought up. (Equivalent to doing CANCEL and then inspecting this card again from the card inspector menu.)

Note that cards that can't be deleted (like the top level fileboxes) don't have the DELETE option on their card parts inspector.

Buttoning an entry in a submenu of a card parts inspector pops up a short menu unless the entry is for a bad version (e.g. "BADSUBSTANCE"). This menu contains at least the entry **Inspect** and possibly **Change Selection**, if the selected entry is not the same as the current one (i.e. not shaded).

Choosing Inspect allows further inspection of the details of the selected card part version. (See Figure B-7 below.) For example, inspecting a title version brings up the Interlisp inspector on a record containing the title, date and card object. Similarly, inspecting a links or prop list version brings up the Interlisp inspector on a record containing the lists of links (i.e. to links, from links and global links) or the prop list. Note that changing values in these Interlisp inspectors has no affect on the notefile and is ignored.

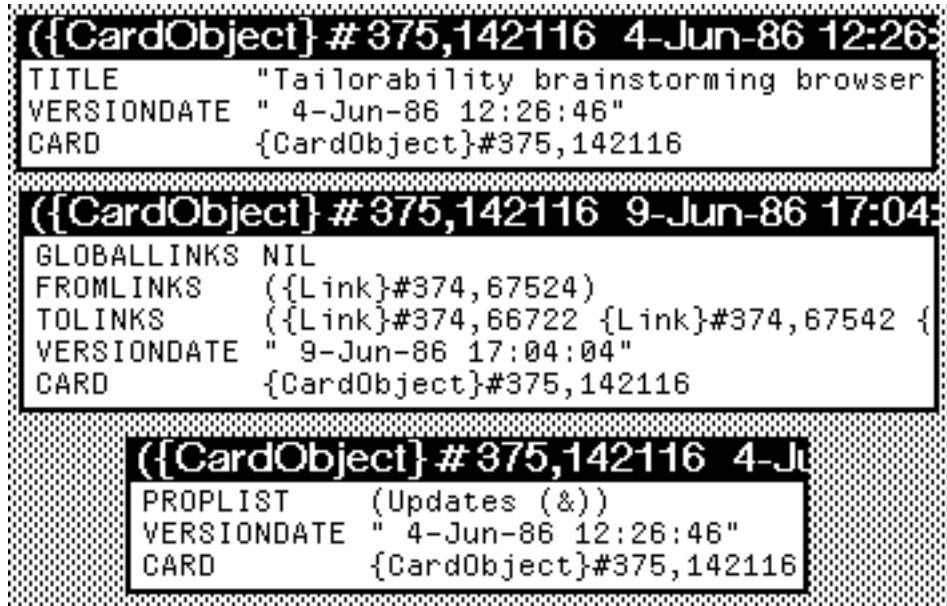


Figure B-7: Card part version inspectors for title, links and prop list.

All substance versions for card types inheriting from Text, Sketch and Graph can currently be inspected. (This includes most cards except those having user-defined substance types, like the NCFfile card.) Inspecting a card's substance version will bring up a window showing a copy of the substance. (Note that changes to this copy have no affect on the notefile.) Any links in the substance of the card will show up as bracketed strings describing the link.

Choosing **Change Selection** from the card part version popup menu causes the current selection to be changed to the selected one. This changes a card part's current version to another legal one. (This change can be undone by resetting or canceling the card parts inspector as well as by later aborting the card inspector or interaction window.)

Running the Notefile inspector: Phase 3: Rebuilding your links

To complete the Inspect&Repair process, select the **Continue Repair** option from the interaction window menu. This invokes phase 3, the links rebuilder. Normally, this simply rebuilds links from card substances. In certain circumstances, it may do extra work as well. If your link types list is bad, and you didn't back it up to a previous version, then phase 3 will rebuild it. If there are fileboxes with bad substances that you haven't either deleted or backed up to previous versions, then phase 3 will rebuild them. Finally, if there are cards with bad links that you haven't backed up or deleted, then phase 3 will rebuild those links as well. (It rebuilds ALL to links and from links anyway. For those cards, it will rebuild global links as well.)

First the links rebuilder removes all the to and from links for every card. Then it reads the substances for each card and recreates to

links and from links by looking at the links found inside the link icons in the card's substance.

The link rebuilder is also able to rebuild bad filebox substances. It does this by looking for all cards in the notefile with from links from the bad box and creating a new substance for the box containing only links to those cards. This process loses any text that the box might have contained as well as scrapping the original ordering of links. Nonetheless, in some cases this may be preferred to backing up the substance to a previous version or to deleting the box altogether.

The links rebuilder can rebuild the notefile's list of link types in a similar manner. That is, it records the set of link types seen on valid links and replaces the old links types with the new set. Note that this throws away any link types for which there are no links in the notefile.

Finally, the links rebuilder can rebuild bad global links for a card. It does this by looking for any cards with from links from the bad card that are global. This assumes that the card at the destination end has good links. Thus, if the cards at both ends of a global link have unreadable links, then there is no way to recover that link.

Note that the links rebuilder automatically files any unfiled cards in the ToBeFiled box. A message is printed in that case.

Tips and hints for using Inspect & Repair

This section contains a list of strategies and tips for using Inspect&Repair. For the most part, they are ordered from the useful and obvious to the esoteric. Several of these are implicit in the first four sections of the document, but are repeated here for emphasis and completeness.

When in doubt, abort!

All your changes will be lost, but then if you're uncertain about what's happened this is the safe course. Often, in fact, you may simply want to check the health of your notefile and abort without tinkering.

Fixing versus tinkering

There are two main ways to use the inspector, either for fixing a broken notefile, or tinkering with a healthy one. The latter case occurs when you wish to recover some card that was inadvertently deleted. Or back up a card that was accidentally changed to its original version.

Inspect & Repair after a crash

If you try to open a notefile and you get the infamous "Work was done since last checkpoint..." message, then you're given the chance to run Inspect&Repair on the notefile. This is a good idea

whenever you wish to recover that post-checkpoint work. The inspector will integrate the changes you made since that checkpoint. (You do have to continue through phase 3, however.)

Compacting

It used to be the case that old versions of card parts were in the notefile but inaccessible. Thus, there was little reason not to compact a notefile often. Now there is a tradeoff between the need to save space by compacting versus the need to be able to back up using Inspect&Repair. Probably the safest course is to keep a backed up copy of the pre-compacted notefile around until you have confidence that the compacted one is healthy and that you have no need for previous versions of any of its cards.

Inspect & Repair closes the notefile before starting

This means that if you are working in your notefile and notice a card you'd like to inspect a previous version of, you must record the card's title and close the notefile. Then, run Inspect&Repair, find the card's entry in the inspector menu using the Search facility, and tinker with it as desired.

Fixing enough problems to allow phase 3 to run

You can't run phase 3 unless Inspect&Repair thinks your notefile is above a certain threshold of health. There are certain problems it can handle (e.g. bad filebox substances, see Section 3), and others that it can't (e.g. bad substance for non-filebox). You have to decide either to fix these sorts of problems yourself in phase 2, let phase 3 attempt to rebuild them, or just abort the whole thing (always an option).

Sometimes these decisions can be tricky. For example, suppose a filebox's substance is bad. Call it BadBox. Should you (a) delete BadBox altogether, (b) back its substance up to a previous version, or (c) allow phase 3 to rebuild it by looking for from links in other cards from BadBox? Option (c) may not be advisable if there was important text in BadBox or if the order of cards in BadBox was important. On the other hand, option (b) may be of little use if the last good version is too out of date (or if there is no good version at all

Once your system administrator has installed the NoteCards software on your Sun, you can customize your NoteCards environment. The initialization file allows you to specialize your NoteCards environment to the idiosyncracies of your individual site. This file is automatically loaded when you start a fresh sysout.

Site Initialization File

When NoteCards starts, it reads in a site initialization file. This file sets things like the pointer to fonts, printers, and other site-specific parameters.

Locating the Initialization File

When running under SunOS, NoteCards looks for a site initialization file in a number of locations.

LDEINIT

If the UNIX environment variable `LDEINIT` is set to a complete NoteCards file name, NoteCards looks there first for the site initialization file:

```
prompt% setenv LDEINIT /users/smith/site-init.lisp
```

- or -

```
prompt% setenv LDEINIT /users/smith/nc-init
```

/usr/local/lde/site-init.lisp

If `LDEINIT` is not set or there is no file with the name given, NoteCards looks for a site initialization file called `/usr/local/lde/site-init.lisp`. The distribution tape contains a standard site initialization file in the NoteCards lisp directory `/usr/local/lde/lisp/init.NoteCards` which is linked to `/usr/local/lde/site-init.lisp`. You or your system administrator should customize this file for your site. The comments below and in the standard `site-init.lisp` describe the parameters it sets and gives some guidelines for customizing it to your local conditions.

{DSK}INIT.LISP

NoteCards looks for a site initialization file on your NoteCards home directory (`{DSK}`). Chapter 4, System Use Issues, describes the `{DSK}` device.

Finally, to load an initialization file after you have been working in a sysout, FileBrowse the directory the initialization file is in, select the file with the left mouse button, and select the **Load** option. The file will load and reset all of the initialization variables.

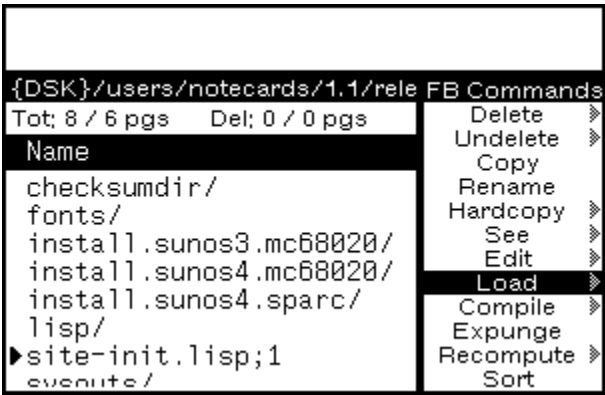


Figure C-1. A FileBrowser loading an initialization file.

Initialization File Structure

The structure of each command in the initialization file is as follows:

`(SETQ variable-name value)`

Each command must appear between parentheses. There must always be an even number of parentheses. Every open parenthesis must have a corresponding close parenthesis. SETQ is the assignment command. The variable name is always a single word, and the value is usually either a single word preceded by a single quote mark (') or a list of words enclosed in parentheses preceded by a single quote mark.

The initialization file must always start with two specific lines. The first line must be a comment line, indicated by starting the line with a semicolon (;). The second line must contain the command `(CL:IN-PACKAGE "INTERLISP")`.

Site Variables

The following NoteCards variables should be set in your site initialization file.

LISPUSERSDIRECTORIES

The list of paths to search for library files. Every path in this list should also be in DIRECTORIES.

```
(SETQ LISPUSERSDIRECTORIES
  ' (" {DSK}/usr/local/lde/lispusers/"
    " {DSK}/usr/local/lde/lisplibrary/"))
```

Note that each directory name is a string enclosed in double quotes ("). Unless you are a developer who is modifying NoteCards, there will probably never be any reason to change these values. You only need to change the value for this variable if you move the files in these directories to some other location, such as a central fileserver.

DIRECTORIES

The list of paths to search for files that are not found in the current NoteCards connected directory. The current NoteCards connected

directory is distinct from the current UNIX connected directory, and NoteCards will not necessarily search for files in the UNIX connected directory.

```
(SETQ DIRECTORIES (COPYALL LISPUSERSDIRECTORIES))
```

Note that there is *no* quote mark (') before the parenthesis in front of COPYALL. If you set DIRECTORIES in this fashion, this command must follow the one which sets LISPUSERSDIRECTORIES.

You can also simply copy the directories from LISPUSERSDIRECTORIES, e.g.

```
(SETQ DIRECTORIES
  ' (" {DSK}/usr/local/lde/lispusers/"
    " {DSK}/usr/local/lde/lisplibrary/"))
```

Unless you are a developer who is modifying NoteCards, there will probably never be any reason to change these values.

DISPLAYFONTDIRECTORIES

A list of directories to search when the system is looking for display fonts. The site initialization file should set DISPLAYFONTDIRECTORIES to a list of path names where each path name is represented as a string in double-quotes ("), e.g.,

```
(SETQ DISPLAYFONTDIRECTORIES
  ' (" {DSK}/usr/local/lde/fonts/display/presentation/"
    " {DSK}/usr/local/lde/fonts/display/publishing/"
    " {DSK}/usr/local/lde/fonts/display/printwheel/"
    " {DSK}/usr/local/lde/fonts/display/miscellaneous/"
    " {DSK}/usr/local/lde/fonts/display/jis1/"
    " {DSK}/usr/local/lde/fonts/display/jis2/"
    " {DSK}/usr/local/lde/fonts/display/chinese/"))
```

If you remove the Chinese and Japanese fonts from the font directories it is not necessary to edit this variable to reflect the change.

You only need to change the value for this variable if you move the fonts to some other location, such as a central fileserver. If you do change the value for this variable, note that each location is represented as a double quoted string and that each path name must end with a slash (/).

INTERPRESSFONTDIRECTORIES

A list of directories for the system to search for Interpress font widths files. The site initialization file should set INTERPRESSFONTDIRECTORIES to a list of path names where each path name is represented as a string in double-quotes ("), e.g.,

```
(SETQ INTERPRESSFONTDIRECTORIES
  ' (" {DSK}/usr/local/lde/fonts/interpress/presentation/"
    " {DSK}/usr/local/lde/fonts/interpress/publishing/"
    " {DSK}/usr/local/lde/fonts/interpress/printwheel/"
    " {DSK}/usr/local/lde/fonts/interpress/miscellaneous/"
    " {DSK}/usr/local/lde/fonts/interpress/jis1/"
    " {DSK}/usr/local/lde/fonts/interpress/jis2/"))
```

```
"{DSK}/usr/local/lde/fonts/interpress/chinese/"))
```

If you remove the Chinese and Japanese fonts from the interpress font directories it is not necessary to edit this variable to reflect the change.

You only need to change the value for this variable if you move the interpress fonts to some other location, such as a central fileserver. If you do change the value for this variable, note that each location is represented as a double quoted string and that the lowest subdirectory name in each path must end with a slash (/).

POSTSCRIPTFONTDIRECTORIES

The list containing the name of the PostScript font width files, for PostScript printers.

```
(SETQ POSTSCRIPTFONTDIRECTORIES  
  ' (" {DSK}/usr/local/lde/fonts/postscript/"))
```

You only need to change the value for this variable if you move the PostScript fonts to some other location, such as a central fileserver. If you do change the value for this variable, note that the location is represented as a double quoted string and that the lowest subdirectory name in the path must end with a slash (/).

USERGREETFILES

The list of places to search for personal initialization files. If not set in the site initialization file, no personal initialization file is used. The list should be similar to the following

```
(SETQ USERGREETFILES  
  ' ( (" {DSK}~/lde/INIT.LISP") (" {DSK}~/INIT.LISP") ) )
```

This will search for the file `INIT.LISP` in your home directory and in the subdirectory `lde` immediately under your home directory. The file `INIT.LISP` can be renamed to what ever you want it to be, but it has to be a name which is used by all users of the NoteCards system.

Unless you are a developer who is modifying NoteCards, there will probably never be any reason to change these values or use user greet files.

DEFAULTOSTYPE

Specifies the default operating system type for file servers.

```
(SETQ DEFAULTOSTYPE 'UNIX)
```

Unless you are using other than Sun NFS (Network File System) servers there is no reason to change the value of this variable.

DEFAULTPRINTINGHOST

A list of names of default printers.

```
(SETQ DEFAULTPRINTINGHOST NIL)
```

`NIL` is the default setting and signifies that there are no default printers specified. This variable can also be set from the **Set Default Printer** background menu option, (see Chapter 16, Printing).

The following example shows how to set `DEFAULTPRINTINGHOST` to a mixed list of PostScript and Interpress printers.

```
(SETQ DEFAULTPRINTINGHOST
```

```
' (MAUI |Tremor:mv:envos|))
```

- or -

```
(SETQ DEFAULTPRINTINGHOST
  ' (MAUI "Tremor:mv:envos"))
```

MAUI is the name of a PostScript printer, |Tremor:mv:envos| the name of an interpress printer. Each interpress printer name must be surrounded by vertical bars (|) or double quotes (").

DEFAULTPRINTERTYPE

The default printer type for all the available printers.

```
(SETQ DEFAULTPRINTERTYPE NIL)
```

Possible values are 'POSTSCRIPT, 'INTERPRESS, and NIL meaning unspecified. Note that 'POSTSCRIPT and 'INTERPRESS are all preceded by single quote marks ('), but that NIL is not.

If all your printers are PostScript printers you would make the following setting.

```
(SETQ DEFAULTPRINTERTYPE 'POSTSCRIPT)
```

If all your printers are not of the same type set DEFAULTPRINTERTYPE to NIL, and add the following command for each printer.

```
(PUTPROP 'Printer-Name 'PRINTERTYPE 'Type)
```

Where 'Printer-Name is the name of the printer and 'Type is one of 'POSTSCRIPT, 'INTERPRESS, or 'PRESS, for example.

```
(PUTPROP 'MAUI 'PRINTERTYPE 'POSTSCRIPT)
```

```
(PUTPROP '|Tremor:mv:envos| 'PRINTERTYPE 'INTERPRESS)
```

IDLE.PROFILE

The lines below set the IDLE.PROFILE such that any user with a valid UNIX login is allowed to exit idle mode.

```
(LISTPUT IDLE.PROFILE 'AUTHENTICATE 'UNIX)
```

The value of AUTHENTICATE determines what mechanism is used to check passwords. If 'UNIX, it checks your password with SunOS. If T, it uses the NS authentication protocol (this requires the presence of an NS Authentication server accessible via the network). If NIL, the password is not checked at all--any password is accepted. The latter is only useful if ALLOWED.LOGINS contains *.

```
(LISTPUT IDLE.PROFILE 'ALLOWED.LOGINS '(*))
```

The value of ALLOWED.LOGINS determines who is allowed to exit idle mode. If the value is NIL no login is required at all to exit idle mode. If the value is '(*), the system requires a login but lets anyone exit idle mode.

```
(LISTPUT IDLE.PROFILE 'SAVEVM NIL)
```

This line sets the IDLE.PROFILE such that NoteCards does not save the virtual memory file to disk when it enters idle mode. To

have NoteCards automatically save the virtual memory file to disk when it enters idle mode, replace `NIL` in the line above with `T`.

\\BeginDST The day of the year on or before which Daylight Savings Time takes effect (i.e., the Sunday on or immediately preceding this day). Must be set to 98 in the USA if NoteCards is to perform time computations correctly. (Note: This number is subject to future Congressional legislation.) If you are in a region where Daylight Savings Time is not observed, use the value 367.

```
(SETQ |\\BeginDST| 98)
```

\\EndDST The day of the year on or before which Daylight Savings Time ends. Must be set to 305 in the USA.

```
(SETQ |\\EndDST| 305)
```

XCL:*LONG-SITE-NAME* A long version of your company's name, for example, "Envos Corporation."

```
(SETQ XCL:*LONG-SITE-NAME* "Envos Corporation")
```

XCL:*SHORT-SITE-NAME* A short version of your Company's name, for example, "Envos."

```
(SETQ XCL:*SHORT-SITE-NAME* "Envos")
```


[This page intentionally left blank]

If you encounter inexplicable problems shortly after you install NoteCards, they may be due to files being corrupted — the release tape may have been damaged, errors may have occurred while the tape was being read, etc. If you have unexplained problems, we recommend that you verify the checksums of your installed files.

Description

The script generates checksum files named *filename.check* and compares them to the released *filename.sum* residing in the */checksumdir* subdirectory.

The checksum script reports inconsistent files, the correct checksum values for the files, and an error message. The checksum of individual files can be generated with the UNIX command: **sum filename**.

Commands

ldechecksum [-cg] [ncdir [dir dirgroup]]	[Command]
-c	Generates checksums for your installed files and compares them with correct values. This is the default action.
-g	Generates checksums for the files specified.
<i>ncdir</i>	Name of the NoteCards installation directory. Usually it is /usr/local/lde .
<i>dir</i>	Any specific directory residing under <i>ncdir</i> . Only relative pathnames with respect to <i>ncdir</i> are accepted.
<i>dirgroup</i>	The directory group. Valid directory groups are system , fonts , and all .
system	includes lisp, sysouts, install.sunos3.mc68020, install.sunos4.mc68020, install.sunos4.sparc.
fonts	includes the display, interpress, press and postscript directories.
all	includes all of the above.

Output

As it begins checking each directory, the script prints a message in the form:

```
Checking directory: /usr/local/lde/subdir
```

Error and warning messages may be in one of two forms:

```
< E > 36610 6973 NoteCards.sysout
```

Indicates that file NoteCards.sysout is erroneous or does not reside in the directory. The correct checksum of 36610, together with the size (6973 kbytes) of the file, are shown.

```
< W > /usr/local/lde/fonts/display/chinese:  
Directory not installed
```

Indicates that Chinese fonts were not installed or were removed after NoteCards was installed.

Examples

```
prompt% ldechecksum /usr/local/lde
```

All files in the installed NoteCards directories in `/usr/local/lde` are checked.

```
prompt% ldechecksum /usr/local/somedir/lde system
```

This example checks all files in:

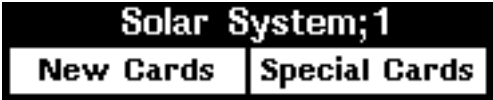
```
/usr/local/somedir/lde/install.sunos3.mc68020  
/usr/local/somedir/lde/install.sunos4.mc68020  
/usr/local/somedir/lde/install.sunos4.sparc  
/usr/local/somedir/lde/sysouts  
/usr/local/somedir/lde/lisp
```

```
prompt% cd/usr/local/lde
```

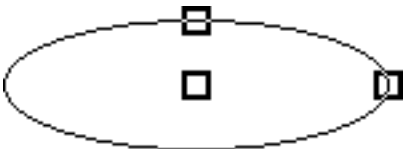
```
prompt% ldechecksum -c . fonts/display
```

This example checks only the display font directories. The period (.) is used because you are positioned under the current NoteCards installation directory.

[This page intentionally left blank]

abort	To terminate any action before it is finished. When selected from the "Notefile Ops" menu, Abort means to close a notefile without saving your changes to the notefile.
back links	System generated to-links. These links point from a Document or LinkIndex card back to the source cards from which the Document or LinkIndex was built. Hence the name "back link."
background menu	The menu brought up by holding the right mouse button down in the gray region not associated with any windows.
backward links	The same as from-links. This term is used when setting Browser and LinkIndex card parameters.
Banner	The location from which you access old cards and create new ones. Also called "notefile Banner." The "Solar System" Banner is shown below.
	
bit map	A two-dimensional grid of zeros and ones where a zero represents white space and a one represents black space. The screen is a bit-mapped display.
break	The state entered by NoteCards during error processing that allows you to recover from the error by typing Lisp commands in the break window. If you don't know what to do with a break, type ^ after the prompt to abort the operation, then start over. When a break has occurred, we recommend that you perform an Inspect & Repair on the notefile after you have closed it.
break window	A window that opens when a break occurs. The title is usually the name of the broken function.
button	A button on the two- or three-button mouse.
card	Refers to any card type in the NoteCards system. Contrast with the entry for note card .
card contents	The text or graphics in the body of a card.
card menu	The menu brought up when you hold the left mouse button down in a card's title bar.
card type	Specifies a card's functional capabilities. NoteCards comes with eight card types. Each card type is specialized to perform a specific of operation, e.g., Search cards perform searches on card titles. Document cards create documents.
card-type bit map	The image on the left of the link icon which indicates the card type. In the image below, the card-type bit map indicates a FileBox card. See the section "Change Display Mode" in Chapter 8 for more a complete list of card-type bit maps.
	
card UID	The number given to each card which uniquely identifies it. See Appendix A for more information.

caret	A blinking arrowhead in a window, indicating where the keyboard characters will appear when typed. You change the position of the caret by moving the mouse cursor and pressing the left mouse button.
cascading menu	Same as submenu.
cross-file link	A link which points to a destination card stored in a notefile other than the source card's notefile.
click	To press and release a mouse button, the left button unless otherwise indicated.
connected directory	The directory the system uses by default when you do not specify a complete path when saving, retrieving, or loading a file. Also referred to as the default directory.
connection	A line drawn between two nodes in a Graph or Browser card. A connection may or may not represent a link.
control point	A position that helps to determine the location and shape of a Sketch element. Each element has one or more control points. In the following sketch, the ellipse has three control points indicated by the boxes.



current selection	The text in a Text card that is marked as selected in some way (underlining, highlighting, etc.) and has the caret at one end of the selection, e.g.

This is ~~the selected text~~,
This is selected text,
This is ~~the selected text~~.

dashing	The property of a line that causes it to be dashed., e.g. - . - . - . - . - . - . - . - . Links in browser cards may be dashed.

default	An action taken (or value specified) unless another is specified by the user.

deleted icon	An icon which appears in a source card when the destination card a link points to is deleted. Shown below.



destination card	The card which is opened (or flashed if already open) when you click on a link icon.
directory	The name of a group of related files. In UNIX a directory is a file containing a list of other directories and files.
extension	A string appended to a file name, indicating the type of file. The extension is separated from the file name with a period; the version number is separated from the extension by a semicolon. Sketch files have the extension ".sketch" followed by a version number. See file name .

file name	A character string used to refer to a file stored on disk. A full file name is composed of the file name followed by the file extension, followed by the version number, e.g. "Solar System. NOTEFILE;2." Here the file name is "Solar System," the extension is "NOTEFILE," and the version number is "2." The file name and extension are separated by a period and the extension and version number are separated by a colon. See extension and version number .
file server	A computer that provides file storage and retrieval service for users on the network.
font	A collection of characters in one size and style of type, e.g., 10-point Modern Italic.
font family	A complete assortment of letters, numbers, punctuation marks, etc., of a given design, such as Modern or Classic.
font size	The distance from the top of the highest character in a font to the bottom of the lowest.
forward links	To-links and global to-links. This term is used when setting Browser and LinkIndex card parameters.
from-link	A link corresponding to every to-link and global to-link. From-links point from the destination card to the source card. All from-links are global.
global links	Links whose start point is the card itself and not some point in the card's contents.
global to-links	Links whose start point is a card itself and whose end point is another card. These links are not represented by link icons in a card's body. To see and access these link icons, you must choose the Show Links option from a card's left button title bar menu.
graph-based cards	Cards based on the graph editor. Graph and Browser cards are graph-based cards.
hard copy	The physical copy (on paper) of an on-screen document.
host	Any machine on a network. Often used to refer specifically to a machine that provides a network service, such as filing.
icon	An on-screen pictorial image which can represent some other object, such as card, window, or link.
image object	A graphic image, such as a Sketch drawing, graph, bit map, link icon, horizontal rule, etc.
initialization file	A file that is loaded when NoteCards is first started, to customize your environment according to your tastes and the idiosyncracies of your site.
justification	The uniform spacing of words in a line, so that the line ends flush with both margins.
label	A character string which does not represent a card in a Graph or Browser card. In the display below, "Planet" is a label.

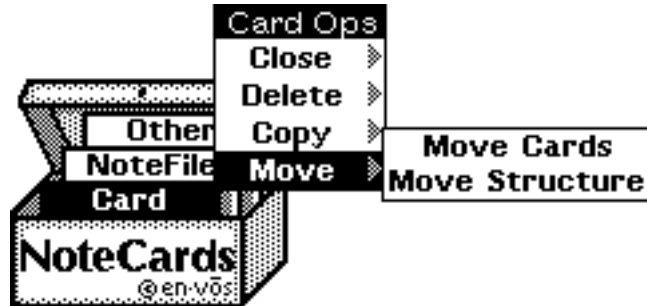
Error in IMAGEOBJ
GETFN: GRAPHOBJ.GETFN

line bar	The left part of a text card where the cursor shifts from being left-pointing to right-pointing allowing you to select larger units (paragraphs and lines) of text.
link	A connection between a source card and a destination card.
link icon	The graphical representation of the link. Clicking on a link icon traverses the link it represents and opens the destination card, or flashes it if it is already open.
link type	A word or phrase assigned by the user which classifies the relationship between two cards. Examples of link types include explanation, question, and answer.
local links	Links whose start point is somewhere in the contents of a card. Local links contrast with global links.
menu	A collection of text strings, buttons, or icons generally used to present a set of possible actions on the screen for user selection with the mouse.
MenuBox Icon	The icon from which you access the NoteCards system menus. Also referred to as the MenuBox.
mouse	A pointing device equipped with buttons.
mouse cursor	A small image (usually an arrow ) on the display screen that tracks the position of the mouse and lets you do things like reposition the caret. The cursor changes shape under certain conditions. An hourglass shape () indicates that a process is going on which may take some time to complete. A small image representing the mouse () which indicates that the system is waiting for a confirmation response before a selected process is performed.
node	Refers to a card or label in a Browser graph. The graph below shows four nodes, one of which (Moons) is a label.
note cards	Refers to non-FileBox cards, i.e., Text, Sketch, Graph, Browser, Search, LinkIndex, and Document cards. The generic term for cards of all types is just "card."
notefile	The file produced by the NoteCards system that contains the card index and cards.
notefile index	A fixed-size index of all the cards in the notefile. Its size must be explicitly changed.

network	An interconnection of several computers and other devices, such as printers, that lets them communicate and share resources. Sometimes also used to refer to the medium itself, such as a coaxial cable in the case of the Ethernet. Also, "net."
number pad menu	A calculator-style menu in which numbers can be entered to specify such values as line thickness.
path name	The complete name of a file you want to access. Includes the name of the host file server, device, directory, subdirectories, file name, extension, and version number. For example, in UNIX a notefile path name might be {dsk}/usr/users/kmount/nc/notefiles/demo.notefile, where dsk is the name of a device, usr is the directory, users, kmount, nc, and notefiles are the subdirectories, and demo.notefile is the file name of the notefile you want to access.
persistent menu	A menu that stays open until you close the menu or a window it is attached to. Also referred to as a "permanent menu" and "fixed menu."
pixel	A pixel is the smallest element of a display surface (a dot on a screen) which can be independently assigned a color or intensity. A blend of the words picture and element.
pop-up menu	A menu that appears when you press a mouse button and disappears when you release the button. Usually appears at the location of the cursor. The background menu is an example of a pop-up menu. See also menu and background menu .
prompt window	A small window attached to the top of a card, icon, or editor window where process information is printed and where you are asked questions. See also system prompt window .
scroll bar	The narrow window that opens when you move the cursor just outside the left or bottom edge of a window. You use the scroll bar to move through a body of information. The scroll bar frequently does not appear unless there is more information than can be displayed in the window.
sketch-based cards	Cards based on the Sketch graphical editor. There is only one sketch-based card and it is the Sketch card.
screen point	The size of a single pixel on the display. A screen point is 1/72 of an inch, normally.
select	This term has several meanings depending on the context of its use. When referring to menus, select means: Move the mouse cursor over the menu command, press the mouse button until the command is highlighted, then release the mouse button. You also select text (see current selection).
source card	A card containing a link icon.

submenu

A menu obtained by sliding to the right off an option on another menu. The existence of a submenu is indicated by a gray arrow head in the extreme right margin of a menu option. In the example below **Close**, **Delete**, **Copy**, and **Move** have submenus. In the case of **Move**, the user has slid the mouse off the right of the **Move** option and the **Move Cards/Move Structure** submenu has appeared.

**sysout**

A frozen version of the Notecards environment. It contains all the information needed to initialize virtual memory when Notecards is started.

system prompt window

The black window, usually at the top of the screen, used to display system information.

**text-based cards**

Cards based on the TEdit text editor which can also contain the output from sketch-based and graph-based cards. The text-based cards are Text, FileBox, Search, LinkIndex, and Document cards.

title bar

The black bar containing the window's title that appears at the top of a card or window.

to-links

Links whose start point is in a card's contents and whose end point is another card. These links are represented as link icons in a card's body.

type size

The distance from the top to the bottom of the characters that represent the highest and lowest points in a character set. Measured in points.

type-in process

The editor process associated with a card or window which is the destination for keyboard input.

version number

The number at the end of the file name that indicates when the file was created with respect to other files of the same name. The system assigns each successive file created with the same name a higher version number. The higher the version number, the newer the file.

virtual memory

Working space on the local disk that can be used, with the aid of swapping programs, to emulate random access memory.

window

A defined area within a display screen that can be used as a working space. Multiple windows can appear simultaneously on one display screen, and can overlay one another.

window menu The menu that appears whenever you press the right button with the cursor in the black title bar of a card or window, or at the top of a window with no title bar.

[This page intentionally left blank]

NOTECARDS PROGRAMMER'S INTERFACE

Introduction

This document describes a facility whereby users with some programming know-how can obtain a software interface to NoteCards in Interlisp. In this way, they can create and modify Notefiles, cards and links under program control.

The functions described below are divided into 8 groups:

1. NoteFile Creation and Access
2. Creating and Accessing NoteCard Types
3. Creating NoteCards and FileBoxes
4. Accessing NoteCards and FileBoxes
5. Creating and Accessing Links
6. Creating and Accessing Link Labels
7. Customizing the NoteCards Interface
8. Handy Miscellaneous Functions

Information for Prerelease Users

The two main changes to NoteCards from 1.2 are multiple open notefiles and cards stored as datatypes rather than atoms. The former change probably has a greater effect on users of the programmer's interface. Many functions now take a NoteFile argument that didn't before. In addition, many functions have been renamed or dropped altogether. Though we attempted to preserve some backward compatibility by keeping around old function names, we strongly recommend that you go over all your code and bring it up to date with this documentation. (For example, the whole notion of "Substance types" has been removed along with the Substance fns, etc.)

Other changes to watch out for include: traversal of NoteCards structures ala browsers; copying of groups of cards and their "internal" links across notefiles; new <quietFlg> args to many of the notefile access functions; undisplaying cards without uncaching; case insensitivity in NCP.TitleSearch; and registering of cards by key in the new notefile hash table. There are lots of handy new functions like NCP.ChangeCardTypeFields, NCP.CreateCardTypeStub, NCP.CollectCards, NCP.CopyCards, NCP.ListOfOpenNoteFiles, NCP.NumCardSlotsRemaining, NCP.ExpandNoteFileIndex, NCP.NoteFileMenu, NCP.NoteFileProp, NCP.WhichNoteFile, NCP.CreateLink, NCP.CardUserDataProp, NCP.DisplayedCards, NCP.RegisterCardByName, NCP.LookupCardByName, NCP.ListRegisteredCards, NCP.OpenCard, NCP.CloseCards, NCP.CacheCards, NCP.UncacheCards, NCP.DisplayCard,

NCP.UndisplayCards, NCP.AskYesOrNo, NCP.CardNeighbors,
NCP.MapCardsOfType, NCP.MapLinksOfType.

1. NoteFile Creation and Access

Note that some of the following accept filename arguments, some take notefile object arguments and some can take either. In any case, if the function accepts a filename, the .notefile suffix will be attached if not already present.

(NCP.CreateNoteFile <Filename><quietFlg>)

If <Filename> is not already a notefile, then creates a notefile with name "<Filename>.NoteFile", and returns this filename which can later be passed to NCP.OpenNoteFile. If <quietFlg> is non-nil, then communicative messages are held to a minimum.

(NCP.OpenNoteFile <NoteFileOrFilename> <don'tCreateFlg> <convertw/oConfirmFlg> <quietFlg> <menuPosition> <readOnlyFlg> <Don'tCreateInterfaceFlg>)

<NoteFileOrFileName> can be either a notefile object or a file name. Opens notefile, returning resultant notefile object if successful, else nil. If <don'tCreateFlg> is non-nil, then a new file is not created if the given one doesn't exist. If <convertw/oConfirmFlg> is non-nil, then if needed, the file is converted to the most recent format without user confirmation. If <quietFlg> is non-nil, then communicative messages are held to a minimum. If <menuPosition> is non-nil, then it should be a position at which to bring up the notefile main menu icon. If <readOnlyFlg> is non-nil, then the notefile is opened for read only. If <Don'tCreateInterfaceFlg> is non-nil, then don't bring up a control panel menu for the notefile.

(NCP.CloseNoteFiles <NoteFilesOrT> <quietFlg>)

If <NoteFilesOrT> is a list (or a single notefile), then close all open notefiles on that list (or the single one). If T, then close all open notefiles. If <quietFlg> is non-nil, then communicative messages are held to a minimum.

(NCP.CheckpointNoteFiles <NoteFilesOrT> <quietFlg>)

If <NoteFilesOrT> is a list (or a single notefile), then checkpoint all open notefiles on that list (or the single one). If T, then checkpoint all open notefiles. In case of a system crash or abort, the notefile can be recovered to the last checkpoint. Note that closing a notefile does a checkpoint. <quietFlg> non-nil will keep messages to a minimum.

(NCP.AbortNoteFiles <NoteFilesOrT> <Don'tConfirmFlg> <quietFlg>)

If <NoteFilesOrT> is a list (or a single notefile), then abort all open notefiles on that list (or the single one). If T, then abort all open notefiles. Aborting a notefile closes it and scraps all work since last checkpoint or successful close. <quietFlg> non-nil will keep messages to a minimum. <Don'tConfirmFlg> non-nil will prevent the asking of user for confirmation before throwing away work since last checkpoint.

(NCP.CompactNoteFile <fromNoteFileOrFilename> <toFilename> <inPlaceFlg>)

<fromNoteFileOrFilename> can be either a notefile or a file name. It is compacted, usually recovering space. If <inPlaceFlg> is non-nil, then <toFilename> is ignored and the compaction is in place. Will close <fromNoteFileOrFilename> if it's currently open.

(NCP.CompactNoteFileInPlace <NoteFileOrFilename>)

Compacts <NoteFile> in place, replacing the old version. Equivalent to (NCP.CompactNoteFile <NoteFile> NIL T).

(NCP.NumCardSlotsRemaining <NoteFile>)

Returns the total number of cards that can be created in <NoteFile> before its index must be expanded. <NoteFile> should be an open notefile.

(NCP.ExpandNoteFileIndex <NoteFile> <numNewCardSlots> <QuietFlg>)

Causes <NoteFile> (which should be a valid notefile) to be checkpointed and then have its index expanded in place to make room for <num NewCardSlots> new cards. If <NoteFile> is currently closed, then it is opened, expanded and then closed immediately.

(NCP.RepairNoteFile <NoteFileOrFilename> <readSubstancesFlg>)

Runs the Inspect&Repair facility on <NoteFileOrFilename>. It must *not* be currently open. If <readSubstancesFlg> is non-nil, then the inspector will check the contents of card substances rather than the simpler check of substance length. This means the first phase of the inspector takes MUCH longer so use with care. See the documentation on the Inspect&Repair facility for more information.

(NCP.DeleteNoteFile <NoteFileOrFilename> <Don'tConfirmFlg> <quietFlg>)

Deletes the <NoteFileOrFilename> notefile. Must not be open. <quietFlg> non-nil will keep messages to a minimum. <Don'tConfirmFlg> non-nil will prevent the asking of user for confirmation before deleting.

(NCP.NoteFileFromFileName <Filename>)

Returns the notefile corresponding to Filename if any, else nil.

(NCP.FileNameFromNoteFile <NoteFile>)

Returns the full name of the given notefile.

(NCP.NoteFileMenu <NoteFile>)

Returns the main menu for <NoteFile>.

(NCP.OpenNoteFileP <NoteFile>)

Returns non-NIL if <NoteFile> is a currently open notefile.

(NCP.ValidNoteFileP <NoteFile>)

Returns non-NIL if the notefile is a valid notefile.

(NCP.NoteFileClosingP <DontCheckForAbortFlg >)

Returns non-NIL if it has been called from under a close notefile operation. If DontCheckForAbortFlg is NIL, aborting the notefile counts as closing it. If DontCheckForAbortFlg is non-NIL, aborting the notefile is considered a different operation, and the function will return NIL.

(NCP.ListOfOpenNoteFiles)

Returns a list of all currently open notefiles.

(NCP.CheckOutNoteFile <fromFilename> <toFilename>)

Copies <fromFilename> to <toFilename> unless <fromFilename> is locked. If successful, creates a lock file in <fromFilename>'s directory. The name of the lock file is formed by concatenating the atom LOCKFILE onto <fromFilename>.

(NCP.CheckInNoteFile <fromFilename> <toFilename>)

Check lock file for <toFilename>. If none, then just copy <fromFilename> to <toFilename>. If there is one and it's owned by us, then do the copy and remove the lock file. If there is a lock file owned by someone else or if date of <toFilename> is more recent than date of lock file, then print a message and do nothing.

2. Creating and Accessing NoteCard Types

These functions give the user access to the NoteCard user-defined types facility. (In the functions below, card type arguments should be atoms.) For example, one of the simplest card types commonly created is what we call a "form" card. At card create time, it adds certain predefined text to the card's contents. Otherwise, form cards behave just like Text cards. Figure 1 shows the code needed to implement a sample form card type. For a full explanation of this facility, see the NoteCards Types Mechanism documentation.

FooForm.AddFooFormType

```
(LAMBDA NIL                                     (* rht; "10-Jul-88 11:41")

  (* * This creates the FooForm card type,)

  (NCP.CreateCardType (QUOTE FooForm)
    (QUOTE Text)
    (QUOTE ((MakeFn FooForm.MakeFn)))
    (QUOTE ((DisplayedInMenuFlg T)))))
```

FooForm.MakeFn


```
(LAMBDA (Card Title NoDisplayFlg)
  (* * The MakeFn for the FooForm card type,
    Note the call to the Text card's makefn,*)

  (PROG1 (NCP.ApplySuperTypeFn MakeFn Card Title NoDisplayFlg)
    (NCP.CardAddText Card (CONCAT
      "Some initial text for the FooForm card."
      (CHARACTER 13)
      ">>field1<<"
      (CHARACTER 13)
      "etc."))))
```

Figure 1: Sample card type definition

(NCP.CardTypes)

Returns list of all currently defined NoteCard types.

(NCP.CardTypeP <type>)

Returns non-nil if <type> is an existing NoteCard type.

(NCP.CardTypeFns)

Returns a list of the valid Fns fields for NoteCard types. Currently, these include: MakeFn, EditFn, QuitFn, GetFn, PutFn, CopyFn, MarkDirtyFn, DirtyPFn, CollectLinksFn, DeleteLinksFn, UpdateLinkIconsFn, InsertLinkFn and TranslateWindowPositionFn.

[The following hooks should some day become legitimate Fns or Vars fields for NoteCard types. Currently, they are properties placed on the card type's atom.

WhenSavedFn

If you need to have certain functionality happen just before a card's contents is saved to the notefile, then make a function be the value of the WhenSavedFn prop of the card type atom. It will be called whenever a card's contents is saved.

WhenDeletedFn

If you need to have certain functionality happen just before a card is deleted, then make a function be the value of the WhenDeletedFn prop of the card type atom. It will be called whenever a card is deleted. If the function returns 'ABORT, then the deletion of the card will be aborted.

LinkIconLeftButtonFn

One may now specify an operation other than TraverseLink to be used when left buttoning in link icons. Just put the property "LinkIconLeftButtonFn" with the value of a function on the (destination card's) card type atom. The function will get called with two args, the destination card and the window containing the link icon. (This is similar to the ExtraLinkIconMenuItems card type property consulted by the link icon middle button code.)

AttachedBitMapFn

You may now supply a function to calculate the attached bitmap for a card type. This function should be the value of the AttachedBitMapFn prop of the card type atom, and it will be passed Card, ScaledHeightToMatch, and Scale. If it exists, it will be applied first in calculating the bitmap to be displayed. If it returns a bitmap, a list of heights and bitmaps will be computed and placed

on the LinkIconAttachedBitMap field of the card type. This hook may also be used to calculate the bitmap to be displayed on the right side of a link icon if it is a cross-file link and its attached bitmap is being displayed. In this case, the function should be the value of the AttachedBitMapFn prop of the card type atom 'CrossFileLink. (For computing the list of heights and bitmaps in an AttachedBitMapFn, see NCP.MakeTypeIconBitMapSet at the end of this section.)

Don'tForceFilingFlg

It is possible to turn off forced filing on either the card type level or the card level. To define all cards of a type as not needing filing, you should put the property "Don'tForceFilingFlg" with the value of T on the card type atom. To define an individual card as not needing filing, use the function (NCP.MarkAsNotNeedingFiling <card>) (see Section 8: Handy Miscellaneous Functions).

NewCardPos

If present, the value of this property determines where new cards of the type are brought up on the screen, assuming the RegionOrPosition argument now accepted by all card type makefns (passed to NC.DetermineDisplayRegion) is NIL.]

(NCP.CardTypeVars)

Returns a list of the valid Vars fields for NoteCard types. Currently, these include: SuperType, StubFlg, FullDefinitionFile, LinkDisplayMode, DefaultWidth, DefaultHeight, LinkAnchorModesSupported, DisplayedInMenuFlg, LinkIconAttachedBitMap, LeftButtonMenuItems and MiddleButtonMenuItems.

(NCP.CardTypeFnP <fn>)

(NCP.CardTypeVarP <var>)

Returns non-nil if <fn> (<var>) is a valid function (variable) field for NoteCard types, for example, the litatom MakeFn (DefaultWidth). In other words, <fn> (<var>) can serve as the <fn> (<var>) arg to NCP.CardTypeFn (NCP.CardTypeVar).

(NCP.CardTypeFn <type> <fn>)

Returns the <fn> field for <type>. Note that this may be a value inherited at card type creation from <type>'s super type.

(NCP.CardTypeVar <type> <var>)

Returns the <var> field for <type>. Note that this may be a value inherited at card type creation from <type>'s super type.

(NCP.CardTypeSuper <type>)

Returns the super type of <type>. Equivalent to (NCP.CardTypeVar <type> 'SuperType).

(NCP.CreateCardType <TypeName> <SuperType> <FnsAssocList> <VarsAssocList>)

Makes a new NoteCard type with name <TypeName> and super type <SuperType>. Any functions not appearing in <FnsAssocList> or vars not appearing in <VarsAssocList> will be inherited from <SuperType>.

Note that, for now, specializing the FileBox card type is very dangerous and has unknown repercussions.

(NCP.DeleteCardType <TypeName> <DeleteSubTypesFlg >)

Deletes the NoteCard type with name <TypeName>. If DeleteSubTypesFlg is non-NIL, then it recursively deletes all sub-types of <TypeName>. If DeleteSubTypesFlg is NIL, then attempting to delete a type with sub-types is an error.

(NCP.CreateCardTypeStub <TypeName> <SuperType> <FullDefinitionFileName> <FnsAssocList> <VarsAssocList>)

Makes a stub for a new NoteCard type with name <TypeName> and super type <SuperType>. Some subset of the fns and vars can appear in <FnsAssocList> and <VarsAssocList>, however, the full definition will be loaded from <FullDefinitionFileName> the first time an attempt is made to access an undefined field name.

(NCP.ChangeCardTypeFields <TypeName> <FnsAssocList> <VarsAssocList>)

<TypeName> should be an existing NoteCard type. Some subset of the fns and vars should appear in <FnsAssocList> and/or <VarsAssocList>. These will be changed in the card type and then the inheritance mechanism will propagate these changes to any inheriting card types. For example, to make some existing card type FOO appears in the card type menu, do (NCP.ChangeCardTypeFields 'FOO NIL '((DisplayedInMenuFlg T))).

(NCP.ApplyCardTypeFn <CardTypeFn> <Card> <arg1> ...)

This macro applies the card type fn <CardTypeFn> (unevaluated) of the card type of <Card> to <Card> and the other args.

(NCP.ApplySuperTypeFn <CardTypeFn> <Card> <arg1> ...)

This macro applies the card type fn <CardTypeFn> (unevaluated) of the super type of <Card>'s type to <Card> and the other args.

(NCP.IsSubTypeOfP <type1> <type2>)

Returns non-nil if type1 inherits directly or indirectly from type2, that is, type2 can be found somewhere up the SuperType chain from type1.

(NCP.TextBasedP <cardOrType>)

(NCP.SketchBasedP <cardOrType>)

(NCP.GraphBasedP <cardOrType>)

If <cardOrType> is a card then we use its type. Returns non-nil if that type inherits directly or indirectly from Text, Sketch or Graph respectively.

(NCP.AutoLoadCardType <TypeName>)

<TypeName> is currently undefined. This asks NoteCards to look around for the file containing it and load it. It searches NOTECARDSDIRECTORIES for a file named NC<TypeName>TYPE.

(NCP.LinkIconAttachedBitMap <TypeName> <Size>)

Returns the link icon attached bitmap of size Size for card type TypeName. Default size is determined by the global variable NCP.DefaultLinkIconAttachedBitMapSize, whose initial value is 17.

(NCP.MakeTypeIconBitMapSet <Bitmap> <Heights>)

Returns a list of heights and bitmaps to be used in determining the appropriately sized bitmap for a given link icon. (Typically used when supplying an AttachedBitmapFn - see above.) <Bitmap> will be scaled to all of the heights in <Heights>, which defaults to NC.DefaultLinkIconAttachedBitMapHeights if NIL.

3. Creating NoteCards and FileBoxes

The following functions create various sorts of cards and boxes within the currently open notefile.

(NCP.CreateCard <type> <NoteFile> <title> <nodisplayflg> <props> <parentfileboxes> <otherargs><InterestedWindow> <RegionOrPosition>)

Creates and returns a card of the given (possibly user-defined) type, with given title, props, and parents. <otherargs> is a possibly nil list of args that will be passed to the MakeCardFn of <type>. Card is initially displayed or not according to value of <nodisplayflg>. Note that actually a top level copy of <props> is used (i.e. (APPEND <props>)). <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.CreateTextCard <NoteFile> <title> <nodisplayflg> <props> <parentfileboxes> <InterestedWindow> <RegionOrPosition>)

Creates and returns a new notecard having type Text. If <title> is non-nil, it is installed as the Notecard's title, otherwise the title is "Untitled." <props>, if non-nil, should be a prop-list of properties and values to be placed on the user property list of the Notecard. If <parentfileboxes> is non-nil, then it should be a list of FileBoxes in which to initially file this card. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.CreateFileBox <NoteFile> <title> <nodisplayflg> <props> <childcardsboxes> <parentfileboxes> <InterestedWindow> <RegionOrPosition>)

Creates and returns a new Filebox with title <title> (or a gensym'ed name if <title> is nil). It will initially contain child cards and boxes from the list <childcardsboxes> (if that arg is non-nil). If <parentfileboxes> is nil, then the new filebox will be filed in the value of (NCP.GetToBeFiledFileBox). The <props> arg is handled as it was for NCP.CreateNoteCard. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.CreateBrowserCard <NoteFile> <title> <paramList> <nodisplayflg> <props> <parentfileboxes> <InterestedWindow><RegionOrPosition>)

Creates and returns a new browser card with given title, props and parents. <paramList> should be a prop list of browser parameters. The properties currently recognized are:

ROOTCARDS	A list of Notecards to serve as roots of the forest or lattice generated by the browser. If omitted or NIL then user is asked to choose root cards.
LINKTYPES	A list of link types to follow when creating the browser. Any label present in the list having the backarrow prefix ("_") represents that link type but in the reverse direction. This list can also contain the atoms ALL or _ALL (<backarrow>ALL) in which case browsing will be done on all links in either the forward or reverse direction. If both ALL and _ALL (<backarrow>ALL) are specified, then links in both directions will be used (generally making a mess).
DEPTH	The depth at which to cut off the browser. This should be a non-negative integer. If NIL or omitted, then will assume no limit. (Currently integers greater than 9 are assumed equivalent to infinity.)
FORMAT	This should be a list of one, two or three elements. The first should be an atom indicating grapher format. The choices are FAST (layed out as a forest, sacrificing screen space for speed), COMPACT (layed out as a forest, using minimal screen space), LATTICE (layed out as a directed acyclic graph, the default), *GRAPH* (layed out as a graph, i.e. virtual nodes are eliminated). The second element of the FORMAT list, if present, should be either HORIZONTAL (the default) or VERTICAL specifying whether the graph is layed on its side or up and down. The third element, if present, should be the atom REVERSE. This indicates that horizontal graphs should be layed out from right to left instead of left to right and that vertical graphs should be layed out from bottom to top rather than vice versa.

If all of LINKTYPES, DEPTH, or FORMAT are omitted, the user is asked to choose them from a stylesheet. If one or more is specified, even as being NIL, the user is not prompted for them. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.CreateSketchCard <NoteFile> <title> <nodisplayflg> <props> <parentfileboxes> <InterestedWindow><RegionOrPosition>)

Creates and returns an initially empty sketch/map card having given title, props, and parents. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.CreateGraphCard <NoteFile> <title> <nodisplayflg> <props> <parentfileboxes> <InterestedWindow><RegionOrPosition>)

Creates and returns an initially empty graph card having given title, props, and parents. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.MakeDocument <NoteFile> <rootcard> <parametersProplist> <nodisplayflg> <props> <parentfileboxes><InterestedWindow> <RegionOrPosition>)

Creates and returns a Document card starting from <rootcard>. The user may specify new values for the set of parameters for making a document with <parametersProplist>. For example, a value of '(TitlesFromNoteCards Bold ExpandEmbeddedLinks ALL)

for <parametersProplist> would cause these values for the parameters TitlesFromNoteCards and ExpandEmbeddedLinks to be used, and the current defaults for the other parameters will be used. If no <parametersProplist> is provided, the user will be prompted for the parameters with a stylesheet. As usual, the resulting card will have the given props and parents. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

(NCP.MakeLinkIndex <NoteFile> <linktypes> <backpointersP> <nodisplayflg> <props> <parentfileboxes> <InterestedWindow> <RegionOrPosition>)

Creates and returns a LinkIndex text card consisting of a sorted record of all instances of links in the current notefile having one of the given link types. <linktypes> can contain the litatoms ALL and/or _ALL (<backarrow>ALL) as well as any particular backwards links. (See the above description of NCP.MakeDocument.) Backpointer links are inserted in the text if <backpointersP> is non-nil. Resulting card will have given props and parents. <InterestedWindow> is a window used to attach a prompt window for messages. <RegionOrPosition> is used to position and/or shape the new card.

4. Accessing NoteCards and FileBoxes

The following functions provide access to the cards and boxes present in the current notefile. Note that whether a card's window has been brought up on the screen has little or no impact on most of the following functions. If the user changes some field of a card while that card is visible on the screen, then the field will update itself automatically. Thus, users can switch between program-driven and screen-interface-driven modes at will.

Most of the following functions take as first argument a card or filebox. If this does not in fact correspond to an existing card or box, then an error message is printed and nil is returned.

Cards can be displayed, cached or closed. A cached card has its information cached in memory thus saving time (to access) at the expense of space. All cards displayed on the screen are also cached. In almost all cases, users will need only use NCP.OpenCard and NCP.CloseCards. NCP.OpenCard does both caching and displaying while NCP.CloseCards does both undisplaying and uncaching, if necessary.

(NCP.OpenCard <card> <region/position> <typeSpecificArgs>)

Brings up on the screen the given card in the given region or at the given position. If <region/position> is nil, then user is asked to specify position with mouse. <typeSpecificArgs>, if any, will be passed to the card type's EditFn.

(NCP.CloseCards <cardOrListOfCards> <quietFlg>)

Undisplay and uncache all the cards in <cardOrListOfCards>. <quietFlg> non-nil cuts down on messages.

(NCP.DisplayCard <card> <region/position> <typeSpecificArgs>)

If <card> was cached but not displayed, then bring it up on the screen. The <region/position> argument is as in NCP.OpenCard. <typeSpecificArgs>, if any, will be passed to the card type's EditFn.

(NCP.UndisplayCards <cardOrListOfCards> <quietFlg> <writeChangesFlg>)

If any of <cardOrListOfCards> were displayed, then undisplay them. <quietFlg> non-nil cuts down on messages. Normally, changes are not written to the notefile when a card is undisplayed, but only when it is uncached. <writeChangesFlg> non-nil makes changes be written through to the notefile.

(NCP.CacheCards <cardOrListOfCards>)

If any of <cardOrListOfCards> are not currently cached, then cache them.

(NCP.UncacheCards <cardOrListOfCards> <quietFlg>)

If any of <cardOrListOfCards> are cached but not displayed, then uncache them. <quietFlg> non-nil cuts down on messages.

(NCP.CardDisplayedP <card>)

Returns non-nil if given card or box is currently displayed in a window.

(NCP.CardCachedP <card>)

Returns non-nil if given card's information is currently cached.

Most of the following functions leave the card in the same state as it was when they started (except NCP.BringUpCard, which makes it active). Thus, users needing to do several consecutive operations to the same card should consider temporarily caching the card's information via NCP.CacheCards (and then uncaching with NCP.CloseCards).

(NCP.CardType <card>)

Returns the type of <card> or NIL if the card does not exist.

(NCP.ValidCardP <card>)

Returns <card> if <card> exists (hasn't been deleted), otherwise returns NIL. (This is currently a synonym for NCP.CardType.)

(NCP.SameCardP <card1> <card2>)

Returns non-nil if <card1> is the same card as <card2>. Error if either arg is not a valid card.

(NCP.NewCardP <card>)

Returns non-nil if <card> is new; ie. not yet saved in the notefile. Error if the arg is not a valid card.

(NCP.CardBeingDeletedP <card>)

Returns non-nil if <card> is in the process of being deleted; for use with user-defined card types. Error if the arg is not a valid card.

(NCP.CardTitle <card> [<newtitle>])

Returns old title of <card>. If <newtitle> is present, then set <card>'s title to <newtitle>. <newtitle> can be an atom or string. Note, however, that all titles are converted internally to strings by NoteCards.

(NCP.FileCards <cards> <fileboxes>)

Every card or box in <cards> is filed in every box in <fileboxes>. Either arg may be a card object or a list.

(NCP.UnfileCards <cards> <fileboxes>)

Every card or box in <cards> is unfiled from every box in <fileboxes>. Furthermore if <cards> is the litatom ALL, then the boxes in <fileboxes> will be cleared of all children. Similarly, if <fileboxes> is the litatom ALL, then the cards and boxes in <cards> will be unfiled from all their parent boxes. Either arg may be a card object or a list.

(NCP.CardParents <card> <FollowCrossFileLinksFlg>)

Returns list of fileboxes in which <card> is filed. <FollowCrossFileLinksFlg>, if non-nil, causes cross-file links to be followed to recover fileboxes in remote notefiles in which <card> is filed.

(NCP.FileBoxChildren <filebox> <FollowCrossFileLinksFlg>)

Returns list of children of <filebox> in the order in which they appear in the box's textstream. <FollowCrossFileLinksFlg> is as in NCP.CardParents.

(NCP.GetLinks <cards> <destinationCards> <labels> <NoteFile>)

Returns list of all links from any of <cards> to any of <destinationCards> having any label in <labels>. Any of these arguments can be nil. For example, if <destinationCards> is nil, then all links pointing from <cards> to anywhere with a label in <labels> are returned. If both <cards> and <destinationCards> are nil, then <NoteFile> should not be nil and this returns all links in <NoteFile> having a label in <labels>. If all three args are nil, then this is a slow synonym for (NCP.AllLinks <NoteFile>).

(NCP.GetCrossFileLinkDestCard <CrossFileLinkCard> <InterestedWindow> <Don'tOpenDestNoteFileFlg>)

For a given <CrossFileLinkCard>, tries to follow the link to a remote card in the destination notefile and returns that card if found. If <Don'tOpenDestNoteFileFlg> is non-nil, then destination notefile must be open. <InterestedWindow> is an optional window argument whose prompt window is used for messages.

Cross-file link cards are "hidden" cards that serve as placeholders for true cross-file links. That is, a (two-way) cross-file link from card A to card B (in different notefiles) actually consists of 2 links and 2 cross-file link cards. In the source notefile, card A is linked to a cross-file link card AA. AA "knows" about the notefile that contains B and B's UID. Similarly, the destination notefile contains a cross-file link card BB which is linked to card B. Again, the substance of BB contains a filename "hint" for the source notefile as well as the

UID for A. One-way cross-file links only contain cross-file link cards in the source notefile. The destination card contains no record that it's been linked to.

(NCP.CardNeighbors <cards> <linkTypes> <FollowCrossFileLinksFlg>)

Return a list of cards each of which is one link away from some card in <cards>. Only links having types in <linkTypes> are considered. Both <cards> or <linkTypes> can be either nil, card objects (or link type atoms) or a list. Null <cards> means that all cards linked to by anyone are returned. <linkTypes> can include link types prefixed with the character '_' (<backarrow>). This means to follow links of that type in the reverse direction. <linkTypes> can also include either or both of the atoms ANY and _ANY (<backarrow>ANY). The former means to include any cards one link away in the forward direction, while the latter causes inclusion of any cards one link away in the backwards direction. <linkTypes>=NIL is equivalent to <linkTypes>=ANY. If <FollowCrossFileLinksFlg> is non-nil, then any links that cross notefile boundaries are followed to their remote destination cards. Otherwise, cross-file links are ignored.

(NCP.CardPropList <card>)

Returns the prop list of the given card.

(NCP.CardProp <card> <propname> [<newvalue>])

Returns old value of property <propname> on <card>'s prop list. If <newvalue> is present, then set <card>'s <propname> property to <newvalue>. (Semantics are analogous to the Interlisp function WINDOWPROP.)

(NCP.CardAddProp <card> <propname> <newitem>)

Adds <newitem> to the list present on the <propname> property of <card>. Returns old value of property. (Semantics are analogous to WINDOWADDPROP.)

(NCP.CardDelProp <card> <propname> <itemToDelete>)

Deletes <itemToDelete> from the <propname> property of <card> if it is there, returning the previous value of that property. If not there, return nil. (Semantics are analogous to WINDOWDELPROP.)

(NCP.CardRegion <card> [<newRegion>])

Returns the region of <card>. This works even if <card> is not currently up on the screen, since the region information is stored on the notefile. If <newRegion> is provided, then the saved region of the card is changed. If the card is currently displayed, then it is reshaped to the new region.

(NCP.CardSubstance <card> [<newSubstance>])

Returns the substance of <card>. For example, returns a textstream in the case that the type of <card> is built on the TEdit text editor. In general, this is what the PutFn of a card type writes down to the notefile and what the GetFn reads in. If a <newSubstance> argument is present, then the substance of the card is replaced and NCP.MarkCardDirty is called.

(NCP.CardAddText <card> <textstr> <loc>)

Adds the text within the string <textstr> to the text card <card>. If <loc> is the litatom START or END, then the text will be placed at the start or end of the card respectively. If <loc> is a number, then it is assumed to be a character count within the card at which to place the new text. If <loc> is NIL, then the text is placed at the current cursor location.

(NCP.ChangeLoc <card> <loc>)

Changes the cursor's location in <card>'s textstream to <loc>. Possible values for <loc> are as described for NCP.CardAddText.

(NCP.DeleteCards <cardOrListOfCards>)

Deletes the given cards and fileboxes from their notefile, or deletes just the one if <cards> is a single card object.

(NCP.CardNoteFile <card>)

Returns <card>'s NoteFile.

(NCP.CardWindow <card>)

Returns <card>'s window if <card> is currently displayed somewhere on the screen. (The function NCP.WindowFromCard is an alias for NCP.CardWindow.)

(NCP.CardFromWindow <window>)

Returns the card associated with <window>, or NIL if not a notecards window.

(NCP.CardFromTextStream <TextStream>)

Returns the card associated with <TextStream>, or NIL if <TextStream> doesn't belong to some text card.

(NCP.FileBoxP <card>)

Returns non-nil if <card> is a filebox.

(NCP.AllCards <NoteFile>)

Returns a list of all extant cards for the given notefile.

(NCP.CardsOfTypes <CardsOrNoteFile> <Types>)

Returns a list of all cards of the given type (or types) for the given notefile (or list of cards).

(NCP.AllBoxes <NoteFile>)

Returns a list of all fileboxes in the given notefile.

(NCP.MapCards <NoteFile><fn> <collectResultsPredicate>)

Maps down the set of all cards in the current notefile, applying <fn> to each. If <collectResultsPredicate> is non-nil, then for those cards satisfying the predicate, the values of <fn> applied to them are collected.

(NCP.MapCardsOfType <types> <NoteFile> <fn> <collectResultsPredicate>)

This is similar to NCP.MapCards, but only looks at cards whose type appears on <types>. <types> can be a single type or a list of types.

(NCP.ContentsFileBox <NoteFile>)

(NCP.OrphansFileBox <NoteFile>)

(NCP.ToBeFiledFileBox <NoteFile>)

These functions retrieve the three predefined FileBoxes for the currently open NoteFile. These boxes can be modified (but not deleted) by the user in the same way as any other filebox.

5. Creating and Accessing Links

Links consist of source card, destination card, link type, display mode and anchoring mode. The new function NCP.CreateLink can be used to create any sort of link. We still provide the four functions NCP.GlobalGlobalLink, NCP.LocalGlobalLink, etc. for those who have grown used to that style.

(NCP.GetLinks <cards> <destinationCards> <labels> <NoteFile>)

See documentation in the previous section.

(NCP.CreateLink <source> <destination> <linkType> <displayMode>)

<source> can be either a card or a list of two elements (<sourceCard> <sourceLoc>). <sourceCard> should be a card to use as the source of the link while <sourceLoc> should be either the atom GLOBAL (in which case a global-to-global link is created) or a Loc directive as described in NCP.CardAddText above, that is, an integer or one of the atoms START, END or NIL. This creates and returns a new link with type <linkType>, connecting <sourceCard> to <destinationCard>. For text cards, Loc, if present, designates where to insert the link. If the link is local-to-global, then <displayMode> should be a valid displaymode or NIL. (See description of NCP.LinkDisplayMode for the valid values for <displayMode>.) (In the future, Locs for non-text cards will be specifiable. In the far future, we hope to allow local anchoring at the destination end of the link as well as the source.)

(NCP.GlobalGlobalLink <label> <sourceCard> <destinationCard>)

Creates and returns a new link with label <label>, connecting <sourceCard> to <destinationCard>.

(NCP.LocalGlobalLink <label> <sourceCard> <destinationCard> <fromloc> <displaymode>)

Creates and returns a new link with label <label>, connecting from <fromloc> of <sourceCard> card to <destinationCard>. If <displaymode> is non-nil, then the new link is displayed in the given mode. Otherwise the default displaymode for the source card's type is used. See the description of NCP.LinkDisplayMode for the valid entries for the <displaymode> arg.

(NCP.GlobalLocalLink <label> <sourceCard> <destinationCard> <toloc>)

Not implemented at this time.

(NCP.LocalLocalLink <label> <sourceCard> <destinationCard> <fromloc> <toloc>)

Not implemented at this time.

(NCP.LinkDesc <link> <followCrossFileLinkFlg>)

Returns list of three items (<label> <sourceDesc> <destinationDesc>) where <label> is the link type and <sourceDesc> and <destinationDesc> have the form (<anchor mode> <card> <loc>). <anchor mode> is either LOCAL or GLOBAL, <card> is the card at this end of the link, and <loc> gives a position in the text of <card> if <anchor type> is LOCAL and <card>'s substance's type is TEXT. If the link is a cross-file link and if <followCrossFileLinkFlg> is non-nil, then the link will be traversed, opening the remote notefile if necessary to determine information about the source or destination card of the link.

(NCP.LinkDisplayMode <link> [<newdisplaymode>])

Returns old display mode of <link>. If <newdisplaymode> is present, then set <link>'s displaymode accordingly. If non-nil, it can be an instance of the LINKDISPLAYMODE record. Or it can be one of the litatoms Icon, Title, Label, or Both. Finally, it can be a list of three elements (<ShowTitleFlg> <ShowLinkTypeFlg> <AttachBitmapFlg>). Each element can have one of the three values T, NIL, or FLOAT. If a field, say <ShowTitleFlg>, has value FLOAT then the corresponding global parameter (DefaultLinkIconShowTitle, in this case) will be consulted to decide whether or not to display the destination card's title in this icon. (See Section 7 for a description of the global parameters.)

(NCP.CoerceToLinkDisplayMode <thing>)

Returns a LINKDISPLAYMODE record. Thing can be a LINKDISPLAYMODE record, cardtype, card, link, atom, or list. If thing is a LINKDISPLAYMODE record, that record is returned. If thing is a cardtype, the default LinkDisplayMode for that cardtype is returned. If thing is a card, the LinkDisplayMode of that card is returned. If thing is a link, the LinkDisplayMode of that link is returned. If thing is an atom or a list, then the corresponding LinkDisplayMode, as specified under **NCP.LinkDisplayMode** (see above), is returned.

(NCP.LinkType<link> [<newLinkType>])

Returns old linktype of <link>. If <newLinkType> is present, set <link>'s type to <newLinkType>.

(NCP.LinkSource <link>)

Returns the card at the source end of <link>.

(NCP.LinkDestination <link>)

Returns the card at the destination end of <link>.

(NCP.DeleteLinks <links>)

Removes all links in <links> (or just one if <links> is a single link object).

(NCP.ValidLinkP <link>)

Error in IMAGEOBJ
1GETEN: HRULE.GETEN Returns non-nil if <link> is a link in the current notefile.

(NCP.SameLinkP <link1> <link2>)

Error in IMAGEOBJ
1GETEN: HRULE.GETEN Returns non-nil if <link1> is the same link as <link2>. Error if either arg is not a valid link.

(NCP.AllLinks <NoteFile>)

Error in IMAGEOBJ
1GETEN: HRULE.GETEN Returns a list of all existing links in <NoteFile>. (This is equivalent to but faster than (NCP.GetLinks NIL NIL NIL <NoteFile>).)

(NCP.MapLinks <NoteFile> <fn> <collectResultsPredicate>)

Error in IMAGEOBJ
1GETEN: HRULE.GETEN Maps down the set of all links in the given notefile, applying <fn> to each. If <collectResultsPredicate> is non-nil, then for those links satisfying the predicate, the values of <fn> applied to them are collected.

(NCP.MapLinksOfType <types> <NoteFile> <fn>)
<collectResultsPredicate>)

Error in IMAGEOBJ
1GETEN: HRULE.GETEN This is similar to NCP.MapLinks, but only looks at links whose type appears on <types>. <types> can be a single type or a list of types.

6. Creating and Accessing Link Labels

Error in IMAGEOBJ
1GETEN: HRULE.GETEN The following functions allow the user to manipulate link labels.

(NCP.CreateLinkType <linkType> <NoteFile> <QuietFlg>)

Error in IMAGEOBJ
1GETEN: HRULE.GETEN Creates a new link type with name <LinkType> in <NoteFile> unless there is already one defined by that name. A non-NIL <QuietFlg > suppresses the error message that is normally printed if the link type has already been defined for this notefile.

(NCP.DeleteLinkType <linkType> <NoteFile>)

Error in IMAGEOBJ
1GETEN: HRULE.GETEN Deletes the link type <linkType> from <NoteFile>. The link type must exist and must not be the type of any existing link, and it must not be a system-defined link type (e.g. SubBox or BrowserContents).

(NCP.RenameLinkType <linkType> <newLinkType>)
<NoteFile>)

Error in IMAGEOBJ
1GETEN: HRULE.GETEN Changes any links in <NoteFile> having link type <linkType> to have type <newLinkType>. <linkType> must exist and neither <linkType> nor <newLinkType> should be a system-defined type.

(NCP.LinkTypes <NoteFile>)

Error in IMAGEOBJ
1GETEN: HRULE.GETEN Returns a list of all existing link types in <NoteFile> including system-defined ones.

(NCP.ReverseLinkTypes <NoteFile>)

Returns a list of the reverse link types for every link type in <NoteFile>. Thus, whereas SubBox would appear in the list returned by NCP.LinkTypes, _SubBox (<backarrow>SubBox) would appear in the list returned by NCP.ReverseLinkTypes.

(NCP.UserLinkTypes <NoteFile>)

Returns a list of all existing user-defined link labels in <NoteFile>.

(NCP.SystemLinkTypeP <LinkType>)

Returns non-nil if <LinkType> is a system link type..

(NCP.ValidLinkTypeP <LinkType> <NoteFile>)

Returns non-nil if <LinkType> is a defined link type for <NoteFile>.

7. Customizing the NoteCards Interface

There are currently several areas where the user may tailor the NoteCards user interface: the NoteCards Session Icon menus, the left button menu on the title bar of a notefile's menu icon, the middle button menu of a notefile's menu icon, the ShowCards middle button menu on a notefile's menu icon, and the title bar menus of displayed cards and card types.

The Session Icon: The menus associated with this icon may be modified using the following functions:

(NCP.AddSessionIconMenuItem <MenuName> <Item>)

Adds the menu item <Item> to the session icon menu specified by <MenuName> (one of 'Card', 'NoteFile', or 'Other'). <Item> should be a complete menu item and may contain subitems in the standard format. For the Card and Other menus, the item can be a normal menu item. For the NoteFile menu, it must be slightly different. Instead of providing an expression to be evaluated as the second part of the menu item (and of any subitems), you should provide the name of a function of two arguments: NoteFile and Window. This function will be applied to these arguments when the menu item is selected.

Returns the item added if successful, NIL otherwise.

Example:

```
(NCP.AddSessionIconMenuItem ' NoteFile '(Foo% Function
NC.DoFoo "Performs the function Foo on this notefile") )
```

(NCP.RemoveSessionIconMenuItem <MenuName> <ItemName>)

Removes the menu item named <ItemName> from the session icon menu specified by <MenuName> (one of 'Card', 'NoteFile', or 'Other'). <ItemName> should be only the name of the menu item.

Returns the full item removed if successful, NIL otherwise

Example:

```
(NCP.RemoveSessionIconMenuItem ' NoteFile 'Foo% Function )
```

(NCP.RestoreSessionIconMenu <MenuName>)

Restores the menu specified by <MenuName> (one of 'Card', 'NoteFile', or 'Other') to its initial state. If <MenuName> is NIL, all three menus will be restored.

(NCP.SessionIconWindow)

Returns the session icon window.

(NCP.BringUpSessionIcon <IconPosition>)

Brings up the NoteCards icon at IconPosition. If no IconPosition is given and the icon is already on the screen, it will be flashed. If it is not on the screen, the user will be prompted to place it.

The left button menu on the title bar of a notefile's menu icons: The user may modify this menu using the following functions.

(NCP.AddNoteFileIconMenuItem <Item> <OpenOrClosedOrBoth>)

Adds the menu item <Item> to the NoteFile icon menu. <Item> should be a complete menu item and may contain subitems.. The second part of the menu item (and of any subitems) should be a function of two arguments: NoteFile and Window. Window will be the icon window for the notefile. This function will be applied to these arguments when this menu item is selected. <OpenOrClosedOrBoth> specifies for which type of notefile the new operation will be valid. If <OpenOrClosedOrBoth> is 'Open', the item will appear as an operation for open notefiles. If <OpenOrClosedOrBoth> is 'Closed', the item will appear as an operation for closed notefiles. If <OpenOrClosedOrBoth> is 'Both', the item will be available for both open and closed notefiles.

Returns the full item added if successful, NIL otherwise.

Example:

```
(NCP.AddNoteFileIconMenuItem '(Foo% Function NC.DoFoo
"Performs the function Foo on this notefile") 'Open)
```

(NCP.RemoveNoteFileIconMenuItem <ItemName>)

Removes the menu item named <ItemName> from the NoteFile icon menu.

Returns the full item removed if successful, NIL otherwise.

Example:

```
(NCP.RemoveNoteFileIconMenuItem 'Foo% Function )
```

(NCP.RestoreNoteFileIconMenu)

Restores the NoteFile Icon menu to its initial state.

The middle button menu of a notefile's menu icon: It is now possible for users to add menu items to this menu using the following functions.

(NCP.AddNoteFileIconMiddleButtonItem <Notefile><MenuItems>)

Adds list of menu items <MenuItems> to the middle button menu of the main menu icon corresponding to <Notefile>. These menu items will remain on the menu until the next time the notefile is closed. The second item of each menu item should be an atom that is the name of a function to be called when selected.

(NCP.AddDefaultNoteFileIconMiddleButtonItem <MenuItems>)

Adds list of menu items <MenuItems> to the middle button menu of the main menu icons for all notefiles. These menu items will remain on the menus for the life of the session. The second item of each menu item should be an atom that is the name of a function to be called when selected.

The ShowCards middle button menu on the notefile icon: Users can indicate that certain cards should be added to or deleted from this menu using the following functions.

(NCP.AddSpecialCard <Card>)

Adds <Card> to the list of special cards appearing in the middle button menu on the ShowCards option of the notefile icon (for the notefile containing <Card>).

(NCP.RemoveSpecialCard <Card>)

Removes <Card> from the list of special cards appearing in the middle button menu on the ShowCards option of the notefile icon (for the notefile containing <Card>).

The title bar menus of cards and card types: Users can add menu items to either a specific window containing a card, or to the default menu of a card type.

(NCP.AddTitleBarMenuItemsToWindow <Win> <Button> <NewMenuItems><TopOrBottom>)

Adds the menu items <NewMenuItems> to the title bar menu of <Win>. <Button> should be one of 'Left or 'Middle, corresponding to either the LeftButtonTitleBarMenu or the MiddleButtonTitleBarMenu. <Win> should be the window of a visible notecard. <TopOrBottom> should be one of 'Top or 'Bottom, indicating where in the menu the new items should appear.

(NCP.AddTitleBarMenuItemsToType <Type> <Button> <NewMenuItems> <TopOrBottom>)

Adds the menu items <NewMenuItems> to the title bar menu of all cards of type <Type>. <Button> should be one of 'Left or 'Middle, corresponding to either the LeftButtonTitleBarMenu or the MiddleButtonTitleBarMenu. <TopOrBottom> should be one of 'Top or 'Bottom, indicating where in the menu the new items should appear.

Menu of notefiles: the user may use the list and menu of noticed notefiles maintained by NoteCards.

NCP.NoticedNoteFileNames

Global var containing a list of the currently available notefile names noticed by NoteCards.

(NCP.NoticedNoteFileNamesMenu <IncludeNewNoteFileFlg> <AllowedOperations> <InterestedWindow> <Operation>))

Provides user with a menu of noticed notefile names. <IncludeNewNoteFileFlg> should be non-NIL if new notefiles are allowed. <AllowedOperations> should be one of the atoms: OPEN, CLOSED or NIL for both. <Operation> should be a string or atom containing the name of the operation to be performed on the result and the word NoteFile; e.g. (QUOTE Open% NoteFile). This is used in the prompt for a new notefile name. <InterestedWindow> is the window to receive a prompt window for any messages printed.

(NCP.ForgetNoteFileName <NoteFileOrFileName>)

The notefile is removed from the menu of noticed notefiles. It will not be added to the menu again until explicitly remembered using NCP.RememberNoteFileName regardless of other operations performed on the notefile.

(NCP.RememberNoteFileName <NoteFileOrFileName>)

The notefile is added to the menu of noticed notefiles. This is only necessary if the NoteFile name has been forgotten.

NCP.GrayShade

Global var containing the shade used for shading various menu items in the interface. The default value of NCP.GrayShade is GRAYSHADE. This variable should be changed with the function NCP.SetGrayShade (see below).

(NCP.SetGrayShade <Shade>)

This function should be used to change the value of NCP.GrayShade to <Shade>. In addition to changing NCP.GrayShade, it also resets all cached menus that use NCP.GrayShade. If <Shade> is NIL, NCP.GrayShade will not be changed, but the menus will still be reset. Returns the new value of NCP.GrayShade.

8. Handy Miscellaneous Functions

(NCP.BringUpNoteCardsIcon <IconPosition>)

Brings up the NoteCards session icon at the position <IconPosition>. If no position is given, will prompt the user to place the icon. If the icon is already on the screen and no position is given, it will be flashed. (equivalent to (NoteCards <IconPosition>).

(NCP.TitleSearch <NoteFile> <keys> <caseSensitiveFlg>)

Returns a list of all cards in <NoteFile> having all of the <keys> (can be atom, string or list) within their titles. Normally case insensitive unless <caseSensitiveFlg> is non-nil.

(NCP.PropSearch <NoteFile> <propOrPair> <propOrPair> ...)

Returns a list of all cards in <NoteFile> such that for every <propOrPair> arg, if it is atomic, then the card contains that property. If it is a list of two elements, then the card must have a property EQ to the first element with value EQ to the second element.

(NCP.WhichCard <x> <y>)

Returns the card currently displayed on the screen whose window contains the position in screen coordinates of <x> if <x> is a POSITION, the position (<x>,<y>) if <x> and <y> are numbers, or the position of the cursor if <x> is NIL. Returns NIL if the coordinates are not in the window of any card. If they are in the window of more than one card, then returns the uppermost. If <x> is a window, then NCP.WhichCard will return the card associated with that window. (The function NCP.WC is an alias for NCP.WhichCard.)

(NCP.WhichNoteFile <x> <y>)

Works just like NCP.WhichCard, returning the notefile corresponding to the indicated window. If the window is for a card, then the card's notefile is returned, if it's for a notefile menu, then

that notefile is returned. (The function NCP.WNF is an alias for NCP.WhichNoteFile.)

(NCP.NoteFileIconWindow <NoteFile>)

Returns the main menu icon window, if any, for given <NoteFile>.

(NCP.DisplayedCards <NoteFiles > <CardTypes >)

Returns a list of all cards currently displayed on the screen that are in one of <NoteFiles > and are of one of the types in <CardTypes >. (Shrunk ones are included.) If <CardTypes > is NIL, then looks at all card types. If <NoteFiles > is NIL, then looks at all notefiles.

(NCP.SelectCards <instigatingCardOrWindow> <singleCardFlg> <selectionPredicate> <message> <checkForCancelFlg> <NewCardFlg>)

Returns a list of those cards selected from the screen. A menu appears in or near <instigatingCardOrWindow> displaying <message> and having buttons for DONE, UNDO, CANCEL (unless <singleCardFlg> is non-nil, in which case there is only a CANCEL button) and NEW CARD (if <NewCardFlg> is non-NIL). Card selections must satisfy <selectionPredicate> and are made by left buttoning in the title bars of the desired cards. If user hits CANCEL button, then NIL is returned unless <checkForCancelFlg> is non-nil, in which case the atom DON'T is returned.

(NCP.DocumentParameters <parametersProplist>)

Returns the old value of the document parameters in the form of a proplist. If <parametersProplist> is non-nil then it should be a proplist whose properties are (some of the) valid document parameter names and whose values are permissible values for those parameters. The valid parameters and possible values are as follows:

HeadingsFromFileboxes	NumberedBold, UnnumberedBold, NONE.
TitlesFromNoteCards	Bold, NotBold, NONE.
BuildBackLinks	ToCardsBoxes, ToCards, ToBoxes, NONE.
CopyEmbeddedLinks	ALL, NONE, <listOfLinkLabels>.
ExpandEmbeddedLinks	ALL, NONE, <listOfLinkLabels>.

[See the Notecards user's manual for an explanation of these parameters and how their values affect the document created.]

(NCP.NoteCardsParameters <parametersProplist>)

Returns the old value of the global Notecards parameters in the form of a proplist. If <parametersProplist> is non-nil then it should be a proplist whose properties are (some of the) valid document parameter names and whose values are permissible values for those parameters. The valid parameters and possible values are as follows:

DefaultCardType	<legalCardType>
MenuLingersAfterNoteFileClose	T or NIL
ShowNoteFileOnCards	T or NIL

NewNoteFileInitialSize	<positive number (default is 1000)>
ForceFiling	T or NIL
ForceTitles	T or NIL
CloseCardsOffScreen	T or NIL
BringUpCardsAtPreviousPos	T or NIL
MarkersInFileBoxes	T or NIL
AlphabetizedFileBoxChildren	T or NIL
DefaultLinkIconAttachBitmap	T or NIL
DefaultLinkIconShowTitle	T or NIL
DefaultLinkIconShowLinkType	T or NIL
UseDeletedLinkIconIndicators	T or NIL
DelTEditProcessWhenShrinking	T or NIL
ExtraTEditProps	a prop list
EnableBravoToTEditConversion	T or NIL
IncludeCardObjectInShowInfo	T or NIL
LinkDashingInBrowsers	T or NIL
ArrowHeadsInBrowsers	one of the litatoms {AtEndpoint, AtMidpoint, None}
SpecialBrowserSpecs	T or NIL
DefaultFont	a font
LinkIconFont	a font
MenuFont	a font
NoteFileIndicatorFont	a font

Here, <legalCardType> should be an existing Notecard type, i.e. one that appears in the list returned by NCP.CardTypes.

(NCP.CoerceToInterestedWindow <WinOrCardOrNoteFile>)

Coerces a window, card or notefile into a window which can have a prompt window.

(NCP.PrintMsg <window> <clearFirstFlg> <arg1> <arg2> ...)

Prints a message in the prompt window of <window>. If <window> is NIL, then prints message in the Lisp prompt window. If <clearFirstFlg> is non-nil, then clears the prompt window first. The args are PRIN1'ed one at a time.

(NCP.ClearMsg <window> <closePromptWinFlg> <WaitMsecs>)

Clears the prompt window associated with <window> (or with the main Lisp prompt window if <window> is NIL) and closes it if <closePromptWinFlg> is non-nil. The prompt window will not be cleared before the specified wait has expired.

(NCP.AskUser <Msg> <prompt> <FirstTry> <ClearFirstFlg> <MainWindow> <DontCloseAtEndFlg> <DontClearAtEndFlg> <PROMPTFORWORDFlg>)

This function can be used to ask questions of the user in a window's prompt window. The <Msg> and <prompt> are printed along with <FirstTry> (if non-nil). The value returned is whatever the user types. If <ClearFirstFlg> is non-nil, then the prompt window is cleared first. If <MainWindow> is nil, then the top level prompt window is used. If <DontCloseAtEndFlg> is non-nil, then the prompt window won't be closed after the question is answered and if <DontClearAtEndFlg> is non-nil, then the prompt window won't be cleared at the end. If <PROMPTFORWORDFlg> is non-nil, then the PROMPTFORWORD typein protocol will be used rather than TTYIN. The former doesn't allow mouse editing of the string typed in. On the other hand, typing automatically overwrites the prompt when PROMPTFORWORD is used.

(NCP.AskYesOrNo <Msg> <prompt> <FirstTry> <ClearFirstFlg> <MainWindow> <DontCloseAtEndFlg> <DontClearAtEndFlg>)

This function can be used to ask yes/no questions of the user in a window's prompt window. The fields are as in NCP.AskUser except that PROMPTFORWORD is always used so there is no Flg for that.

(NCP.CardDates <Card>)

Returns a NOTECARDDATES record structure containing the dates of last modification of each of the four card parts of <Card>. The fields of the record are SUBSTANCEDATE, TITLEDATE, LINKSDATE and PROPLISTDATE.

(NCP.DetermineDisplayRegion <card> <region/position>)

Returns the region that <card> would occupy if displayed. If <region/position> is NIL, then asks the user to position a ghost region to get the position. Also checks previous size of card or default heights and widths for the card type to find the size. If BringUpCardAtPreviousPos is on, then won't require positioning of ghost region even if <region/position> is NIL.

(NCP.SetUpTitleBar <CardWindow> <CardType>)

This function is usually called from the MakeFn for those card types that inherit from a card type that doesn't display in a window, like the List or NoteCard type. It creates and installs left and middle menus using the menu items found in <CardType>'s definition. It also installs a default button event fn on the window.

(NCP.LinkFromLinkIcon <linkIcon>)

If <linkIcon> is an image object for a link icon, then returns the associated link. Note that link icons can be obtained by calling the card type's CollectLinksFn (see the card types mechanism documentation).

(NCP.MakeLinkIcon <link>)

If <link> is a valid link, then creates and returns a new link icon image object containing it.

(NCP.MarkCardDirty <card> <resetFlg>)

Mark <card>'s substance as dirty thus forcing it to be written down at the next save. If <resetFlg> is non-nil, then *unmark* <card> as dirty.

(NCP.MarkAsNotNeedingFiling <card>)

Mark <card> as not needing filing. This is done by setting the card's Don'tRequireFilingFlg prop to T.

(NCP.CoerceToCard <cardIdentifier>)

Return the card object (if any) associated with <cardIdentifier> which can currently be any of: a card object, window or text stream.

(NCP.CollectCards <RootCards> <LinkTypes> <MaxDepth> <FollowCrossFileLinksFlg>)

Starting from <RootCards> and following links having types in <LinkTypes> to a maximum depth of <MaxDepth>, collect and return a list of all cards encountered. A value of NIL for <MaxDepth> causes search to be "infinitely" deep. <LinkTypes> and <FollowCrossFileLinksFlg> are as described in NCP.CardNeighbors.

**(NCP.CopyCards <Cards> <DestNoteFileOrFileBox> <RootCards> <QuietFlg>
<CopyExternalToLinksMode> <InterestedWindow>)**

This copies all cards in <Cards> along with all links among them. If <CopyExternalToLinksMode> is 'COPY, external to-links will also be copied; if it is 'DON'TCOPY, external to-links will not be copied; and if it is NIL, the user will be asked if he/she wishes to copy these links. <DestNoteFileOrFileBox> designates a filebox in which to file the card copies. (If <DestNoteFileOrFileBox> is a notefile, then its Contents box is used.) <RootCards> should be a subset of Cards or NIL. If <RootCards> is NIL, then all the card copies are filed in the destination box, otherwise just those appearing in <RootCards>. If <RootCards> is the atom NONE, then none of the new cards will be filed in the destination filebox. <QuietFlg> cuts out the messages. Note that currently, <Cards> must all live in the same notefile, but this *can* be a different notefile from the destination notefile. <InterestedWindow> is a window used to attach a prompt window for messages.

**(NCP.MoveCards <Cards> <DestNoteFileOrFileBox> <RootCards> <QuietFlg>
<CopyExternalToLinksMode><InterestedWindow>)**

This moves all cards in <Cards> along with all links among them. If <CopyExternalToLinksMode> is 'COPY, external to-links will also be copied; if it is 'DON'TCOPY, external to-links will not be copied; and if it is NIL, the user will be asked if he/she wishes to copy these links. <DestNoteFileOrFileBox> designates a filebox in which to file the card copies. (If <DestNoteFileOrFileBox> is a notefile, then its Contents box is used.) <RootCards> should be a subset of Cards or NIL. If <RootCards> is NIL, then all the card copies are filed in the destination box, otherwise just those appearing in <RootCards>. If <RootCards> is the atom NONE, then none of the new cards will be filed in the destination filebox. <QuietFlg> cuts out the messages. Note that currently, <Cards> must all live in the same notefile, but this *can* be a different notefile from the

destination notefile. <InterestedWindow> is a window used to attach a prompt window for messages.

The following functions allow users to register cards by name for fast access. This is normally done using the card's notefile's system registry. Thus this registering is preserved over notefile closing.

(NCP.RegisterCardByName <Name> <Card> <RegistryCard>)

Stores <Card> in <RegistryCard> hashed under the name <Name>. If <RegistryCard> is nil, then use <Card>'s notefile's system registry.

(NCP.LookupCardByName <Name><NoteFileOrRegistryCard>)

If <NoteFileOrRegistryCard> is a registry card, then recovers the card that was hashed under <Name>. If <NoteFileOrRegistryCard> is a notefile, then use its system registry.

(NCP.UnregisterName <Name><NoteFileOrRegistryCard>)

If <NoteFileOrRegistryCard> is a registry card, then smashes whatever was stored under <Name>. If <NoteFileOrRegistryCard> is a notefile, then use its system registry.

(NCP.ListRegisteredCards <NoteFileOrRegistryCard> <IncludeKeysFlg>)

If <NoteFileOrRegistryCard> is a registry card, then returns the list of cards hashed in it. If <NoteFileOrRegistryCard> is a notefile, then use its system registry. If <IncludeKeysFlg> is non-nil, then return list of cons pairs with car=Name and cdr=Card.

One can associate functionality with the opening and closing of notefiles using two special List cards registered under the names OpenEventsCard and CloseEventsCard. Their substances each consist of a list of lisp s-expressions to be evaluated at notefile open and close time respectively. Thus, to cause a new expression to be evaluated at notefile open time, one cons's (or appends) the expression to the substance of OpenEventsCard (either via DEdit or NCP.CardSubstance). Note that during the evaluation of the expressions, the atom NoteFile is bound to the notefile being opened or closed. When a notefile is closed, the expressions in the CloseEventsCard are evaluated both just before the notefile is closed, and after it is finished being closed. In this case, the atom When is bound to either 'Before or 'After, indicating when the expressions are evaluated with respect to the close operation.

(NCP.GetOpenEventsCard <NoteFile>)

Returns the open events card for <NoteFile> creating a new one if necessary. Thus, this is basically just (NCP.LookupCardByName 'OpenEventsCard <NoteFile>), except that a new List card is created and registered under the name OpenEventsCard if none exists.

(NCP.GetCloseEventsCard <NoteFile>)

Returns the close events card for <NoteFile> creating a new one if necessary. Thus, this is basically just (NCP.LookupCardByName 'CloseEventsCard <NoteFile>), except that a new List card is

created and registered under the name CloseEventsCard if none exists.

Sometimes, it's nice to have a place to temporarily hang your hat, so to speak. For this purpose, we provide what are called *UserProps* for both card and notefile objects. These are temporary in that they vanish when the notefile closes.

(NCP.CardUserDataProp <Card><Prop> [<NewValue>])

Returns the value under the <Prop> user data prop for <Card>. If <NewValue> is present, then value under <Card>'s <Prop> is reset. (That is, it works like WINDOWPROP or NCP.CardProp.)

(NCP.NoteFileProp <NoteFile><Prop> [<NewValue>])

Returns the value under the <Prop> user data prop for <NoteFile>. If <NewValue> is present, then value under <NoteFile>'s <Prop> is reset. (That is, it works like WINDOWPROP or NCP.CardProp.)

(NCP.NoteFileAddProp <NoteFile><Prop> [<NewValue>])

Returns the value under the <Prop> user data prop for <NoteFile>. If <NewValue> is present, then it is added to the list under <NoteFile>'s <Prop>. If <NoteFile> has no value under <Prop>, then <NewValue> is placed on that property as a list. (That is, it works like WINDOWADDPROP.)

Error breaks/messages under the programmer's interface: When the programmer's interface detects an error, it calls NCP.ReportError or NCP.ReportWarning depending on the severity of the error. Normally the former causes a break while the latter merely prints a message. However, this behavior can be changed using the global var NCP.ErrorBrkWhenFlg, a litatom whose value should be one of NIL, ALWAYS or NEVER (default is NIL).

(NCP.ReportError <Function><Message>)

Forces a break using BREAK1 unless the value of NCP.ErrorBrkWhenFlg is the litatom NEVER. If so, then simply prints a message and continues.

(NCP.ReportWarning <Function><Message>)

Prints the given warning message <Message> unless the value of NCP.ErrorBrkWhenFlg is the litatom ALWAYS. If so, then forces a break with BREAK1.

There are a few old functions preserved for backward compatibility, but we encourage users to rewrite all their code to use only functions described above. The old functions still available are:
NCP.BringUpCard, NCP.ActivateCards, NCP.ActiveCardP,
NCP.DeactivateCards, NCP.ValidCard, NCP.GetContentsFileBox,
NCP.GetOrphansFileBox, NCP.GetToBeFiledFileBox,
NCP.GetLinkSource, NCP.GetLinkDestination,
NCP.CreateLinkLabel, NCP.DeleteLinkLabel,
NCP.RenameLinkLabel, NCP.GetLinkLabels,
NCP.GetUserLinkLabels, NCP.GetReverseLinkLabels,
NCP.ValidLinkLabel, NCP.AddLeftButtonTitleBarMenuItems,
NCP.AddMiddleButtonTitleBarMenuItems.

[This page intentionally left blank]

Bitmap Editor

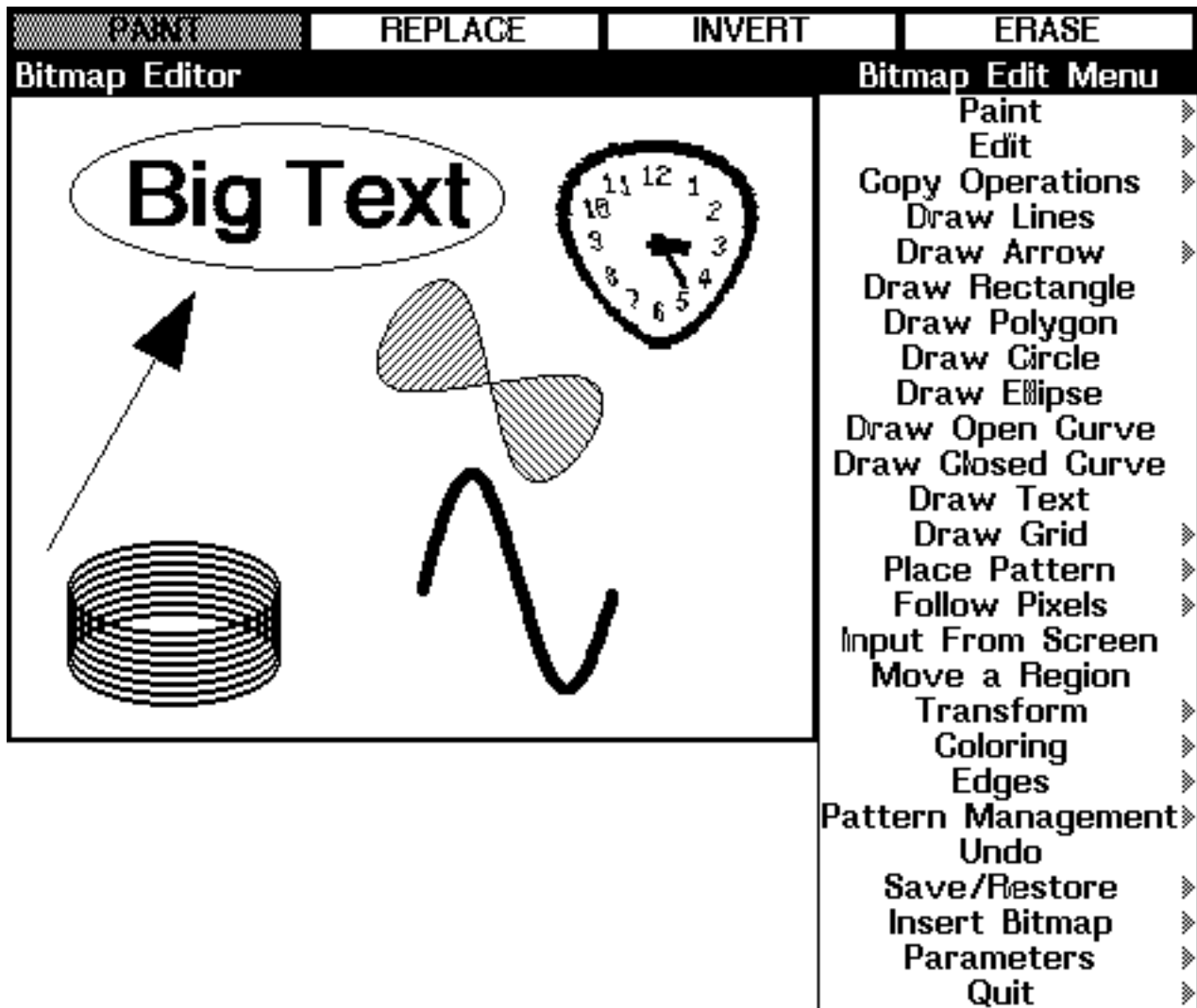
**Vista Laboratory
Xerox Special Information Systems**

November, 1988

The bitmap editor is invoked as follows.

(EditBitmap BitmapOrFile Window ...)

If BitmapOrFile is an existing bitmap, the user will be prompted to place a window displaying the existing bitmap. If BitmapOrFile represents a file containing a bitmap in binary form, it will be read from the file and subsequently edited via the window. If BitmapOrFile is non-NIL and Window is given, the bitmap will be edited within the existing window although it may be clipped if the bitmap is larger than the window. Finally, if both BitmapOrFile and Window are NIL, the user will be prompted to sweep out a region on the screen that will define the bitmap. The editing window is then placed as desired.



The four menu items at the top of the editor window set the primary drawing operation. Paint logically ORs the drawing bits with the bitmap bits to give the result. In Replace mode, all drawing bits replace the bitmap bits whether they are on or off. Invert does a logical exclusive OR between the drawing bits and the bitmap bits. Erase preforms a

logical AND between the inverse of the drawing bits and the destination bits. The drawing operation may also be changed via Parameter menu subitems.

A brief description of all the menu items follows.

Abort - Error out of the bitmap editor. Equivalent to a control-E.

Add Border - The bitmap is expanded by adding a border with a user specified width and texture.

Add Texture - Adds a user selectable texture to the entire bitmap.

Add Pattern - Allows the user to define a pattern (an independent bitmap) that may be placed in the primary bitmap, used for painting, etc.

Air Brush Size - Sets the size of the air brush.

Air Brush Speed - Sets the speed of the air brush. The affects the density of "spray."

Apply Operation to Copy and Original - Applies the primary bitmap operation (Paint, Replace, Invert, or Erase) to the original bitmap and an independent copy that is created and maintained by the primary editor. The copy may be separately edited.

Arrowhead - Sets the width and height of the arrowhead created by the Draw Arrow command.

Auto-Save File Name - Sets the name of the file to be used for auto-saving of the bitmap.

Auto-Save Interval - Sets the interval between automatic saving of the bitmap.

Auto-Save Status - Displays the current auto-save status, file name, and interval.

Averaging - Applies an averaging matrix to the entire bitmap. The weighted sum of each pixel and its eight neighbors is compared with a threshold. If the sum is greater than the threshold, the pixel is turned on (or left on), else it is turned off. The user may use a default averaging scheme or define her own in a Lisp executive.

Bitmap Editor Paint with Airbrush - Allows painting of the bitmap with an "air brush." The size, shape, and speed of the air brush may be changed via subitems of the Parameters menu item. The operation (Paint, Replace, Invert, or Erase) and the speed may be changed while in this mode by clicking the right mouse button and selecting the appropriate menu item.

Bitmap Editor Paint with Brush - Similar to the standard Interlisp-D paint facility. The size and shape of the brush are modified via subitems of the Parameters menu item. The operation (Paint, Replace, Invert, or Erase) may be changed while in this mode by clicking the right mouse button and selecting the appropriate menu item.

Bitmap Editor Paint with Pattern - Similar to Bitmap Editor Paint with Brush, but paints with a pattern (arbitrary bitmap) instead of a standard brush.

Brush Shape - Sets the brush and air brush shape. The available choices are round, square, horizontal, vertical, and diagonal.

Checkpoint Restore - Resets the bitmap to the state it was in just prior to the last Checkpoint Save.

Checkpoint Save - Saves the current bitmap locally. The saved bitmap is not written to a file, but merely saved as a window property. Each Checkpoint Save destroys the previous saved bitmap.

Clear - Clears the entire bitmap.

Clear All But Region - Clears the entire bitmap except for a region swept out by the user.

Clear Region - Clears a region swept out by the user.

Coloring - Menu subitems change coloring of the bitmap and regions.

Copy Operations - Menu subitems create and edit a copy of the bitmap.

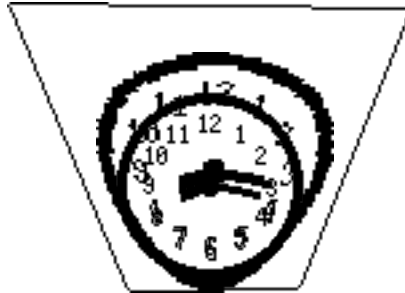
Copy/Edit Pattern - Creates a copy of an existing pattern and opens an edit window on the pattern.

Create New Copy of Bitmap - Creates a copy of the existing bitmap and maintains it separately. The copy may be edited separately.

Dashing - Turns dashing on or off and sets the style. When dashing is on, all line drawing operations will be done with the existing dashing pattern.

Delete Pattern - Allows the deletion of one of the patterns currently defined for the bitmap editor.

Distort Region - Distorts a rectangular region of the bitmap. This operation is time consuming on an 1186 which does not have hardware floating point. The operation operates as follows. A rectangular region is first swept out by the user as in the figure to the left. The vertices are then moved with the left button resulting in a display as in the middle figure. The result is shown in the rightmost figure. Note that if the user wishes to replace the region with the distorted result, the editor must be in the REPLACE mode. Otherwise, the distorted region will be PAINTed over the original region.



Draw Arrow - Draws one or more line segments with an open arrow at the end point. The size of the arrow may be changed by setting the appropriate parameters.

Draw Circle - Draws a circle in the same manner as SKETCH. The circle will be drawn with the current drawing brush size and dashing.

Draw Closed Curve - Draws a closed curve in the same manner as SKETCH. The curve will be drawn with the current drawing brush size and dashing.

Draw Ellipse - Draws an ellipse in the same manner as SKETCH. The ellipse will be drawn with the current drawing brush and dashing.

Draw Filled Arrow - Draws one or more line segments with a filled arrow at the end point. The size of the arrow may be changed by setting the appropriate parameters.

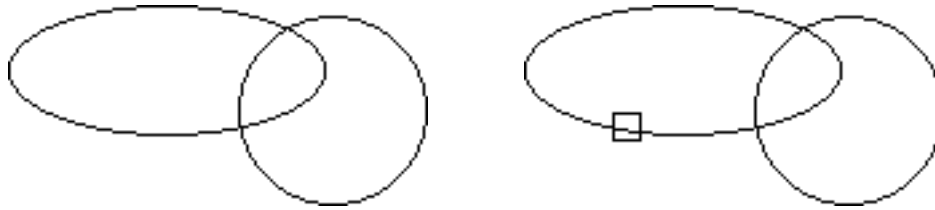
Draw Grid - Draws a grid over the entire bitmap. The width and height of the grid is specified via a number menu. The grid will be drawn with the current drawing brush size and dashing.

Draw Grid in Region - Draws a grid in a specified region. The width and height of the grid is specified via a number menu. The grid will be drawn with the current drawing brush size and dashing.

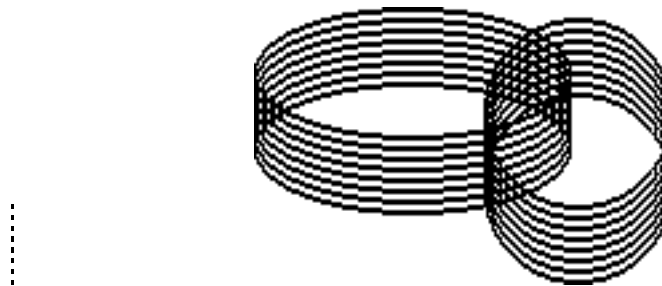
Draw Lines - Draws one or more line segments in the same manner as SKETCH. The lines will be drawn with the current drawing brush size and dashing.

Draw Open Curve - Draws a closed curve in the same manner as SKETCH. The curve will be drawn with the current drawing brush size and dashing.

Draw Over Connected Pixels from Region - Draws over all "connected" pixels starting from one or more pixels in a small region. Drawing may be done with the current drawing brush or a pattern. A pixel is connected to another pixel if it is one of the eight neighbors of the pixel. The operation is as follows. Suppose we have a display as shown to the left. We then select starting pixels by placing the box cursor as shown in the right figure.



Suppose we then select a pattern as shown to the left below. The resulting display will appear as in the figure to the right. Note that the user is requested to select the offset of the drawing brush or pattern.



Draw Over Connected Pixels from Selected Pixel - Similar to Draw Over Connected Pixels from Region, but a single pixel is selected as the starting point. The user places a separate "magnifier" window. When the cursor enters the bitmap editor, it changes to rectangular crosshairs and the surrounding 64 pixels are magnified 8 times and displayed in the magnifier window. When the crosshairs are over the selected pixel, the left button is clicked to specify the starting pixel.

Draw Polygon - Draws a polygon in the same manner as SKETCH. The polygon will be drawn with the current drawing brush size and dashing.

Draw Rectangle - Draws a rectangle in the same manner as SKETCH. The rectangle will be drawn with the current drawing brush size and dashing.

Draw Text - Places text. The font family, size, and style may be changed via Parameters menu subitems. The text is typed into a prompt window and then moved to the desired position with the mouse.

Drawing Brush Size - Sets the size of the drawing brush.

Edges - Menu subitems trim the bitmap and add borders.

Edit - Menu subitems edit regions of the bitmap.

Edit Copy of Bitmap - Opens an edit window on the currently stored copy of the bitmap. See Create New Copy of Bitmap.

Edit Pattern - Opens an edit window on a selected pattern.

Edit Region - Opens an edit window on a selected rectangular region of the bitmap.

Erase Connected Pixels from Region - Erases all "connected" pixels starting from one or more pixels in a small region. A pixel is connected to another pixel if it is one of the eight neighbors of the other pixel.

Erase Connected Pixels from Selected Pixel - Similar to Erase Connected Pixels from Region, but a single pixel is selected as the starting point. The user places a separate "magnifier" window. When the cursor enters the bitmap editor, it changes to rectangular crosshairs and the surrounding 64 pixels are magnified 8 times and displayed in the magnifier window. When the crosshairs are over the selected pixel, the left button is clicked to specify the starting pixel.

Exact Size - Allows changing the size of the current bitmap by explicitly resetting the origin, width, and height in terms of the bitmap coordinates.

Expand - Expands the width and/or height of the bitmap by an integer factor of one to four.

Fill Box - Fills a specified rectangular region with the default texture.

Fill Box with Specified Color - Fills a specified rectangular region with a chosen texture.

Fill Region(s) - Fill one or more closed regions with the default texture.

Fill Region(s) with Specified Color - Fill one or more closed regions with a chosen texture.

Follow Pixels - Menu subitems follow connected pixels and erase or draw over them.

Font - Sets the font family, size, and style for text placement.

Get Fro

BOONE-V-COE

Plaintiffs, W. H. Boone and J.T. Coe, brought this action against defendant, J.F. Coe, to recover certain damages alleged to have resulted from defendant's breach of a parol contract of lease for one year to commence at a future date. It appears from the petition that the defendant was the owner of a large farm in Ford Co., Texas. Plaintiffs were farmers living in Monroe Co., Kentucky. In the fall of 1909, defendant made a verbal contract with plaintiffs, whereby he rented to them his farm in Texas for a period of 12 months, to commence from the date of plaintiff's arrival at defendant's farm. Defendant agreed that if plaintiffs would leave Kentucky and move to defendant's farm in Texas and take over the farm for 12 months after their arrival, the defendant would prepare a home on the farm for plaintiffs which they could occupy as residence. Defendant also agreed

The plaintiffs did move from Kentucky during the fall of 1909. Upon their arrival at the farm in December, defendant denied them entry to the farm. Plaintiffs then left, returning to Kentucky. In reliance on the contract for the farm, plaintiffs incurred costs of \$1400.

Defendant's demurred to the petition for damages which was sustained. Plaintiffs appeal. We concur with the previous decision.

.....

CrossFile Links Destination NoteFile Browser

By: Daniel Jordan (Jordan.pa@Xerox.COM)

Stored: {qv}<notecards>1.3L>Library>DestNoteFileBrowser, .lcom, .ted
and {nb:parc:xerox}<notecards>1.3L>Library>DestNoteFileBrowser, .lcom, .ted
Written: October 10, 1988 by Dan Jordan
Last updated: January 6, 1989 by Dan Jordan
Supporting files: (all NC patches) dsjpatch075, dsjpatch076, rarpatch026

Introduction to the Destination NoteFile Browser

The Crossfile Links Destination Notefile Browser allows the user to modify and manage the destination notefiles associated with crossfile links in (source) notefiles. Using this facility, the user can easily move groups (or subgroups) of cross-linked notefiles to different directories and keep the cross-linked information between them up to date.

This facility features a "destination notefile browser" that displays in a window all destination notefiles for all crossfile links in a particular (source) notefile or group of notefiles. This browser is similar in appearance to a "file browser". Through a destination notefile browser, the directories or entire names of the destination notefiles associated with any crossfile links may be changed, to reflect changes in the location or name, respectively, of the actual destination notefile themselves.

Loading the Destination NoteFile Browser Package

Loading the Crossfile Links Destination Notefile Browser package is done in the same way as loading any other NoteCards library package. For example, you can type the following in an Interlisp exec:

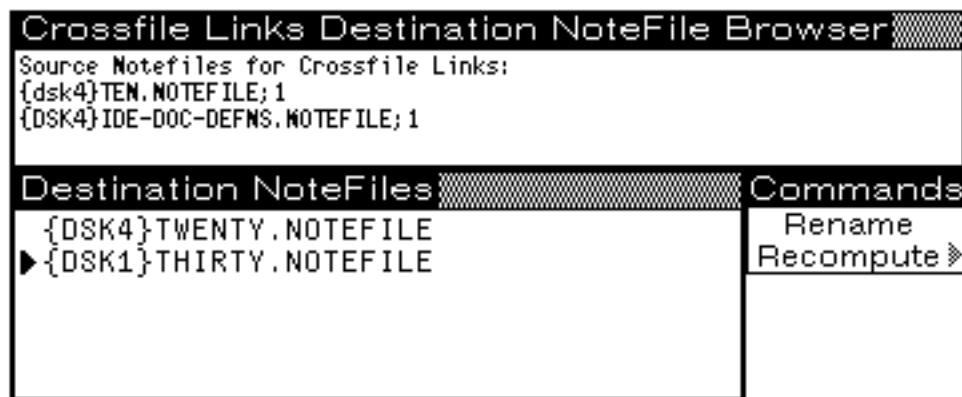
```
(FILESLOAD (FROM NOTECARDS) DESTNOTEFILEBROWSER)
```

Alternatively, you can add the atom DESTNOTEFILEBROWSER to the global var NOTECARDSLIBRARYFILES in your init file. This will ensure that DESTNOTEFILEBROWSER is loaded when bringing up a fresh sysout. (Note that the supporting patch files are to be found on the directory specified by the global var NOTECARDSNEXTDIRECTORIES.)

Using the Destination NoteFile Browser

Once the package is loaded, the user can access the Crossfile Links Destination Notefile Browser facility by selecting the "Destination NoteFile Browser" item on the "Other" menu item on the Notecards main menu. The user will be prompted to specify a region for a window and a "destination notefile browser" window will be opened in that region. The user will then be prompted to select the source notefiles that are to be used as the basis for the computed set of crossfile link destination notefiles. That is, for the set of source notefiles selected, ALL crossfile link destination notefiles for ALL crossfile links in the source notefiles will be computed. The source notefiles will be displayed in the top window of the browser, and any resultant destination notefiles for this set of source notefiles will be displayed in the main browser window.

The directories or entire names of any of the destination notefiles may be changed by selecting the destination notefiles of interest as you would items in a "file browser". There are two commands available on a destination notefile browser:



Rename

The user will be prompted for a new name. The selected destination notefiles of all crossfile links for this group of source notefiles will have their names changed to this new name. If the new name is just a directory name, e.g., {qv}<ide>notefiles>, all selected destination notefiles will only have their directories changed and will retain their (root) filenames. If the new name is a complete filename (with directory), e.g., {qv}<ide>notefiles>new.notefile, all selected destination notefiles will have their names changed to exactly that new name. The browser will automatically be updated to reflect the destination notefile names.

Recompute

This command is used to update, or recompute, the information in the browser (e.g., after new crossfile links have been added to the source notefiles.) There are two subitems on this menu: **Same source notefiles** (the default) and **New source notefiles**. The browser will be recomputed with the same source notefiles, unless

New source notefiles has been selected, in which case the user will be prompted for a new set of source notefiles.

NoteCards Library Packages

Stored as: {nb:parc:xerox}<notecards>1.3m>library>librarydoc.ted

ACE animation package	Denber
Animation-Cards	Bagley
Animation2	Bagley
Areafill	Bagley
BitmapEditor	Gaska
CaseCardFunctions	Cole
DestNoteFileBrowser	Jordan
LexDemo	Todd/Newman
NCBitmapCard	Gaska
NCCaseCluster	Trigg
NCChain	Trigg
NCClusterCard	Trigg
NCCollaboration	Trigg
NCCollaboratorCard	Trigg
NCDemo	Trigg
NCDraftCard	Rao
NCFunctionCard	Halasz
NCFullTextSearch	Todd
NCGestures	Trigg
NCGuidedTourCard	Trigg/Cohen/Ir
ish	
NCHacks	[Many]
NCHistoryCard	Irish
NCIdeaSketchCard	?
NCInspectorCard	Trigg
NCKey	Rao
NCLogger	Trigg/Irish
NCMailCard	Kelley
NCMaps	Feuerman
NCMergeFiles	Halasz
NCPath (NCPathLanguage, NCPathParse, NCPathUse)	Newman
NCPlotCard	Jordan
NCScreen	Trigg
NCSeditCard	Kelley
NCShadedCrossFileLinks	Irish
NCSentences	Halasz
NCSpreadSheetCard	Trigg
NCStat	Trigg
NCStructEditCard	?
NCStructEditBrowserCard	Trigg

NCTableTopCard
ohen/Irish
NCToulminCard
ShapeOfIt
TextCardKeys

Shrager/Trigg/C

Irish
Kelley

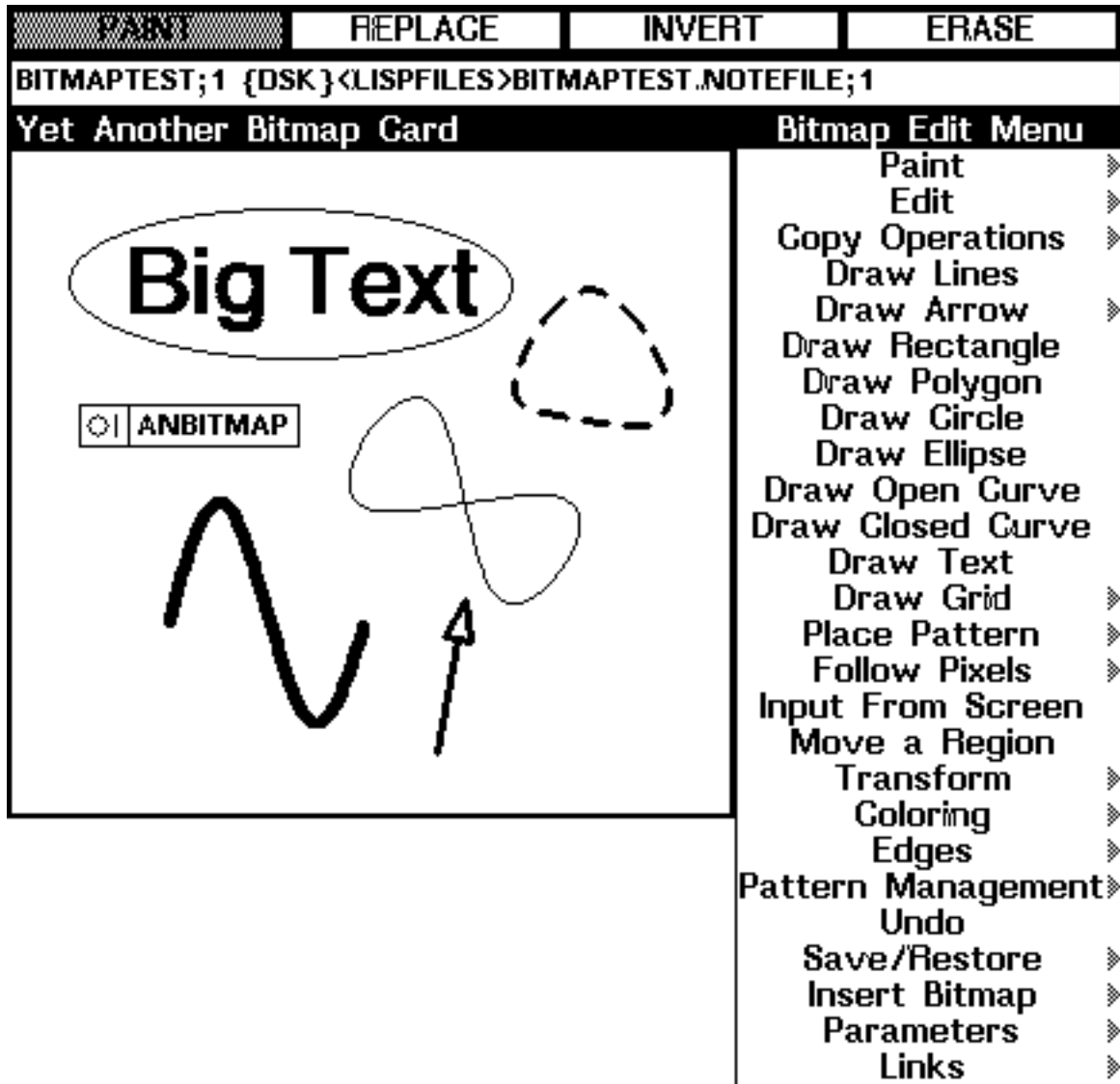
Notecards Bitmap Card

**Vista Laboratory
Xerox Special Information Systems**

November, 1988

This software implements bitmaps as a new substance type within Notecards. Although one can create and edit bitmaps within TEdit or SKETCH, the Bitmap Card gives more capabilities. The Bitmap Editor is described in a separate document (Reference 3). Editing is terminated via the Notecards left button title bar menu items (Close and Save, Delete Card, etc.).

Links are added, deleted, and moved via the Links menu item and its sub-items. Links are not actually part of the bitmap and so are not treated as items to be manipulated by the editor.



To install the Bitmap Card, load NCBITMAPCARD.LCOM which loads BITMAPEDITOR.LCOM and READNUM.LCOM. The Bitmap Editor also requires the Lispuser's package FILLREGION.

References:

1. Notecards Release 1.2k Reference Manual, Xerox Special Information Systems, April, 1986.
2. Notecards 1.3 Release Notes, Intelligent Systems Laboratory, Xerox Palo Alto Research Center, August, 1986.
3. Bitmap Editor, Vista Laboratory, Xerox Special Information Systems, November 1988.

NCCollaboration

By: Randy Trigg (Trigg.pa@Xerox.COM)

Stored: {qv}<notecards>1.3L>Library>NCCollaboration, .lcom, .ted

Written: June 14, 1988 by Peggy Irish

Last updated: June 15, 1988 by Peggy Irish.

See also NCCollaboratorCard and NCHistoryCard packages.

INTRODUCTION

This package was written to assist people using the draft-passing style of collaboration. This style refers to settings in which a small number of collaborators work closely together, but not at the same time. Rather, they take turns using a single shared hypertext workspace. These collaborators need the ability to discuss the use of the medium as well as the work constituting the content of the project. In particular, they need tools to support the processes by which they monitor their own and each other's work.

To this end, several mechanisms have been created to support collaboration. First, the notion of having a notefile understand the identities and certain properties of its individual collaborators has been developed. (See NCCollaboratorCard documentation.) This has also been extended so that multiple simultaneous users may be recognized, in the case where several people are using a notefile while sitting together at a workstation. Second, a method for recording the changes made to a notefile by a collaborator has been implemented. (See NCHistoryCard documentation.) Finally, a "collaboratization" process for configuring a notefile so that it contains these mechanisms has been provided, and is contained in this package. (The NCCollaboratorCard and NCHistoryCard packages will automatically be loaded when NCCollaboration is loaded.)

LOADING NCCOLLABORATION

Loading the NCCollaboration package adds the menu item "Collaboratize" to the session icon notefile menu and to all notefile icons' left button

  "menusGETFN: BMOBJ.GETFN3UP GETFN: BMOBJ.GETFN3"

COLLABORATIZING A NOTEFILE

Selecting "Collaboratize" will cause a number of things to happen to the chosen notefile. A collaborator card will be created for the current collaborator. A new menu item, "Collaborator Info," will be added to the notefile icon's middle button menu. The NoteFile Use History filebox will be created to hold all history cards in this notefile. Also, as part of the opening of a collaboratized notefile, a history card will be created

and opened for the current notefile session. This card will automatically be filed in the NoteFile Use History filebox.

Creating a Collaborator Card

When a collaboratized notefile is opened, a menu of collaborators' names will appear, from which the current collaborator should select his name. During the collaboratization process, the notefile has no prior knowledge about collaborators;

therefore the menu will contain only the "New Name(s)" option: . The user should select "New Name(s)," which will cause a new collaborator card to

be opened. 

This collaborator card stores information about the current collaborator. The title of the card is initialized with the username of the collaborator, which he may change. The collaborator should fill in his or her initials by selecting the "?" with the left mouse button. A prompt window will appear, in which the collaborator can type his or her initials. These initials will appear in the titles of history cards created later. The collaborator may also change the Font property by selecting the existing Font value

and choosing the desired font properties from the resulting menu:  and then selecting DONE. The ExtraTEditProps property may also be changed by selecting the existing value of ExtraTEditProps and typing in the appropriate attribute-

value pairs. 

The Font and ExtraTEditProps properties define the defaults to be used in cards created by this collaborator. Once the collaborator is satisfied with the collaborator

card, he or she should select DONE. 

COLLABORATOR INFO

If the notefile icon middle button menu item "Collaborator Info" is selected, a menu containing the collaborators in this notefile will appear, with the current collaborator's name highlighted. Three commands also appear at the bottom of the

menu: 

The information in any of the collaborator cards may be changed at any time by selecting the name from this menu and editing the collaborator card that is brought up. The "New Name(s)" command is used as described in the **Creating a Collaborator Card** section, with the exception that a new collaborator card does not have a default title unless it is the first one created in the notefile. The "Remove Name" command allows the current user to remove a collaborator name from this notefile. The "Change current collaborator" command allows a different collaborator name to be chosen for the current session in this notefile.

CREATING A HISTORY CARD

Once collaboratized, (and upon collaboratization), a new history card is automatically created for a notefile each time the notefile is opened. Each history card is automatically filed in the NoteFile Use History filebox. Upon creation, each history

card is titled with the current date and the initials of the current collaborator. A time stamp is entered in the card to indicate when the current session

begin

For more detailed information on history cards, please see the history card documentati

NCCollaboratorCard

By: Randy Trigg (Trigg.pa@Xerox.COM)

Stored: {qv}<notecards>1.3L>Library>NCCollaboratorCard, .lcom, .ted

Written: June 14, 1988 by Peggy Irish

Last updated: June 15, 1988 by Peggy Irish.

See also NCCollaboration and NCHistoryCard packages.

INTRODUCTION

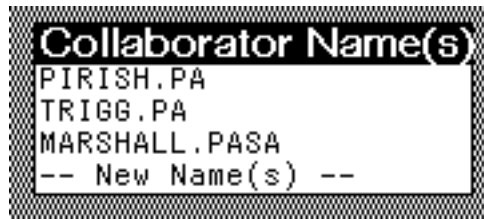
To allow a notefile to distinguish between collaborators, the Collaborator card type was created. A card of this type should be created for each person or set of persons working in a notefile. The collaborator card contains information relevant to collaboration, such as the user's name, initials, and preferred font. The initials are used by History cards, which are described in the History Card documentation. The font is automatically used to display text in cards created by the collaborator. When each collaborator or set of collaborators chooses a different font, it is easy to tell at a glance what work each person has contributed to a notefile's text. A collaborator card may also be created for multiple users, in the case where several people are working on a notefile together and want to claim group ownership of their contributions. When a collaborator (or set of collaborators) opens a "collaboratized" notefile, they are asked to identify themselves. The notefile then looks up their collaborator properties by searching for their collaborator card, and applies these properties to the working session.

LOADING NCCOLLABORATORCARD

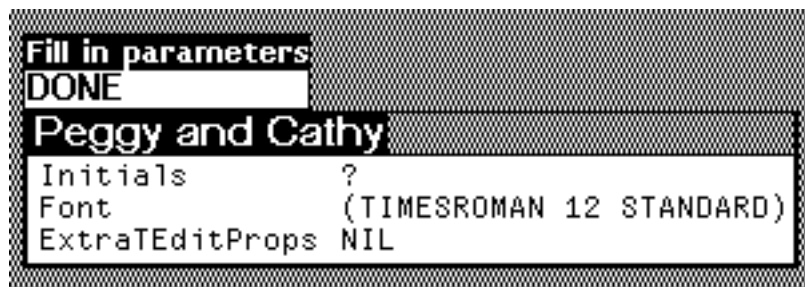
This package should be used in conjunction with the NCCollaboration package. NCCollaboration should be loaded, and it will automatically load the NCCollaboratorCard and NCHistoryCard packages. (See the NCCollaboration documentation for details of its use.)

OPENING A COLLABORATIZED NOTEFILE

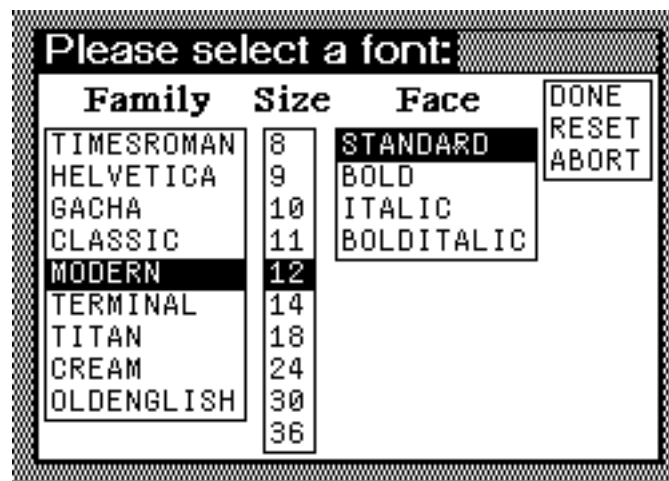
When a collaboratized notefile is opened, a menu of collaborators' names will appear, from which the current collaborator should select either his name or "New Name(s)." (During the collaboratization process, the notefile has no prior knowledge about collaborators; therefore the menu will contain only the "New Name(s)" option):



If one of the names is chosen, that collaborator will be instantiated as the current collaborator, and the corresponding collaborator information will be applied to the session. If "New Name(s)" is selected, the user will be prompted for a name. Then a new collaborator card will be created which has that name as its title:

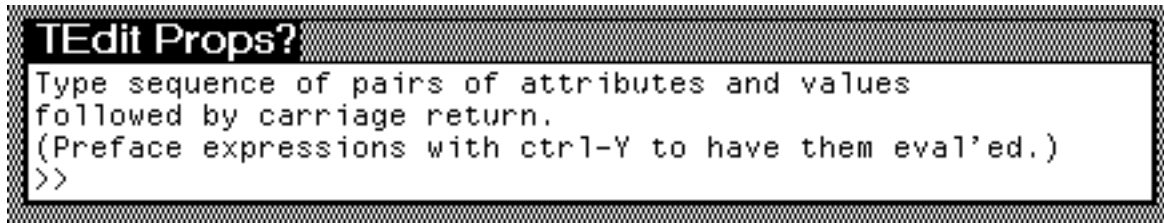


The collaborator should provide a value for the Initials property by selecting the "?" with the left mouse button. A prompt window will appear, in which the collaborator can type the desired initials. These initials will appear in the titles of history cards created later. The collaborator may also change the Font property by selecting the existing Font value and choosing the desired font properties from the resulting menu:

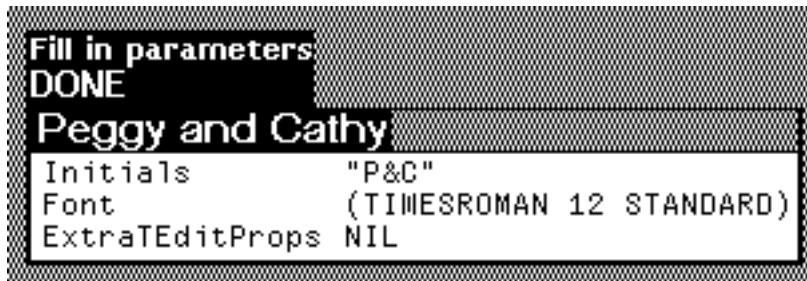


Selecting DONE from the right side of this menu completes the Font property edit operation. The ExtraTEditProps property may also be changed by selecting the

existing value of ExtraTEditProps and typing the appropriate attribute-value pairs into the prompt window:

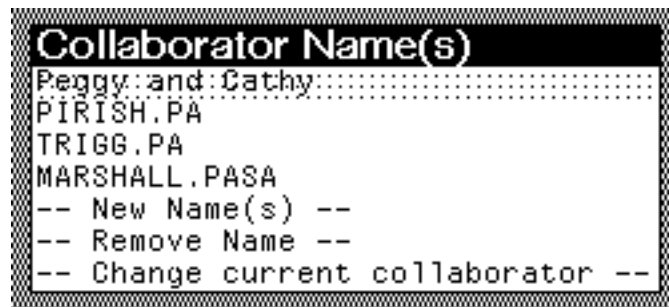


The Font and ExtraTEditProps properties define the defaults to be used in cards created by this collaborator. Once the collaborator is satisfied with the collaborator card, DONE should be selected.



COLLABORATOR INFO

If the notefile icon middle button menu item "Collaborator Info" is selected, a menu containing the collaborators in this notefile will appear, with the current collaborator's name highlighted. Three commands also appear at the bottom of the menu:



The information in any of the collaborator cards may be changed at any time by selecting the name from this menu and editing the collaborator card that is brought up. The "New Name(s)" command is used as described in the **OPENING A COLLABORATIZED NOTEFILE** section. The "Remove Name" command allows the current user to remove a collaborator name from this notefile. The "Change current collaborator" command allows a different collaborator name to be chosen for the

current session in this notefile.

NCDraftCard

By: Ramana Rao (Rao.pa@Xerox.COM)

Stored: {qv}<notecards>1.3k>library>NCDraftCard, .dcom, .ted

Last updated: Mar 30, 1987.

INTRODUCTION

This NoteCards library package defines a new card type called draft cards which supercede document cards. A draft card, primarily, collects the substance from a tree of cards starting from a set of root card, but also does auxiliary processing like sectioning and titling. (The process can be viewed and is implemented as a non-cyclic traversing and interpreting of a notecard network by invoking methods to process card and link types. The draft card defines a set of interpretation methods which implement functionality for document generation).

When a draft card is created, the user is asked to select a set of root card (using the standard NoteCards card selection interface). After this selection, the user is presented with a menu for defining the link and card categorys which are processed by the draft generator. After the draft is generated, a user can use the left button title bar menu to edit the draft style and recompute the draft. A programmer's interface function will be implemented on demand (or rather courteous request) or when I see a break in the clouds. Also true for a long list of desirable features.

CREATING A DRAFT

Draft cards are created by selecting Draft from the NewCards menu (assuming the NCDraftCard library package has been loaded). The user selects a set of root card for generating the document. Then the following draft style editor pops up allowing the user to edit the fields by selecting them. (This is not all there is to a document style as I will say below).

Draft Style Sheet	
ExpandLinks	(SubSection Expander)
BackToCards	NONE
CopyLinks	NONE
TitleLinks	NONE
SectionLinks	(SubSection)
ToSectionsLinks	(SectionPtr)
--DONE--	--CANCEL--

Selecting the items on the bottom line will generate the draft or cancel the creation. Each of the other lines represent a category of links or cards which provides specific functionality during draft generation. The *ExpandLinks* category define the link types which are expanded during draft generation i.e. the substance of the target card (the card pointed to by the link) is pulled into the draft and processed recursively. To prevent runaway loops, a card is only expanded once. The *BackToCards* category define the card types for which back links are put into the draft. Back links allow the user to return to the "underlying" card from the draft. *CopyLinks* are copied directly into the draft. *TitleLinks* produce titles in the draft by inserting the title of the card pointed to by the link. Similarly, *SectionLinks* produce section headings, but in addition they generate section numbers. The section numbers are generated according to "depth in tree using *SectionLinks*" i.e. the section number of a card has as many numbers as *SectionLinks* have been traversed to get to it. *ToSectionLinks* produce references to sections within the draft. Whenever a link of the *ToSectionLinks* category is encountered, the target card is checked for a section number and a reference of the form, "Section xx", is produced. This works for all sections which are part of the draft independent of whether they are before or after the section reference. All categories can be set to ALL, NONE, or a list of link or card types (using the interactive type selector which pops up when the user selects on the field).

The initial settings of the draft style sheet are inherited from a global default style. Any changes made to the style sheet only apply to the associated draft and do not affect the global default settings. To change the global default settings in the current sysout, the daring user can inspect and edit **ncdraft-default-style**. To maintain these defaults across sysouts the user can set **ncdraft-user-props** (a property list using the properties you see when you inspect **ncdraft-default-style**) in her init file. The user can also inspect and edit a draft's style by pulling across on "Edit Draft Style" item in the card menu. Inspecting styles will reveal a host of other parameters which will influence the draft generation process and will also more annoyingly reveal that the

field names which correspond to the link and card types have different names than in the style sheet editor. Use the field name you see in the inspector for adding an entry to **ncdraft-user-assoc** in your init file. (This is all hoaky, but at least functional. You are welcomed to bug me about how you can do something).

Alternatively draft cards can be created programatically as follows:

```
(NCP.MakeDraft <NoteFile> <rootcard> <style-assoc-list> <nodisplayflg> <props>
<parent-fileboxes>)
```

Currently Unimplemented. Bug me when you really need this. Creates and returns a draft card starting from <rootcard>. All parameters not set by <style-assoc-list> will be inherited from the default style. The resulting draft card will have the given props and parents.

DRAFT CARD OPERATIONS

The Draft Card operations are available via the left button (card) menu in the Draft Card card's title bar. **Recompute Draft** recomputes the draft using the associated root card and style. **Edit Draft Style** pops up the style sheet described above and allows you to change the link or card type categories. **Edit with Inspector** is a pull-across item of **Edit Draft Style** which allows you to inspect and edit all of the style parameters. Please READ the last section if you are going to do this.

KNOWN BUGS

Ha! All features, non-features, or future features.

FUTURE FEATURES

User Interface Improvements: Allow the user to edit all the style parameters with a style sheet editor that is like a form (like what FREEMENU would be if it were what it is trying to be). Right now my focus is more on what stylistic things a user might want to dictate, and not on how he will set these parameters.

Automatic generation of Table of Contents. You can do this in a separate draft card with the current functionality by being a little clever at defining which link types to expand, copy, section, or title depending on how you want it to come out.

Sectioning and Title Style parameters. Font choice and other looks options. The hidden style parameters are a small step in this direction, allowing bolding/notbolding of these as well as dictating number of carriage returns before and after a section.

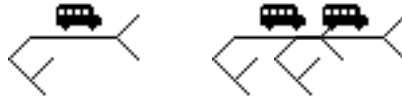
Figure Numbering and Figure references.

Bibliographic References. You can use titling to achieve this with great pain currently, but better ways are coming along. A quick and dirty hack first, and then a full service hack that Miller is working on.

Draft/Network Coupling. Editing the draft would automatically edit the underlying substance. Not an easy feature to provide, but probably a very useful one.

Generate Draft to Tedit Stream.

NCGuidedTourCard



By: Randy Trigg (Trigg.pa@Xerox.COM)

Stored: {qv}<notecards>1.3L>Library>NCGuidedTourCard, .lcom, .ted

Written: August 1, 1988 by Randy Trigg

Last updated: May 5, 1989 by Peggy Irish.

Loads NCTableTopCard package.

INTRODUCTION

The author of a complex NoteCards network is sometimes faced with the problem of conveying its meaning to future online readers using the same hypermedia environment. The NoteCards guided tours library package provides a way to reify and manipulate branching paths through a NoteCards network, thus allowing readers to explore the notefile under the guidance of the notefile's author.

A guided tour is a graph whose nodes are tabletop cards (see the documentation for the NCTableTopCard NoteCards library package) and whose edges are GuidedTour links connecting the cards. Because the tabletop cards may have other links connecting them to other parts of the network, the guided tour is a subgraph of the subnetwork containing those cards. The guided tour card is implemented as a specialization of the NoteCards browser card. Thus, creating and editing a guided tour is almost identical to creating and editing a browser. The biggest change is the ability to "run" the guided tour by means of a row of buttons across the top of the browser window. In this way, readers and authors access the tour through the same graph-based interface.

Readers interested in learning more about the history of and motivation for the guided tour package (as well as the tabletop facility) are directed to the paper "Guided Tours and Tabletops: Tools for Communicating in a Hypertext Environment," by Randall H. Trigg, Conference on Computer-Supported Cooperative Work, September 1988. (Copies can be obtained by writing to SSLPapers.pa@Xerox.com.)

LOADING THE NCGUIDEDTOURCARD PACKAGE

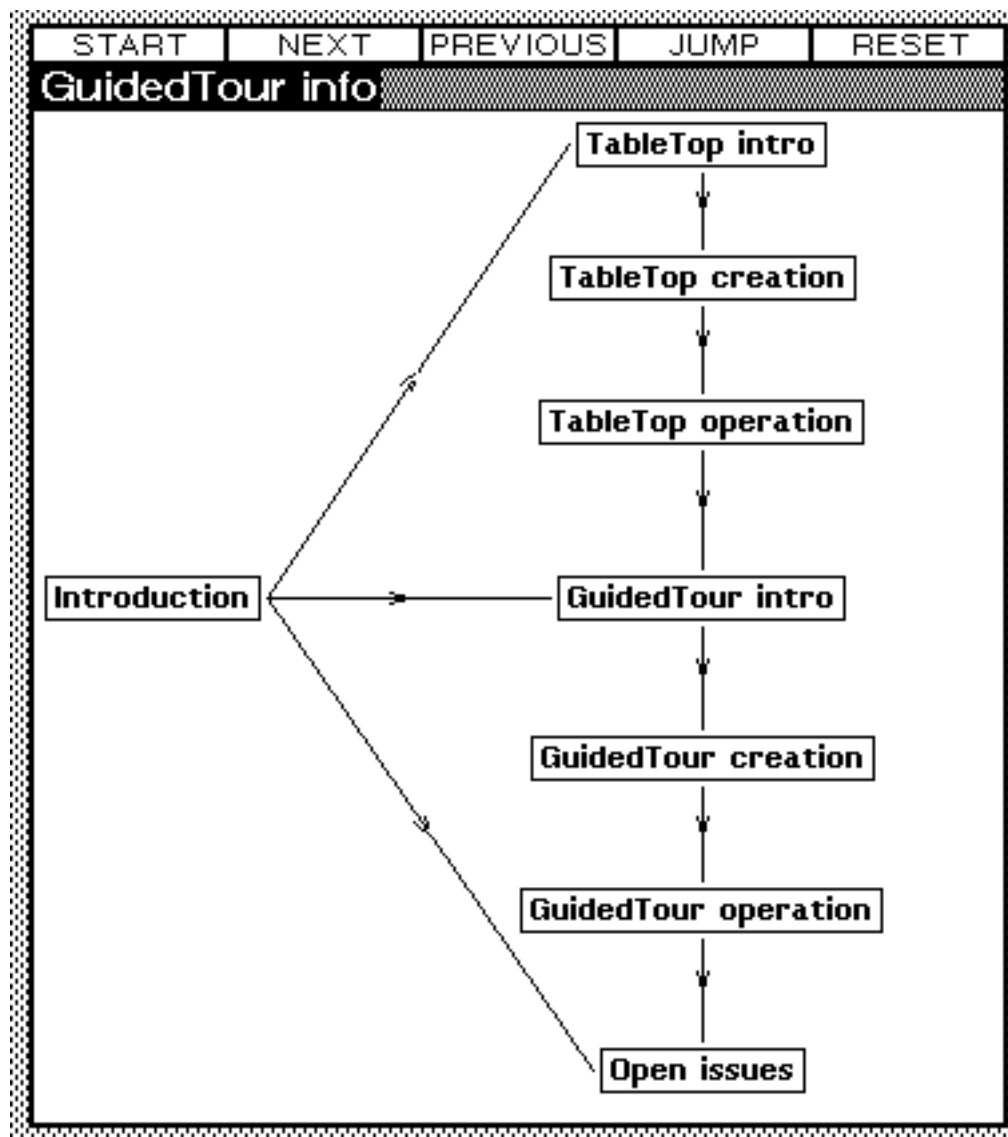
Loading the NCGuidedTourCard package is done in the same way as loading any other NoteCards card type library package. For example, you can type the following at an Interlisp exec:

```
(FILESLOAD (FROM NOTECARDS) NCGUIDEDTOURCARD)
```

Alternatively, you can add the atom NCGUIDEDTOURCARD to the global var NOTECARDSLBRARYFILES in your init file. This will ensure that NCGUIDEDTOURCARD is loaded when bringing up a fresh sysout. Finally, if a link to a guided tour card is followed and the NCGUIDEDTOURCARD package is not currently loaded, then it will be auto-loaded by the system.

"RUNNING" A GUIDED TOUR

Operating or "running" a guided tour is done via the five buttons arrayed along the top of the guided tour card as shown below.



In each of the following operations, the current displayed tabletop is automatically closed down before bringing up the next. (This is made more efficient by not closing those cards shared by the current and next tabletops.)

START - Clicking on **START** causes the first tabletop in the tour to be brought up. For simple linear or singly-rooted tree structures there is an obvious "first" tabletop. Some tours, however, might have multiple roots, i.e. multiple nodes without incoming edges. In such cases, the user is asked to choose from a menu of possible starting tabletops.

NEXT - This is the standard way to move along a path. Clicking **NEXT** simply closes down the current tabletop and brings up the next one on the path. If the current node

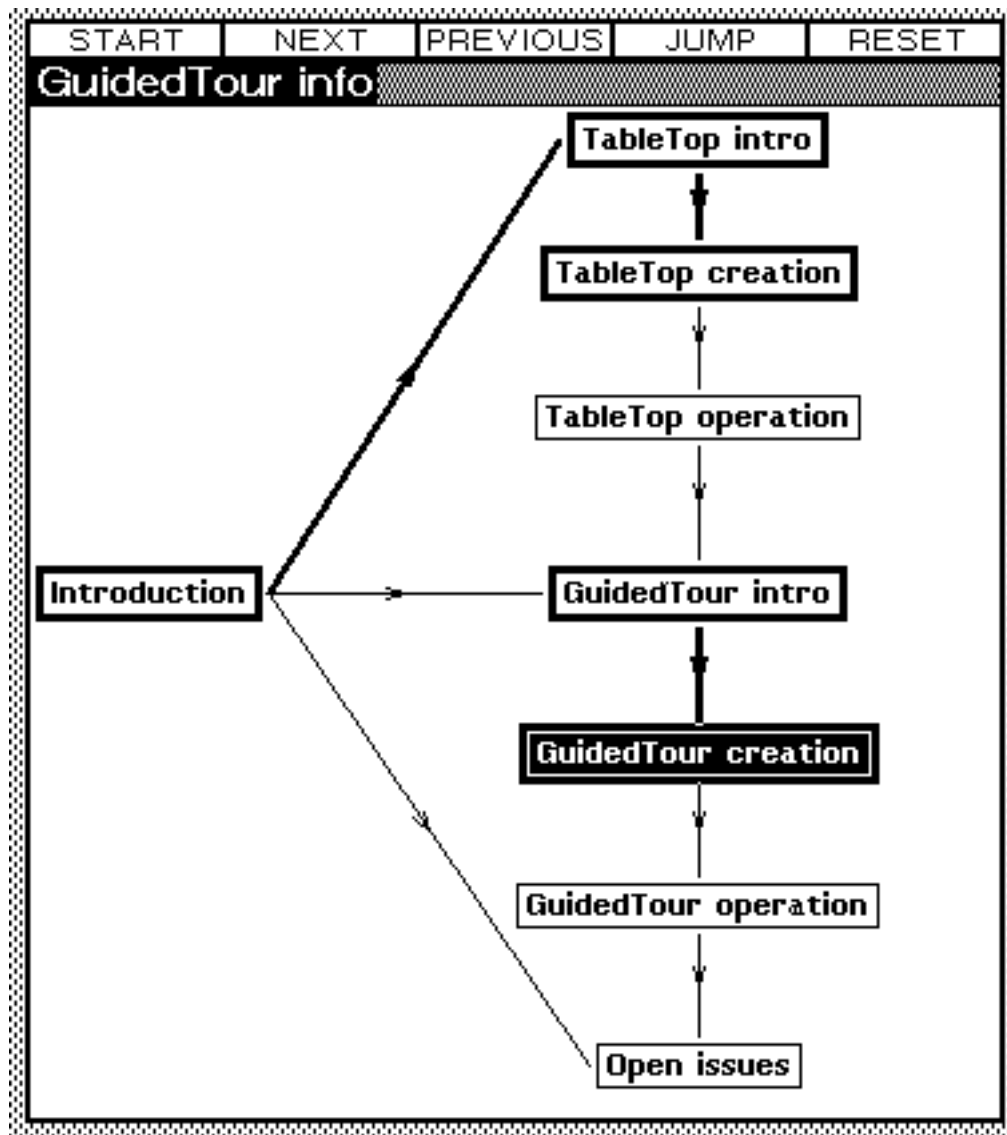
is a branch point having multiple outgoing edges, then the user is asked to choose from a menu of possible next tablesps. In any case, the new current node is highlighted as is the edge that was followed.

PREVIOUS - Clicking here provides the reader with a menu of tablesps already visited. These appear in the order in which they were visited. Clicking on an item in the menu effectively JUMP's to that point in the tour.

JUMP - This operation is used to leave the current path by jumping to an arbitrary tabletop selected by the user from the guided tour graph (again after closing down the current tabletop).

RESET - This operation closes the current tabletop and resets the "state" of the guided tour. In particular, any highlighting of nodes and links in the guided tour graph is turned off and the PREVIOUS list is cleared.

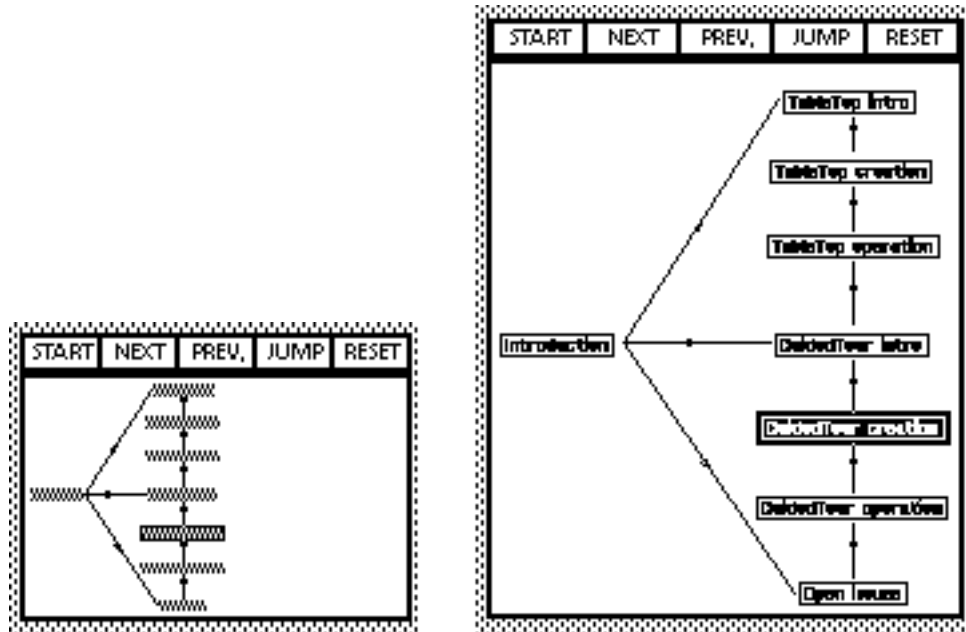
At any point in time, the graph structure displayed in the guided tour card provides various indicators of the reader's place in the tour.



The node in the graph whose tabletop is currently open is highlighted in reverse-video. The nodes for tabletops previously visited are drawn with heavier borders. Finally, the edges corresponding to links followed using the NEXT command are displayed in bold. The guided tour card appearing above shows a point in time when the reader has visited five cards. Starting at "Introduction," the top branch was taken, two tabletops were visited, and then a JUMP was made to "GuidedTour intro" whereupon NEXT was again used to move to the presently displayed tabletop "GuidedTour creation."

This standard mode of operating a guided tour (using the buttons along the top of the card) is extended in several ways. First, because screen space is at a premium and

because the window containing the guided tour graph is often large, the user can run the tour from the guided tour window's shrunk icon, which is a miniture version of the guided tour window. This icon may be shaped to suit the user's needs and to fit the available screen space. The current tour stop is highlighted in reverse-video in the shrunk icon.



Second, it may be desirable to "circumvent" the tour by peeking at the cards in the next tabletop or by bringing up a tabletop without closing down the current one. This can be done directly by middle-buttoning the link icons appearing in the guided tour graph. (See the NCTableTopCard documentation for a description of tabletop operations.) Finally, it is important to emphasize that you can stray from the tour at any time by following links emanating from cards in the tabletop. Such explorations deeper into the notefile don't affect the current state of the tour.

CREATING AND MODIFYING GUIDED TOURS

There are two common styles of constructing tours. In the first, authors create an empty guided tour card and build and link together tabletops using the guided tour's editing menu. In the second, authors build a guided tour over already constructed tabletop cards. In any case, the elements of guided tour construction include creating guided tour cards, creating or including tabletops (and other tour nodes), linking together nodes in the tour, and modifying the tour.

Creating a new guided tour card

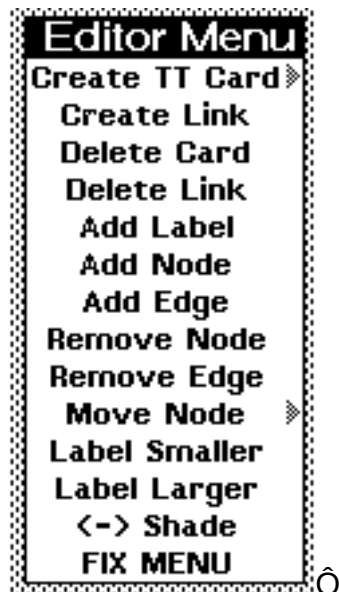
Creating a guided tour card is very much like creating a Browser card. Just choose GuidedTour from the NewCards menu on the notefile icon. If you want the guided tour to be empty, then choose DONE when asked to specify the tour's roots. Otherwise, select a set of starting cards (usually tabletops). Unlike browser cards, guided tour cards don't require you to specify a set of link types to follow. There is a default set of link types (initially the single GuidedTour link type) used by guided tour cards during creation and Recompute. Because of this, the BrowserSpecs stylesheet for guided tour cards doesn't contain the usual columns for link types.

Depth	Format	Orientation
0	*GRAPH*	Horizontal
1	LATTICE	Vertical
2	COMPACT	Reverse/Horizontal
3	FAST	Reverse/Vertical
4		
5		
6		
7		
8		
9		
INF		

DONE
RESET
ABORT

Adding nodes to a guided tour

All editing of the guided tour contents is done through the guided tour editing menu.



Ô

Ä F

Â@ F

Â@ Ô

Ä Ô 0 ÄÄ

Â@ Ô

‘Ü°

Ä Ô Û 0Ž>8| Ä F 0|Ûì3

13

'ÜÄ F Ã 3>| Â@ F Â@ Ô Ä Ô Ä F Â@ F Â@ Ô Ä Ô ì08

Y08P I' Â@ F

Ü = 14yA@ 0 ìû

ÜYİ' Ä F

Ü 7IÜP' Â@ F

Y07IÜP' Â@ Ô

Nodes in the guided tour are usually tabletops, but can be **arbitrary card** types. To add an existing card as a node in the guided tour, use the Add Node menu item just as for standard browsers. To create a new tabletop card from scratch, click on the Create TT Card item. You will be asked for cards to be included in the tabletop just as though you created the tabletop from the NewCards menu on the notefile icon. By *pulling across on the Create TT Card menu item*, you can create guided tour nodes which are cards of other types. If you use such non-tabletop cards in your tour, you'll find that they generally act similar to tabletop cards that contain only one card. (But see the Handy Hooks section below to change this default behavior.)

Nodes in the guided tour can be linked together **using the Create Link** menu item. Normally, you will not be asked to specify a link type when creating a link using the editing menu. (If, however, the default set of link types has been changed to have greater than one element, then **a menu of link types is popped up for you** to select from.) Note also that new links created from the editing menu are always "Global" links.

As with standard browser cards, guided tours can be modified by adding new nodes and edges, removing old ones, rearranging nodes spatially within the tour, and annotating the tour using labels. In addition, it's usually a good idea to specify explicitly what the roots of the tour should be. (Roots can be (re-)specified by selecting the Change Browser Roots item from the middle button title bar menu of the guided tour card.) Otherwise, tour readers clicking on START have to choose from a system-constructed menu of all nodes in the tour that don't have incoming edges.

To create a new guided tour **card programmatically**, call `NCP.CreateCard` just as **you would** to create a browser card:

See the [documentation on NCP.CreateCard](#) for details on the args.

Equivalent to hitting **START** on guided tour card **GTCard**.

Equivalent to **hitting NEXT** on guided **tour card GTCard** except that if `Don'tCloseCurrentFlg` is non-nil, then current **tabletop won't be closed down first**.

Equivalent to hitting PREVIOUS on guided tour card GTCard except that if Don'tCloseCurrentFlg is non-nil, then current tabletop won't be closed down first.

Equivalent to hitting JUMP on guided tour card GTCard *except that if*

Don'tCloseCurrentFlg is non-nil, then current tabletop won't be closed down first. Also if *NextNode* is a valid node in the graph, then user won't get bugged; we'll jump directly to that card.

(NCGT.ResetTour GTCard Don'tCloseCurrentFlg)

Equivalent to *hitting RESET* on guided tour card GTCard except that if

Don'tCloseCurrentFlg is non-nil, then current tabletop won't be closed down first.

Handy Hooks

GuidedTourBringUpFn. If this prop is on a card type's atom, then when cards of that type are encountered in guided tours, they are brought up by calling this function.

(This is useful when you don't want the default behavior for stops in the tour that aren't tabletop cards.) This function is expected to return the list of cards it brings up.

GuidedTourCloseDownFn. If this prop is on a card type's atom, then when cards of that type are encountered in *guided tours*, they are closed down by calling this function. (Note: This prop doesn't seem to be called anywhere, as of the version of October 19, 1988.)

GTNextTourStopFn. This function is called by *NCGT.NextTourStop*, and its job is to "filter" the list of possible next tour stops from the *current tour stop*. The list of next cards (the destination cards of the *NCGT.GuidedTourLinkTypes* links from the current tour stop) is passed to the *GTNextTourStopFn*, which returns some sublist of the next cards. Both the card type atom of the card in the current tour stop and the *card type atom of the guided tour card* are checked for the *GTNextTourStopFn*. If both card type atoms have a *GTNextTourStopFn*, only the one from the current tour stop cardtype is called.

GTCreateGuidedTourLinkFn. If this prop is on the card type atom of the guided tour card, then this function is called whenever a new link is created using the graph edit menu of the guided tour card. This is a way of getting control over *the type of link used*, whether it's global or local, etc.

GTResetGuidedTourFn. If this prop is on the card type atom of the guided tour card, then this function is called whenever the guided tour is reset.

GTBringUpCachingFn. This is an *NCP.CardUserDataProp* of a guided tour card which is called each time a card or tabletop is brought up.

GTCloseDownCachingFn. This is an *NCP.CardUserDataProp* of a guided tour card which is called each time a card or tabletop is closed down. If this prop is non-nil, all cards in the guided tour stop which are being closed are merely undisplayed -- they are still cached. It is the job of the **GTBringUpCachingFn** and the *GTCloseDownCachingFn* to decide which of these cached but undisplayed tour stop cards should be uncached.

GTCleanUpCachingFn. This is an *NCP.CardUserDataProp* of a guided tour card which is **called** when a guided tour card is closed. Its job is to clean up whatever cached but undisplayed guided tour stop cards are still laying around from the *GTBringUpCachingFn* and the *GTCleanUpCachingFn*.

ImmediatelyGoToNextGTStopFn. Whenever a new **guided** tour stop card is brought

up, its card type atom is checked for this function. If the function exists and returns non-NIL, NCGT.NextTourStop is immediately called. Thus, for example, it is possible for a user **to travel** through a whole guided tour, by merely pressing a guided tour's Start button.

Global Variables

NCGT.GuidedTourLinkTypes

This variable is initially bound to the single atom GuidedTour. The link types on this list will be the only legal link types users will be able to create in a guid

The NCHACKS Library Package

Xerox Corporation

Randy Trigg

[First written: 8/23/85 Randy Trigg]
[Last updated: 8/18/87 Peggy Irish]
[Code: {qv}<notecards>1.3k>library>nchacks.dcom]
[Doc: {qv}<notecards>1.3k>library>nchacks.ted]

This document describes a library package containing miscellaneous handy functions written using the programmer's interface for use in NoteCards Release 1.3k. It is hoped that you will find these useful not only for their functionality, but also as examples of the sort of code that can be written using the programmer's interface.

I encourage any readers who have created such functions themselves to send me the code and a short description and I'll incorporate into this package.

These functions all apply to groups of cards (and often to a notefile as well) and allow searching for text, global replacement of text, removal of deleted link icons, sorting of link icons, making link icons invisible, card collection by date, and creating chains of cards using a given link type.

The following functions are applicable to sets of cards and allow global searching, replacing, and chain creation. For those that take a CardsOrNoteFile argument, it can either be single card, a list of cards or a notefile. In the latter case, the entire notefile is used. To allow user selection of cards, pass (NCP.SelectCards) as the CardsOrNoteFile argument. (NCHACKS.MakeChain takes a Cards argument which can either be a single card or a list of cards, but not a notefile.)

(NCHACKS.TextSearch String WildCards? CardsOrNoteFile ReturnLocsFlg)

This goes through the list of cards looking for occurrences of text string String. WildCards? being non-nil means wild card characters may appear in String. The pound sign character '#' matches any single character, asterisk '*' matches any sequence of characters, and single quote, "'" can be used to quote one of the wildcard characters.

If ReturnLocsFlg is nil, then this just returns a list of cards containing at least one occurrence of String. If ReturnLocsFlg is non-nil, then this returns a list of lists. Each sublist has as first element a card ID followed by the locations of occurrences of String. These are single integers if WildCards? is nil and correspond to the location of the first character of String in the text. If WildCards? is non-nil, then these are two element lists of start and end locations of the matching string.

Any cards in Cards that are not TEdit-based cards are ignored (i.e. their card type must inherit from the Text card type).

(NCHACKS.GlobalTextReplace String1 String2 WildCards? CardsOrNoteFile)

This goes through the list of cards replacing every occurrence of text string String1 by String2. WildCards? being non-nil means wild card characters may appear in String1. The pound sign character '#' matches any single character, asterisk '*' matches any sequence of characters, and single quote, "'" can be used to quote one of the wildcard characters.

Any cards in Cards that are not TEdit-based cards are ignored (i.e. their card type must inherit from the Text card type).

(NCHACKS.RemoveDeletedIconsFromTextCards CardsOrNoteFile)

This removes all deleted link icons from the TEdit-based cards in the list of cards.

(NCHACKS.ReorderLinkIconsInTextCards CardsOrNoteFile OrderingFn QuietFlg)

This moves link icons around in each of the TEdit-based cards in the list of cards so that they appear in sorted order. OrderingFn should be a function which accepts two link arguments (the destination cards of the links) and returns non-nil if the first should appear ahead of the second. OrderingFn can also be the atom ALPHABETIZE in which case NC.IDAlphOrder (orders by title of link's destination card) will be the ordering fn used. If OrderingFn is NIL, then the card's OrderingFn card prop is checked. If that card prop is NIL, then NC.IDAlphOrder is again used.

(NCHACKS.MakeLinkIconsInvisible CardsOrLinksOrNoteFile Invisibility)

CardsOrLinksOrNoteFile should be one of: a card, a link, a list of cards and links, or a notefile. Make any link icons for given links or contained in given cards or notefile (but only those in TEdit-based cards) invisible or not depending on whether Invisibility is ON or OFF.

(NCHACKS.DateSearch DateString1 DateString2 CardsOrNoteFile)

This goes through the list of cards looking for occurrences of card parts modified between the dates DateString1 and DateString2. If DateString1 is NIL, then it defaults to a very early date. If DateString2 is NIL, then it defaults to current date. If you provide values for either DateString1 or DateString2, they should be strings in the same format as that returned by the (DATE) format, i.e. "1-Jan-85 00:00:00".

This returns a list of lists, one sublist for each card modified between the given dates. This sublist consists of the card ID followed by two-element lists containing the card part name that was modified and the date of last modification. For example, if there was one hit, you might get something like (({CardObject}#56,164470 (SUBSTANCEDATE "23-Aug-85 19:15:27") (TITLEDATE "23-Aug-85 19:15:44")))

(NCHACKS.CardsModifiedBetweenDates CardsOrNoteFile CardTypes ModifiedRange LastModifiedRange CreatedRange)

This function goes through the list of cards or the notefile looking for cards of types in CardTypes which satisfy all of these criteria: they were modified in the range ModifiedRange, and last modified in the range LastModifiedRange, and created in the range CreatedRange. Each range should be a list of two dates, each in the form DD-Mon-YY (i.e. "1-Jan-85"). Any of the ranges may be NIL, which means not to include that type of modification/creation. If the first (earlier) date in a range is NIL, then it defaults to the earliest known date. If the second (later) date in a range is NIL, then it defaults to the current date. CardTypes should be a list of card types. CardsOrNoteFile should be a list of cards or an open notefile. The cards which meet all of the given criteria are returned in a list.

Examples:

(NCHACKS.CardsModifiedBetweenDates (NCP.WNF) '(Text FileBox) ("3-Jul-87" "10-Jul-87") NIL NIL)

would return all cards of type Text and FileBox in the notefile pointed to by the mouse which were modified between the dates of July 3, 1987 and July 10, 1987.

(NCHACKS.CardsModifiedBetweenDates (NCP.WNF) 'Text NIL ("1-Jun-87" NIL) (NIL "31-May-87"))

would return all cards of type Text in the notefile pointed to by the mouse which were created on or before May 31, 1987, and last modified after June 1, 1987.

(NCHACKS.MakeChain LinkType Cards Position AddCRFlg)

This creates links between successive cards in Cards each of type LinkType positioned at Position. Thus the first card in Cards will be linked to the second, the second to the third, etc. (No link will be built from the last card in Cards.) Position should be in the same form as that accepted by NCP.CardAddText, i.e. either an integer location or one of the litatoms START or END. If the AddCRFlg is non-nil and Position is one of START or END, then a carriage return is inserted after the link icon in the case of START or before the link icon in the case of END.

Any cards in Cards that are not TEdit-based cards are ignored (i.e. their card type must inherit from the Text card type).

NCKey

By: Ramana Rao (Rao.pa@Xerox.COM)

Stored: {qv}<notecards>1.3k>library>NCKey, .dcom, .ted

Last updated: July 13, 1987.

Requires: TEditKey Lispusers Package

INTRODUCTION

This NoteCards library package provides an extensible keyboard interface to useful Notecard Operations (useful means that I needed them at the time). In particular, NCKey provides keystroke commands for creating a link to a new text card using the current selection as either the title or the text of the new card (I was taking a large document and bashing it into a notecards hierarchy, so you can see why these were useful), entitling a card using the current selection, deleting all the "DeletedLinkIcon", and coercing the type and displaymode of all the links in the current card. However, the particular functions provided are not so important as the ability to easily bind your own functions to keys. Possibilities for functions as demonstrated by my bindings (the default bindings) include using available functions in NCHACKS or writing your own (or begging, asking, or demanding some one else to write) functions using the NC Programmer's Interface.

USING THE KEYBOARD INTERFACE

NCKey requires TEditKey but doesn't expect that it be previously loaded. NCKey actions are invoked by typing Meta-W followed by a single character. Currently, the single character can not be Meta-ed (but I realize that this is a pain since you want to be able to keep holding down the Meta key while striking the second character). The mapping from dispatch character to an action routine is made using *NCKEY-BINDINGS*. The default mapping is the following:

Inserts a Link Using Selection as Title	(J j)
Inserts a Link Using Selection as Text	(K k)
Removes Deleted Link Icons	(R r)
Sets Title Using Selection	(T t)
Coerces Links using Global Parameters	(L l)
Brings Up Help Menu	(?)

These items and their operation are described in greater detail below.

CUSTOMIZING OR EXTENDING THE KEYBOARD INTERFACE

The provided mechanism isn't particularly elaborate or elegant, but it's basically good enough for most purposes. You can edit the variable `*NCKEY-BINDINGS*` to remap the bindings to the provided action routines or to bind new action routines. This variable is INITVAR-red by this package, so users can put their own binding in their INIT files. Below is a specification of the syntax of `*NCKEY-BINDINGS*`:

`*NCKEY-BINDINGS*` = (<Binding Triple>*)

<Binding Triple> = (<Key Group> <Action-Fn> <Description String>) | NIL

A NIL <Binding Triple> is used to indicate a space in the Help menu.

<Key Group> = <Key> or (<Key>*)

<Action-Fn> = <function> or (<function> <Action-arg>*).

In the first case, the function is applied to a default set of parameters (<function> CARD NOTEFILE TEXTSTREAM SEL). In the second case, the form is eval-led.

<Action-arg> = The keywords CARD, NOTEFILE, TEXTSTREAM, TEXTOBJ, or SEL or an evalable form using these keywords as arguments. For example, either of the following may be used as an Action-Fn Item:

(RandomFunction CARD NOTEFILE)

(AnotherRandomFunction (HelloFn SEL) TEXTOBJ).

<Description String> is used in the Help menu (Meta-W ?) to indicate what the key is bound to.

(NCKEY-BUILD-HLP)

After editing *NCKEY-BINDINGS*, you must call this function to update the Help menu (Meta-W ?).

PROVIDED ACTION-FNS AND PARAMETERS

NCKEY-DEFAULT-CARDTYPE = Text

NCKEY-DEFAULT-TITLE = "Untitled"

NCKEY-DEFAULT-LINKTYPE = Unspecified

NCKEY-DEFAULT-DISPLAYMODE = NIL

These four user parameters are used by the Provided Action-fns as default values. There is a tradeoff between just using default value and prompting the user for values. I opted for default values because 1) they're a little easier and 2) a major point of this was to get something keyboardishly quick. All of these user parameters are INITVAR-red by this package, so users can put their own bindings in their INIT files.

NCKEY-CREATE-LINK-WITH-TITLE

NCKEY-CREATE-LINK-WITH-TEXT

These create a new card and insert a link to this card into the current card. The current selection is used as either the title or the text of the new card, and is also deleted from the current card.

NCKEY-MAP-LINKS

This function coerces the type and displaymode of all the links in the current card using *NCKEY-DEFAULT-LINKTYPE* and *NCKEY-DEFAULT-DISPLAYMODE* .

(NCHACKS.RemoveDeletedIconsFromTextCards CARD)

This function is obviously a part of NCHACKS and demonstrates an Action-FN which takes arguments other than the default arguments. It removes all the delete icons from the current card.

NCKEY-ENTITLE

This function sets the title of the current card using the current selection. It leaves the current selection alone.

NCKEY-HELP

This function brings up the help menu of the current key bindings. Remember that (NCKEY-BUILD-HELP) should be called to update the help menu after editing the key bindings.

KNOWN PROBLEMS

As mentioned above, the dispatch character can't be a META-character.

NCPPath

By: David Newman (Newman.pasa@Xerox.COM)

NCPPathParse, NCPPathUse

INTRODUCTION

This file is a module designed to allow a programmer to specify a path through a notecards network and a root card, and obtain from these a data structure describing all the paths through the network from the root card that match the path specification. The function `NCPPath.FSM.PathCollect` is the top level function. In conjunction with `NCPPathParse` (which translates a simple path language into the path descriptor that `NCPPath.FSM.PathCollect` requires), it returns a collection of paths for the programmer to manipulate. A typical call to the `NCPPath` functions should look like the following.

```
(NCPPath.FSM.PathCollect (NCPPathParse <Path-description> ) <RootCard>)
```

This package and its functions are implemented on a bastardized finite state machine model. The key difference is that in order to allow the imposition of limits on loops in the finite state network (limits on the number of times a loop can be traversed), the number of times each loop is traversed must be recorded in the finite state machine. In addition, this system will not collect any paths which attempt to traverse the same link of a notefile in the same direction a second time.

The data structures of this system include the InterLisp datatypes `NCPPathFSM`, `NCPPathFSMNode`, and `NCPPathPathStep`, and two list structures. A collection is a list of paths. A path is a list of pathsteps with the root card at the end. Paths contain the steps taken in reverse order, and the various paths in a collection share cons cells in order to conserve memory. Please note also the descriptions of the three records listed below.

RECORDS

`NCPPathFSM`

[Datatype]

The fields of an `NCPPathFSM` are `InitialState`, `CurrentState`, `Path`, and `LoopLimitAList`. Each is a Lisp POINTER. `InitialState` and `CurrentState` contain objects of type `NCPPathFSMNode`, and are the current and initial states of the finite state machine respectively. The `Path` field contains the intermediate

results of a traversal through the NoteCards network. Finally, LoopLimitAList is an a-list that keeps track of the number of times any loop in the finite state transition network has been traversed.

NCPPathFSMNode [Datatype]

The fields of an NCPPathFSMNode are Predicate, Card/Link, Direction, NextNodes, and LoopLimit. The first and fourth are POINTERS, the second and third are FLAGS, and the last is an INTEGER. The Predicate field points to a function or LAMBDA expression, which when applied to a notecard or link returns T or NIL; this indicates whether the item in question will satisfy the next step in the path description. The Card/Link flag indicates whether the Predicate is to be applied to links or cards, (T indicates links and is the default, while NIL indicates cards), and the Direction flag indicates whether a link is to be followed in a forward direction or a backward direction (T indicates forward, and is the default setting). NextNodes is a list of the next nodes in the finite state transition network, and hence is essentially a list of arcs away from the current node. Lastly, LoopLimit indicates the number of times that a loop beginning with the current node may be traversed; its default value is one.

NCPPathPathStep [Datatype]

An NCPPathPathStep has only two fields, both of which are POINTERS. the Link field contains a link in the notecards network, and the Direction field indicates the direction that this link was followed.

FUNCTIONS

NCPPath.FSM.PathCollect *PathSpec RootCard*) [Function]

This function collects a list of complete paths starting at RootCard as specified by PathSpec. The paths are really a network as they share CONS cells, and are actually reversed. The end of each is a pointer to the ID for RootCard. The first item in each path is actually the NCPPathFSM representing the remaining parts of the path to be collected. When the paths are complete, this is a NIL.

(NCPPath.FSM.RealPathCollect *FSMInstance RootCard*) [Function]

This function does the real work of NCPPath.FSM.PathCollect after that function has done the error checking. (FinishedPaths has an extra NIL at the front, so the CDR at the end removes it)

(NCPPath.FSM.FirstStep *RootCard FSMInstance*) [Function]

This function takes care of the case where the first NCPPathFSMNode has a list as its NextState.

(NCPPath.FSM.RealFirstStep *RootCard FSMInstance*) [Function]

This function is specially intended to get the first set of links from a path specification (the NCPPathFSM) and its root card. It constructs the first level or two of the tree of working paths. The function handles the special case where the first two FSMNodes in NCPPathFSM include card predicates.

(NCPPath.FSM.ListFirstSteps *RootCard FSMInstance*) [Function]

This function gets the first set of links from a path specification and a root card. End is bound specially so that all the paths created will share their last cons cells.

(NCPPath.FSM.AddPotentialSteps *FSMInstance*) [Function]
 Add all potential steps to the Path in *FSMInstance* without checking them against any specification.
 Returns a list of NCPathFSMs

(NCPPath.FSM.ListMultiplePaths *FSMInstance*) [Function]
 This function is intended to help out with the problem of true FSMs. It takes a NCPathFSM that has a list of FSMNodes as its CurrentState, and returns a list of NCPathFSMs each of which has one of the FSMNodes as its CurrentState.

(NCPPath.FSM.ComputeMultipleCollections *FSMInstance*) [Function]
 This function handles the case where the NextNodes field of the NCPathFSMNode in the CurrentState field of *FSMInstance* is a list rather than an individual FSMNode.

(NCPPath.FSM.AddNextSteps *FSMInstance*) [Function]
 This function adds the next set of steps to a particular path. That is, it takes a path, gets the NCPathFSM representing the PathSpec and the last link of the path, and adds each of the appropriate next steps to that path - returning a list of several paths with common CONS cells. It also puts the next state of the NCPathFSM on the front of the Path for the next time around.

(NCPPath.FSM.IncrementUseCount *FSMInstance*) [Function]
 Increment the count kept in the AList on the FSM indicating how many times the CurrentState has been used in this path.

(NCPPath.FSM.AddStep Step *FSMInstance*) [Function]
 Given an old FSMInstance and a new step, this function adds the step to the path in the FSMInstance. This function also increments the CurrentState to the NextState

(NCPPath.FSM.NextState *FSMInstance*) [Function]
 This function returns the next state in *FSMInstance* which defines a path specification. If the CurrentState of *FSMInstance* is NIL, we report the error and return NIL.

(NCPPath.FSM.LoopLimitExceededP *FSMInstance*) [Function]
 This predicate determines whether or not the FSMNodeInstance stored in the CurrentState of *FSMInstance* has been used too many times to get to the current place in the path. This function determines if a limited loop has been followed too many times.

(NCPPath.FSMState.ComputeCollection *FSMInstance*) [Function]
 This function takes an NCPathFSM as its single argument. The CurrentState of the NCPathFSM specifies a card rather than a link. If the last step of the path meets specification, and the next node in the NCPathFSM specifies a link, the path is returned. If the last step of the path meets the specification, and the next node in the NCPathFSM specifies a card, the intervening links are added, and the list of new paths is returned.

(NCPPath.FSMState.ListNextSteps *PathStepORCard FSMNodeInstance*) [Function]

This function finds the next steps that can be taken from a particular point in a path. It accepts either a card or a link as the indicator of the current path position and it returns a list of links.

(NCPATH.FSMState.SpecifiesCardP *FSMState*) [Function]

This function returns T iff the FSM-State is not NIL and it's CARD/LINK flag indicates that the

predicate is to be applied to a CARD.

(NCPATH.FSMState.SpecifiesLinkP *FSMNodeInstance*) [Function]

This function returns T iff the FSM-State is not NIL and the LINK/CARD flag indicates that the predicate is to be applied to a LINK.

(NCPATH.FSMState.TerminalP *FSMNode*) [Function]

This function determines whether or not a NCPATHFSM node is a terminal node. That is, whether or not there is a transition to another node from this one.

(NCPATH.Collection.ComputeNextCollection *PathCollection*) [Function]

This function computes a new list of FSMs from an old one. The new FSMs are created by taking an old FSM and adding one step to its path. This function also distinguishes the case where the next step is specified as a card predicate rather than a link predicate.

(NCPATH.Collection.CollectMultiplePaths *Collection*) [Function]

This function takes a collection of NCPATHFSMs and expands those that have multiple FSMNodes as their CurrentState into a list of NCPATHFSMs each having one FSMNode as its current state.

(NCPATH.Collection.ListRemovablePaths *Collection*) [Function]

List those paths of Collection which are complete, or loopy, or which have traversed the same loop of the FSM too many times.

(NCPATH.Collection.ListFinishedPaths *Collection*) [Function]

List those paths in Collection that are complete as specified. These are the paths that the user wants to see.

(NCPATH.Path.Create Root FirstStepLink *FSMInstance*) [Function]

This function creates a new path. Since a path is a list of NCPATHPathStep in reverse order with a root card ID at the end, we create the first step and the root and cons them together.

(NCPATH.Path.End *Path*) [Function]

This function returns the end card in a path by taking the last step of the path and using the Direction field to determine whether to return the source or destination of the link in the Link field.

(NCPATH.Path.AddStep Step *Path*) [Function]

Given Path and a new step, this function adds the step to the Path

(NCPATH.Path.LastStep <i>Path</i>)	[Function]
Since a Path is a reversed list, we just take the CAR to get the last step.	
(NCPATH.Path.StepInPathP <i>TestStep Path</i>)	[Function]
This predicate determines <i>if TestStep is in Path</i> or not.	
(NCPATH.Path.EQUAL <i>Path1 Path2</i>)	[Function]
This function checks for equality of two paths that are still reversed, and have the root card at the end. It is significantly faster than EQUALALL, but uses a number of CONS cells.	
(NCPATH.Path.LoopsP <i>Path</i>)	[Function]
This predicate returns T iff the last step in Path makes Path circular.	
(NCPATH.PathStep.End <i>PathStep</i>)	[Function]
This function returns the card at the appropriate end of PathStep, as indicated by the Direction field of the PathStep.	
(NCPATH.PathStep.PotentialSteps <i>PathStep Direction</i>)	[Function]
This function computes the possible next links from the link in PathStep.	
(NCPATH.PathStep.MeetsFSMCardSpecificationP <i>PathStep FSMNode</i>)	[Function]
This function determines whether or not the card specified <i>by PathStep</i> meets the specification of <i>FSMNode</i> .	
(NCPATH.Apply <i>FSMNodeInstance Item</i>)	[Function]
This function is anticipated to make the general path language easier to implement. It will look at the NCPATHFSMNode and it will use APPLY appropriately, else it will perform the right parsing and whatnot and then use apply.	
(Copy.NCPATHFSM <i>FSMInstance</i>)	[Function]
This function copies an NCPATHFSM much faster than COPYALL does. It should save time in NCPATH.FSM.IncrementCurrentState.	
(Copy.NCPATHFSMNode <i>FSMNodeInstance</i>)	[Function]
This function duplicates an NCPATHFSMNode.	
(NCPATH.Link.ListPotentialSteps <i>PreviousLink PreviousDirection Direction</i>)	[Function]
This function takes a link, the direction it was followed, and another direction flag, and returns a list of all potential links that may be the next step in the path from the appropriate end of the Link in the. The second Direction flag determines which way the found links are followed.	
(NCPATH.NoteCard.ListPotentialSteps <i>Card Direction</i>)	[Function]
This function returns the potential links that might be of use from Card.	
(NCPATH.Link.GetCard <i>PreviousLink Direction</i>)	[Function]
Given a link and a flag, this function decides which end of the link (source card or destination card) to return	

NC Path Language Description

<path>	::= <path step> (<path step> ...) (<path>) (<path> <repeater exp>)
<path step>	::= <path step descriptor> (<path step op> <path step> ...)
<path step descriptor>	::= <literal descriptor> <functional descriptor> ANY <lisp-variable>
<literal descriptor>	::= <link-type> @<card-type> _<link-type> _@<card-type>
<functional descriptor>	::= #<lisp-link-predicate> @#<lisp-card-predicate> _#<lisp-link-predicate> _@#<lisp-card-predicate>
<path step op>	::= OR AND NOT
<repeater exp>	::= <repeater token> <repeater token> <depth cutoff>
<repeater token>	::= + *
<depth cutoff>	::= <positive integer>
<positive integer>	::= <digit> <digit> <positive integer>
<digit>	::= 1 2 3 4 5 6 7 8 9 0

Notes:

- 1) The characters #, @, and _ are called the prefix of a path step descriptor, and their order is unimportant.
- 2) Note that <lisp-variable> *must have* a value parseable as a <path step descriptor>.

NCPPathParse

By: David Newman (Newman.pasa@Xerox.COM)

NCPPath, NCPPathUse

INTRODUCTION

This package parses expressions of a simple path description language into NCPPathFSM data structures for use by NCPPath.FSM.PathCollect. The language is described by a BNF grammar in the file NCPPathLanguage.TEdit. A simpler description of the language follows.

PATH DESCRIPTION LANGUAGE

The basic units of the language are path step descriptors. Each descriptor is either a literal descriptor or a functional descriptor. The literal descriptors consist of link types and card types currently existing in the active notefile. Functional descriptors are the names of lisp predicates that can be applied to a link or card, and which return T or NIL. A proposed new step in a path through a notefile is accepted when it is of the type specified by a literal descriptor or when application of the function specified by a functional descriptor returns T. Each path step descriptor has an optional prefix of several characters. The character @ indicates that the descriptor specifies the card at the end of the path step. The character # prefaces all functional descriptors. The character _ indicates that links followed in obtaining this step should be followed backwards. Path descriptors can be combined into other descriptors by the use of AND, OR, and NOT subject to the restriction that all descriptors in a combination must either apply to cards or links. Lastly, repetitive path structures may be specified using a number or Kleene star notation. See the examples below.

EXAMPLES

FOO is a function name, GOODP is a function name, and BAR is a variable with value @#GOODP.

(NCPPathParse (QUOTE #FOO))	(NCPPathParse (QUOTE @#FOO))
(NCPPathParse (QUOTE _@#FOO))	(NCPPathParse (QUOTE BAR))

```

(NCPPathParse (QUOTE @Text))          (NCPPathParse (QUOTE SubBox))
(NCPPathParse (QUOTE (SubBox)))        (NCPPathParse (QUOTE (@#FOO)))
(NCPPathParse (QUOTE (SubBox FiledCard)))
(NCPPathParse (QUOTE (@#FOO @#FOO)))  (NCPPathParse (QUOTE (AND @#FOO BAR)))
(NCPPathParse (QUOTE (OR @#FOO (AND @Text BAR))))
(NCPPathParse (QUOTE (#FOO *3)))      (NCPPathParse (QUOTE ((#FOO #FOO) *3)))

```

FUNCTIONS

(NCPPathParse *Expression*) [Function]
 This function is intended to be the top-level function that parses an expression into an FSM that NCPPath.FSM.PathCollect can deal with.

(NCPPathParse.FSMFromPathSpec *Expression*) [Function]
 This function is intended to be the top-level function that parses an expression into an FSM that NCPPath.FSM.PathCollect can deal with.

(NCPPathParse.Path *Expression*) [Function]
 This function parses an individual path as a list of steps or repeater expressions.

(NCPPathParse.PathStep *Expression*) [Function]
 This function parses an individual step of a path. A step is either a path step descriptor or some combination of descriptors.

(NCPPathParse.PathStepDescriptor *Expression*) [Function]
 Parses a path step descriptor. A descriptor is a literal descriptor, a functional descriptor, a "don't care" expression, or a variable pointing to a descriptor of one of the other types. The variable is checked for immediate circularity, but not for indirect circularity; either of these conditions could cause an infinite loop.

(NCPPathParse.LiteralPathDescription *Expression*) [Function]
 This function parses path descriptors that literally describe what kind of object qualifies as a step in the path sought through the notecards network.

(NCPPathParse.FunctionalPathDescription *Expression*) [Function]
 This function parses path descriptors that give function names which act as predicates to determine whether a link or card matches this step in the path descriptor.

(NCPPathParse.CreateFSMNode *Item Link/CardFlag DirectionFlag FunctionFlag*) [Function]
 This function creates the NCPPathFSMNodes that will become part of the NCPPathFSM being created by NCPPathParse. It contains a call to the code which creates the LAMBDA expressions to be used as predicates if necessary.

(NCPPathParse.CreatePredicateForm *Type Link/CardFLAG*) [Function]

This function creates a LAMBDA expression that will serve as a predicate. See NCPPathParse.CombinePredicates.

(NCPPathParse.RepeaterExpression *RepeaterExpression LoopSteps*) [Function]

This function parses repeater expressions.

(NCPPathParse.RepeaterToken *RepeaterExpression LoopSteps*) [Function]

Creates a repeater loop with no limit, or with infinite limit

(NCPPathParse.LimitedRepeaterToken *RepeaterExpression LoopSteps*) [Function]

This function parses repeater tokens that have an integral limit.

(NCPPathParse.LoopDecision *Expression LoopSteps MinimumRepeat Symbol*) [Function]

This function decides what kind of loop is to be created (requiring one or zero passes through the loop) and then creates a description of that loop.

(NCPPathParse.CreateLoop *LoopSteps MinTimes MaxTimes*) [Function]

Here we create an expression describing a loop that NCPPathParse.CoalesceStates can transform into an actual loop.

(NCPPathParse.PathStepOperation *Expression*) [Function]

Here we parse combinations of pathsteps. Combinations are created using AND, OR, and NOT, to combine other pathsteps.

(NCPPathParse.FunctionP *Function*) [Function]

This function returns T if its argument names an appropriate function for use as a predicate in an NCPPathFSMNode.

(NCPPathParse.CheckAndComputeFlag *StepList FlagName*) [Function]

This function computes the Direction and Card/Link flags when parsing a combination FSMNode. It also determines if the flags involved match, and notifies the user if they do not. It returns NIL if the flags do not match.

(NCPPathParse.CombinePredicates *StepList Operation*) [Function]

This function builds the LAMBDA expression that will be the predicate in a combination FSMNode.

(NCPPathParse.CombinationExpressions *StepList Operation*) [Function]

This function combines pathstep specifications using AND, OR, or NOT into a single NCPPathFSMNode.

(NCPPathParse.CoalesceStates *NodeList*) [Function]

This function takes a lisp-style list of NCPPathFSMNodes and turns them into a linked list. The linked list will include circularities where repeater expressions exist.

(NCPathParse.CoalesceRepeaterStates *LoopList* Next Limit) [Function]

This function creates the circular linked list structure that repeater expressions need.

(LOGEQUAL . *U*) [Function]

This function is a logical operator. It determines if the arbitrary number of arguments are logically equal or not.

(NAND . *Args*) [Function]

This function is a logical operator similar to AND or OR, except that it performs the logical operation of (NOT (AND *Args*)).

NCPATHUse

By: David Newman (Newman.pasa@Xerox.COM)

NCPATH, NCPATHParse

INTRODUCTION

This package is intended to provide some functions useful to the programmer using the NCPATH package. These functions allow the programmer to transform the data returned by NCPATH.FSM.PathCollect into something more useful to a particular application. Also, three functions intended for use as Print definitions for the three major datatypes in NCPATH are included in this file.

FUNCTIONS

(NCPATH.GetCardPathListsFromPathCollection *PathCollection*) [Function]

This function collects a list of cards traversed by each path in PathCollection. Thus, it returns a list of lists. Each sublist is a list of all the cards traversed in a particular path.

(NCPATH.GetTerminalsFromPathCollection *PathCollection*) [Function]

This function returns a list of the terminal cards in a collection of paths. It is intended to assist the user in getting the results desired from what NCPATH.FSM.PathCollect returns.

(NCPATH.GetUniqueTerminalsFromPathCollection *PathCollection*) [Function]

This function is very similar to NCPATH.GetTerminalsFromPathCollection, but it collects only unique terminal cards, where the other function can collect multiple copies of the same terminal card.

(NCPATH.ReverseAndSeparatePaths *PathCollection*) [Function]

The results of NCPATH.FSM.PathCollect is a list of lists, each sublist being a path in the collection. These paths share cons cells for efficiency, and are stored with the steps in the paths in reverse order. This function accepts the result of NCPATH.FSM.PathCollect, and returns a list of paths with their steps in the correct order, and broken apart such that no cons cells are shared between paths.

(PathStepPrint *Instance Stream*) [Function]

This function is intended to be a print definition for the NCPATHPathStep datatype.

(PrintFSM *Instance Stream*) [Function]

This function is intended for use as a print definition for the NCPATHFSM datatype.

(PrintFSMNode *Instance Stream*)

[Function]

This function is to be a print definition function for FSMNodes.

NCScreen

By: Randy Trigg (Trigg.pa@Xerox.COM)

Stored: {qv}<notecards>1.3k>library>NCScreen, .dcom, .ted

Last updated: May 4, 1987.

INTRODUCTION

This NoteCards library package provides a set of user-callable functions that can be used to manipulate cards on the screen. Other NoteCards library packages including NCCluster use NCScreen functions.

FUNCTIONS FOR LAYING OUT CARDS

(SCREEN.LayoutCardsInSquare <StartPos> <Cards>)

Lay out the cards appearing in the list <Cards> in a square arrangement with the upper left corner of the first card at the position <StartPos>. The number of columns is equal to the floor of the square root of the number of cards. Card windows are arranged in row order attempting to fit all windows on the screen without overlapping. However, some windows may overlap in order to fit each window on the screen.

(SCREEN.LayoutCardsInCascade <StartPos> <Cards>)

Lay out the cards in <Cards> in a cascaded overlapping deck with the upper left corner of the first, deepest card at the position <StartPos>. Calls SCREEN.GetCascadePosition.

(SCREEN.GetCascadePosition <Window>)

This function is used to determine the position for the next card in a cascade. It takes one argument which is a window and returns the position of the upper left corner for the next cascaded window. The spacing between cards in the cascade is determined by the global variables SCREEN.CascadeXSpace and SCREEN.CascadeYSpace (initially set to 3 and 8, respectively).

(SCREEN.LayoutCardsInSurround <CenterCard> <Cards>)

Arranges the cards in the list <Cards> around three sides of the card <CenterCard>. Cards are placed in order down the right side, the left side, and along the bottom. Note: this may not display all cards in <Cards>. The number of cards displayed is returned.

(SCREEN.SurroundCardWithFileBoxContents <Card> <Box>)

Arrange the child cards of the filebox <Box> around the card <Card>. If either arg is NIL, then the user is asked to select. This calls SCREEN.LayoutCardsInSurround. If some cards in <Box> could not be displayed, then a statement to that effect is printed in the prompt window.

MISCELLANEOUS HANDY FUNCTIONS

(SCREEN.MoveCardToPos <Card> <Pos> <Corner>)

Move the <Card> card's window such that its <Corner> corner is at the position <Pos>. <Corner> should be one of the litatoms UL, LL, UR, or LR. The default is UL. If the card is not currently on the screen, then it will be brought up. This calls SCREEN.MoveWinToPos.

(SCREEN.MoveWinToPos <Win> <Pos> <Corner>)

Move the <Win> window so that its <Corner> is at the position <Pos>. <Corner> should be one of UL, LL, UR, or LR as above. If the window doesn't completely fit on the screen at that position, then it is moved so it does.

(SCREEN.WinLLCorner <Win>)

(SCREEN.WinLRCorner <Win>)

(SCREEN.WinULCorner <Win>)

(SCREEN.WinURCorner <Win>)

These functions return the positions of the various corners of the window <Win>.

(SCREEN.WinShrinkAndPlace <Win>)

Shrink the window <Win> (if not already shrunk) and place so its upper left corner is at the upper left corner of <Win>.

NCTableTopCard



By: Randy Trigg (Trigg.pa@Xerox.COM)¹

Stored: {qv}<notecards>1.3L>Library>NCTableTopCard, .lcom, .ted

Written: July 30, 1988 by Randy Trigg

Last updated: May 5, 1989 by Peggy Irish.

See also NCGuidedTourCard package.

¹ The first version of TableTops was written by Jeff Shrager and subsequently modified by George Cole.

INTRODUCTION

Tabletop cards are a means in NoteCards of capturing the layout of some set of cards on the screen. A tabletop is a snapshot which includes the list of cards, the shapes of their windows, their positions on the screen, the scrolled locations (vertically and possibly horizontally) of the windows' contents, and the order in which to open the windows so that the original (possibly) overlapping arrangement can be preserved. Tabletops have three main uses: (1) as a means of storing card layout across notefile closings/openings. In this case, the system constructs and maintains the tabletop automatically. (2) as aids for online demonstrations of notefiles and to help in creating hard copy screen snapshots for offline presentations. Here, the user creates the tabletop card manually. (3) as "stops" along a NoteCards guided tour. In what follows, I describe the system support for the first two cases. For a description of the use of tabletop cards in guided tours, see the documentation on the NCGuidedTourCard package.

LOADING THE NCTABLETOPCARD PACKAGE

Loading the NCTableTopCard package is done in the same way as loading any other NoteCards card type library package. For example, you can type the following at an Interlisp exec:

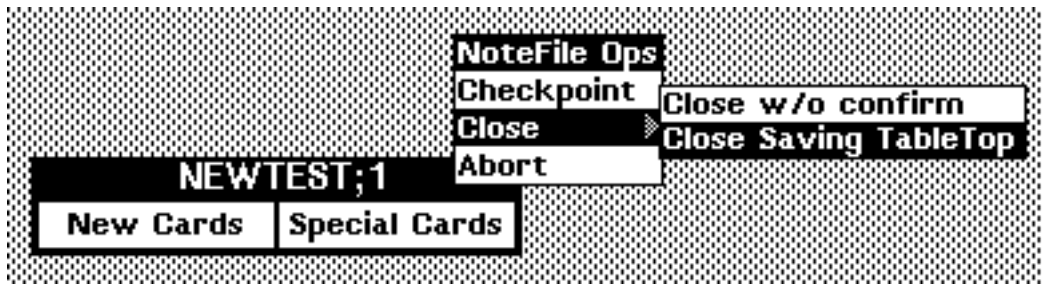
```
(FILESLOAD (FROM NOTECARDS) NCTABLETOPCARD)
```

Alternatively, you can add the atom NCTABLETOPCARD to the global var NOTECARDSLIBRARYFILES in your init file. This will ensure that NCTABLETOPCARD is loaded when bringing up a fresh sysout. Finally, if a link to a

tabletop card is followed and the NCTABLETOPCARD package is not currently loaded, then it will be auto-loaded by the system.

SAVING A TABLETOP AT NOTEFILE CLOSING

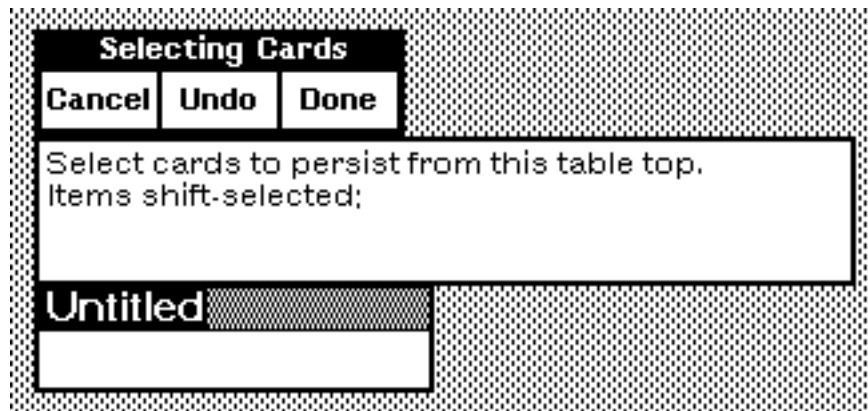
Using a tabletop, one can save the screen layout of cards from a notefile in such a way that the layout can be reinstated the next time the notefile is opened. This allows for greater continuity of work across notefile closings and reopenings. When this package is loaded, the "NoteFile ops" left-button menu available from the notefile icon is augmented slightly. The new pull-across menu for "Close" appears as follows



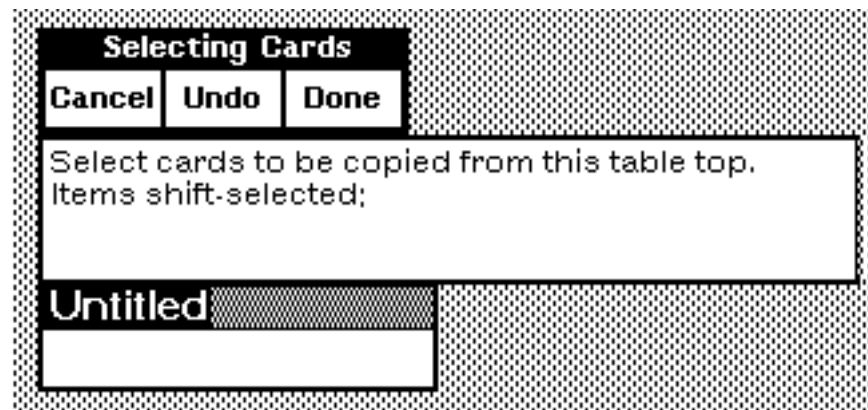
Choosing the option **Close Saving TableTop** causes the current layout of cards with open (or shrunk) windows to be saved in a special tabletop and then the notefile to be closed. The next time the notefile is opened, those same cards will be brought up and arranged (or shrunk) according to the saved layout. (The NCTABLETOPCARD package will be auto-loaded if necessary.)

BUILDING AND MANIPULATING TABLETOPS BY HAND

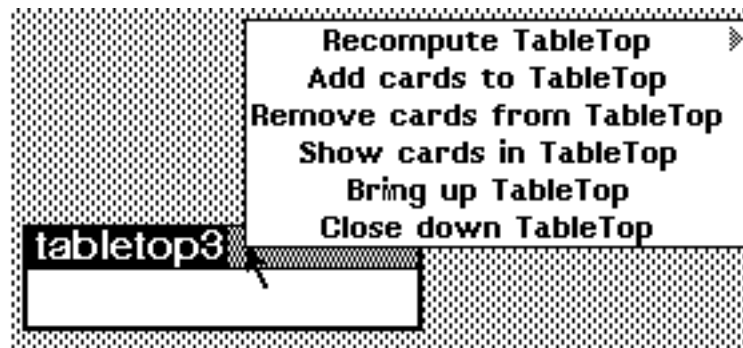
You can create a tabletop card in the usual way, namely by middle-buttoning the NewCards menu on the notefile icon and choosing **TableTop**. You will be asked if you want to use a template tabletop for the creation of this new tabletop. Frequently a user will want several tabletops to share one or more windows, so this option allows one tabletop to be modeled easlily after another. Selecting Yes allows you to pick a tabletop to be used as a template for the new tabletop. Then you will be asked to select the cards to persist from the template, ie. which cards should actually exist in both tabletops.



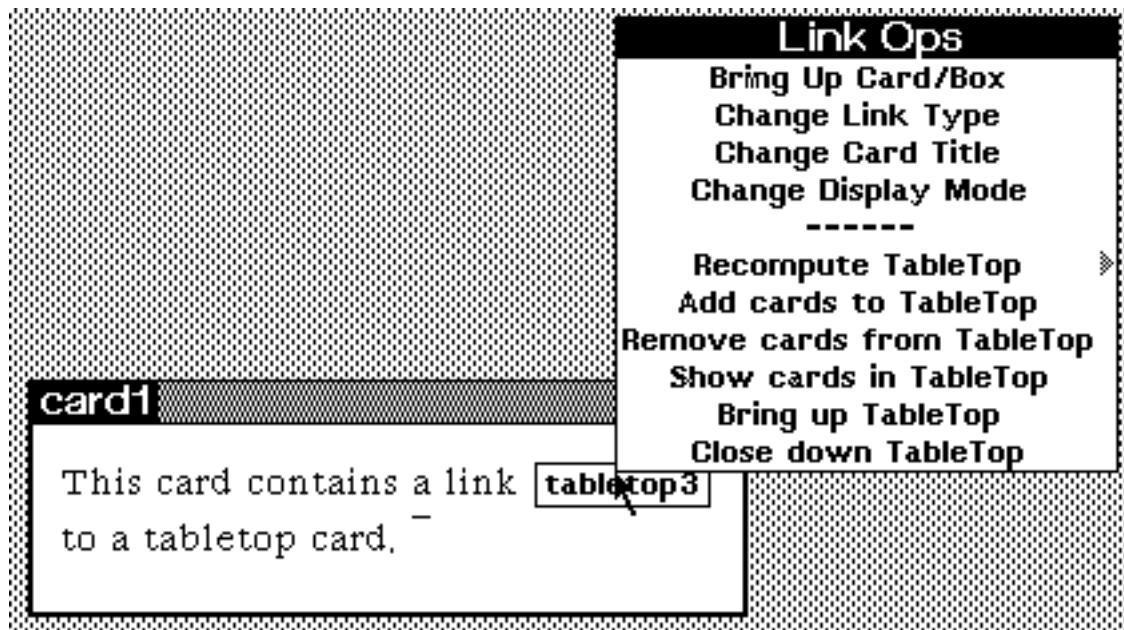
You will also be asked to select the cards to copy from the template. These cards will be copied for the new tabletop, preserving the title, size, shape, location, and default looks of the original cards.



If you don't use a template tabletop, you'll be asked to select a set of cards to form the contents of the tabletop. Select these in the usual way, by holding the shift-key down and left-buttoning in the card windows' titlebars. You'll notice that the tabletop card's window is rather small and has no directly editable substance. Nonetheless, the contents of a tabletop card can be easily modified through a set of tabletop operations available from the card's middle-button titlebar menu.



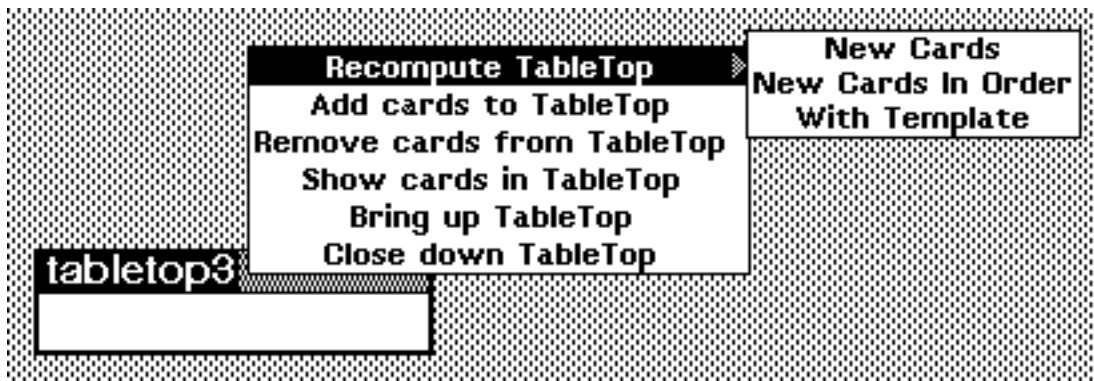
In addition, each of these operations is available from (the middle-button menu of) any link icon pointing at the tabletop card.



The operations available for tabletop cards are:

Recompute TableTop - Before choosing this operation, you presumably will have rearranged the cards in the tabletop changing some aspect of their layout. Choosing Recompute causes the new layout to be installed in the tabletop card.

Recompute TableTop pull across menu -



New Cards - This operation requires you to specify a new set of cards. That is, the tabletop is recomputed from scratch.

New Cards In Order - This operation requires you to specify a new set of cards, like the **New Cards** menu item. The order in which you choose the new cards is the order in which the cards will appear when you bring up the tabletop.

With Template - This operation asks you to select an existing tabletop to be used as a template for this tabletop.

Add cards to TableTop - This lets you add one or more cards to the tabletop by shift-selecting their windows' titlebars (or links icons pointing at them) in the usual manner. Once the selection is made, the tabletop is recomputed so that the new cards will be ordered correctly relative to the cards already in the tabletop.

Remove cards from TableTop - This brings up a menu containing the titles of all cards in the tabletop. Choose a card to remove by selecting from this menu. A second menu will then pop up containing the remaining cards. In this way, as many cards as desired can be removed from the tabletop. You'll be asked to confirm the removal of all selected cards as soon as you click outside of one of the card title menus.

Show cards in TableTop - This brings up a menu containing the titles of all cards in the tabletop. You can then select from this menu to bring up a single card in the tabletop at its assigned position/shape and scrolled properly.

Bring up TableTop - This opens all cards in the tabletop and positions, shapes, scrolls, shrinks and orders their windows according to the layout information stored in the tabletop. Any cards already open are moved into place and scrolled and shaped (or shrunk) appropriately.

Close down TableTop - This closes any open cards in the tabletop.

The above description covers the standard operation of tabletops. There are, however, two exceptional cases having to do with unopened cards in the tabletop. First, you might try to add a card to the tabletop whose window is not currently open

(or shrunken). In this case, the card's most recent position is used and it will appear in the bottom-most position in the window overlap ordering.

Second, you might try to recompute your tabletop even though one or more cards are not currently open. That causes something like the following

interactIOGETFN: BMOBJ.GETFN3ÓÑ

Of the three choices, **Cancel** aborts the Recompute operation. **Keep** causes the card to remain in the tabletop. (Note however, that it will be moved to bottom-most in the overlap ordering.) **Remove** causes the card to be removed from the tabletop.

INFORMATION STORED IN A TABLETOP CARD

As mentioned above, the tabletop card stores layout-related information for each of its cards. This includes: the card window region (i.e. shape and position); the scrolling positions (both vertically and horizontally if any); and whether the card's window is shrunken and if so, the position of its icon. Furthermore, the tabletop keeps the list of cards sorted according to the window overlapping ordering (a.k.a. the TOTOPW ordering). If the **Recompute** sub-command **New Cards In Order** was used to select the cards in the tabletop, then the list of cards is kept in the order specified.

In addition, extra information is kept for certain card types. For browser-based cards, tabletops record information about the shrunken browser overview window (if and where it is attached and its region) and whether the links legend and Editor menu windows are currently open. For Sketch-based cards, tabletops record whether the Sketch editing menu is attached.

PROGRAMMER'S INTERFACE TO TABLETOP CARDS

To create a new tabletop card programmatically, call `NCP.CreateCard` as follows:

**(NCP.CreateCard 'TableTop NoteFile Title NoDisplayFlg Props
ParentFileBoxes ListOfCards InterestedWindow)**

Here, all the parameters are as per the documentation of `NCP.CreateCard` except that `ListOfCards` should be a list of (open or shrunken) cards whose layout is to be the tabletop card's initial substance.

The following functions can be used to programmatically manipulate tabletop cards and their contents.

(NCTableTop.BringUpTableTop TableTopCardOrWindow DoNotUncacheFlg)

The tabletop is brought up, that is, its cards opened and layed out appropriately on the screen. `TableTopCardOrWindow` can be either a tabletop card object or the window for same. Normally, if the tabletop card itself isn't part of the tabletop being displayed, it is cached, the cards in the tabletop are displayed, and the tabletop card is uncached. A non-NIL `DoNotUncacheFlg` will prevent the uncaching of the tabletop card (this is useful, for example, when control of caching is desired at the Guided Tour level -- see the `NCGuidedTour` package documentation).

`NCTableTop.BringUpTableTop` returns the list of cards it brought up.

**(NCTableTop.OpenCardInTableTop TableTopCardOrWindow
InterestedWindow)**

This causes a single card within a tabletop to be opened and positioned, shaped, etc.

at its proper location. A pop-up menu is used to elicit the card to be opened from the user.

(NCTableTop.CloseDownTableTop TableTopCardOrWindow)

Close any open cards in the tabletop.

(NCTableTop.RemakeTableTop TableTopCardOrWindow ListOfCards RetainClosedCardsFlg NoToTopOrder)

Rebuild the tabletop to contain the cards appearing in **ListOfCards**. If **RetainClosedCardsFlg** is non-nil, then even closed cards in **ListOfCards** will be included in the tabletop. If **NoToTopOrder** is non-NIL, the cards are ordered in the tabletop according to the order in which they appear in **ListOfCards**, instead of by their overlapping order on the screen, which is the default.

(NCTableTop.RecomputeTableTop TableTopCardOrWindow NoToTopOrder)

Recompute the tabletop contents. Normally, the cards in the tabletop are open and arranged in a new arrangement when this call is made. If **NoToTopOrder** is non-NIL, the order of the cards in the tabletop is not altered according to their overlapping order on the screen, which is the default.

(NCTableTop.RemoveCardsFromTableTop TableTopCardOrWindow CardsToRemove)

Remove cards in the list **CardsToRemove** from the tabletop's substance. Returns the cards actually removed.

(NCTableTop.AddCardsToTableTop TableTopCardOrWindow CardsToAdd InterestedWindow)

Add cards appearing in **CardsToAdd** to the substance of the tabletop.

(NCTableTop.CardsInTableTop TableTopCard)

Return the list of cards currently appearing in the tabletop.

(NCTableTop.TableTopBasedP CardOrCardType)

Return non-nil if **CardOrCardType** is either a tabletop card or a card of a type that inherits from tabletop card. (It does the obvious thing if **CardOrCardType** is a card type.)

(NCTableTop.CollectCards RootTableTopCards LinkTypes MaxDepth FollowCrossFileLinksFlg)

This function is analogous to NCP.CollectCards. It starts from a list of tabletop cards given by **RootTableTopCards** and computes the transitive closure by following links of types appearing on **LinkTypes** to a maximum depth specified by **MaxDepth** where the depth 1 cards are the cards directly in the tabletop su

TextCardKeys

by Kirk Kelley (Kelley.pa@Xerox.COM)

{QV}<NoteCards>1.3k>library>TextCardKeys.dcom
Last updated: Jan 29, 1988

Introduction

Ever wish you could capture the structure of your ideas before they escaped short-term memory?

TextCardKeys sets special keys on your keyboard to create with one chord a new text card linked from your caret location.

Initial Settings

For a See link type: META asterisk (META SHIFT 9)

For an Include link type: META open paren (META SHIFT 9)

To close a card: META close paren (META SHIFT 0)

The new cards come up with default titles ready to be edited.

For See links, the title is an asterisk.

For Include links, the title is empty.

The new cards also come up already positioned on the screen.

The best way to see how it works is to load it and try it out. I use this with `ForcedFiling` turned off and find my hands rarely need to leave the keyboard.

Philosophy

TextCardKeys facilitate one of the proposed new ways of keeping track of cards. In this new way cards would always be linked somewhere. It is hoped this will eliminate the need for forced filing, Orphans, the To Be Filed box, un-filed cards, and all of the concepts and mechanisms surrounding them.

Changing Settings

The following is the nitty gritty for those that want to program different behavior or settings.

NC.CreateLinkedTextNoteChar [variable]

This is the key for creating a "See" link. It comes set to 298 (Meta asterisk).

NC.CreateLinkedTextNoteLinkType [variable]

This is a string containing the type of link for the above character. It comes set to "See".

NC.CreateLinkedTextNoteTitle [variable]

This is a string containing the initial title for the above link. It comes set to "".

NC.CreateLinkedTextInclusionChar [variable]

This is the key for creating an "include" link. It comes set to 296 (Meta open-paren).

NC.NC.CreateLinkedTextInclusionLinkType [variable]

This is a string containing the type of link for the above character. It comes set to "Include".

NC.CreateLinkedTextNoteTitle [variable]

This is a string containing the initial title for the above link. It comes set to "".

NCP.NewTextCardRegion [function]

Alter this to change where, and what shape, cards will automatically have when positioned on the screen. To turn off auto-positioning, have this return NIL.

If you change any of the above settings, I would be interested to know how you change them

One of the key components of CLOS in inheritance. The CLOS Browser provides functionality for displaying this structure and for extending it. It also provides functions for displaying and changing the class definitions and method definitions which make up a system written in CLOS.

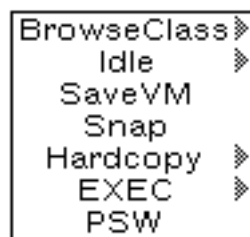
Creating a Browser

A browser can be created in two ways:

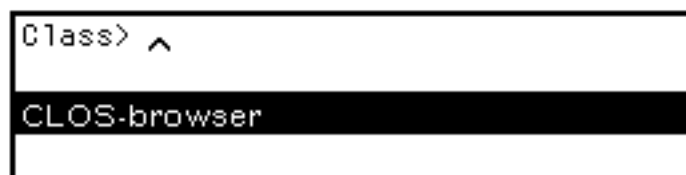
- Via a menu option from the Background Menu
- By calling the function CLOS-BROWSER:BROWSE-CLASS on a class

Creating a browser via the Background Menu

When the CLOS-BROWSER module is loaded, an entry is added to the Background Menu, as shown below:



Selecting the menu item BrowseClass brings up a window, with a prompt for the name of the class to use as the root of the browser as shown below.



Type in the name of the class you wish to browse at the flashing cursor, and the class graph will be drawn in the window.

Creating a browser programmatically

Browsers can also be created by calling the function BROWSE-CLASS:

(BROWSE-CLASS &OPTIONAL CLASS-NAME-OR-LIST &KEY :WINDOW-OR-TITLE :GOOD-CLASSES :POSITION) **[Function]**

This function brings up a browser on the class named or the list of classes named. If a window is supplied for the :WINDOW-OR-TITLE argument, then the browser is created in that window, else an appropriately sized window is created. The window is positioned at the :POSITION argument or, if not supplied, then the position is set via the mouse. If a text string is supplied for the :WINDOW-OR-TITLE argument, then that string is used for the window title, else the string "CLOS-browse" is used. If :GOOD-CLASSES is supplied, then only those classes in the list are displayed.

Using the Class browser

Instances of CLOS-BROWSER are operated on through a mouse-based interface.

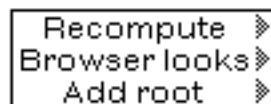
Buttoning on the browser will cause one of the following menus to be popped up:

- One menu appears when the left or middle button is pressed while the mouse is in the title bar. This menu has operations that apply to the browser itself.
- The other menu appears when the middle button is pressed when the mouse is on one of the nodes in the browser.

If the left button is pressed when the mouse is on a node, that node is boxed. This marks the node for some operations.

Options in the title bar menu

The following menu appears when you left- or middle-button in the title bar.



Recompute and it's suboptions

Selecting the Recompute option and dragging the mouse to the right causes the following submenu to appear:



Most of these items change the appearance of the browser, not the contents.

Recompute	Recomputes the browser from the starting objects. It does not recompute the labels for each node if those labels are cached in the Label-Cache slot of the browser.
Recompute Labels	Recompute the browser from the starting objects, including the labels.
Recompute inPlace	Recompute the browser without affecting the scrolled location of the lattice within the window.
Clear caches	Clear the caches of the nodes.

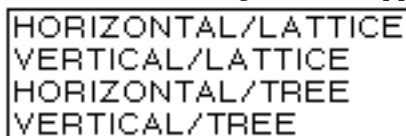
Browser looks and it's suboptions

Selecting the Browser looks menu item and sliding to the right causes the following submenu to appear:



Selecting one of these options changes the looks of the browser.

Shape to hold	Make the window for the browser just large enough to contain the browser.
Change font size	Causes a menu of alternative font sizes to pop up. Selecting one of these causes the browser to be redrawn with the nodes at that font size.
Change format	Causes the following menu to appear:



Horizontal/Lattice Lays out the grapher as an horizontal lattice.

Vertical/Lattice Lays out the grapher as a vertical lattice.

Horizontal/Tree Lays out the grapher as a horizontal tree.

Vertical/Tree lays out the grapher as a vertical tree.

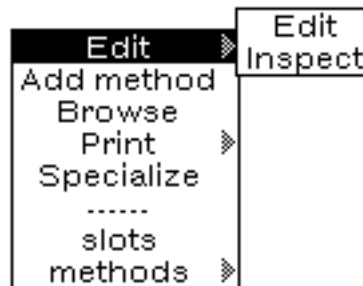
Options in the Middle-button menu

The following menu appears when you middle-button over a node in the graph:



Edit and it's suboptions

Selecting **Edit** causes an editor on the class definition to be brought up. Sliding the mouse to the right causes the following menu to appear:



Edit Edits the class named by the node

Inspect Inspects the class object named by the node.

Add Method

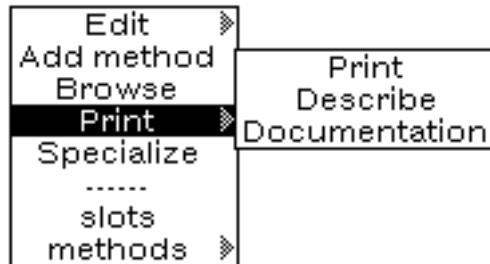
Selecting the **Add Method** option brings up an editor window with a template for a method to be added to that class. When the editor is done the method is installed for that class and the menu updated.

Browse

Selecting the **Browse** option causes a browser to be created starting with that class as the root.

Print and it's suboptions

Selecting Print prints out the class definition. Sliding the mouse to the right causes the following menu to appear:



Print	Print's the class definition
Describe	Describes the class, listing it's metaclass, it's supers classes, it's subclasses, it's CPL, and the number of methods specialized to it.
Documentation	Print's the documentation string for the class

Specialize

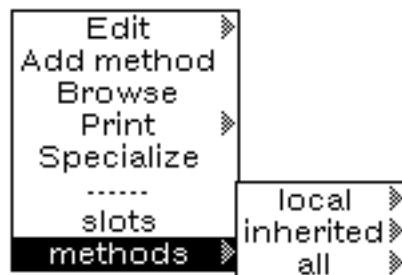
Selecting the Specialize option brings up an editor window with a template for a subclass to be added to that class. When the editor is done the class is installed and the browser updated.

Slots

Selecting the Slots option is the same as selecting the Edit option, it brings up an editor on the class definition.

Methods

The Methods option allows you to edit one of the methods defined for that class. Selecting it and sliding to the right brings up the following sub-menu:



Local	Bring up a menu of the local methods, ie methods directly defined for this class

Inherited	Bring up a menu of the methods this class inherits from it's superclasses.
All	Bring up a menu of all the methods defined for this class, both local and inherited.

Selecting an item with the left button from the resulting menu brings up an editor on that method. If there are multiple methods that apply, a gray triangle appears in the right edge of the menu next to that item. Sliding to the right brings up a menu of method specializers to select the appropriate method.

== IMINDEX ==

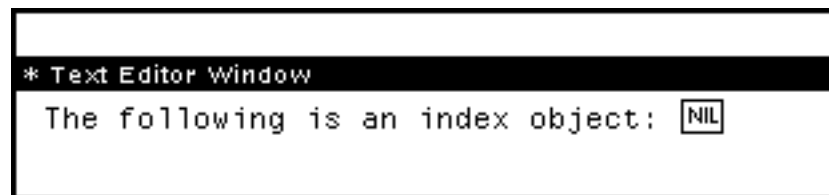
The file IMINDEX contains the functions used for creating and editing index image objects, and inserting them into a Tedit document. When a Tedit document containing index objects is formatted for printing, the index objects do not appear, but information about the index objects is put into an auxilliary "IMPTR" file. The functions in IMTOOLS can be used to take a set of IMPTR files, and generate an index.

Adding an Index Object to a Tedit Document

The simplest way of adding an index object to a Tedit document is to type ctrl-O while typing at Tedit, which will cause a window to pop up asking you to type a form to eval:



Typing (IM.INDEX.CREATEOBJ) will create an empty index object, indexing the term NIL, inserted in the Tedit document at the caret. Index objects appear in the Tedit window as words with boxes around them:

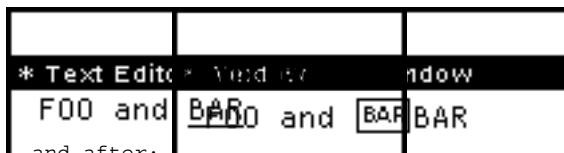


Using the IM Index Menu

An easier way to put many index objects into a Tedit document is to type (IM.INDEX.MENU) at the lisp exec, which will prompt you to position a menu that looks like this:



If the caret is blinking in a Tedit window, selecting [Index Selection as Term] with the left button will create and insert an index object that indexes the selected string. For example:



Before: and after:

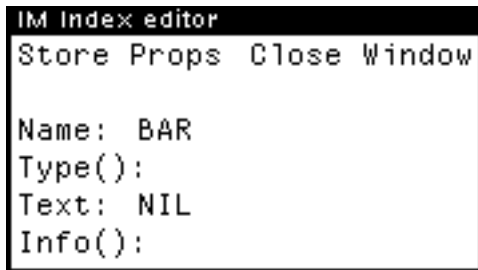
Selecting [Index Selection as Term] with the MIDDLE button will insert an index object that indexes the selected string, and then create an index object editing window, that can be used to edit the values of the fields in an index image object.

Editing an Index Object in a Tedit Document

If you select an index object, a menu will appear asking whether you want to edit the contents of the index object:



If you select [Edit Index], you will be prompted to position an index object editing window, which looks like the following:



This is a freemenu that allows you to edit the values of the fields in an index image object. Selecting [Store Props] stores the values in the editor into the object itself -- if the "Name:" field is changed, the Tedit window will change to show the new index name. Selecting [Close Window] will close the editor window. Note that any changes are lost if you close an index object editing window without storing the new property values.

Index Object Properties

There are more properties in an index object than show in the window initially; the window can be scrolled or reshaped to see and edit the other properties. The fields are interpreted as follows:

Note: The properties listed with "()", such as "Type():", interpret their value as a list of take a list of items, delimited by spaces. The other properties interpret their value as an atom, including spaces.

Name: Value is the name used to sort and merge index entries. This should normally be all-uppercase. The [Index Selection as Term] item in the IM Index menu will automatically uppercase the selection if it is not all-uppercase, and put the real value in the "Text:" field. Examples:
 Name: FOO
 Name: BAR

Type(): Value is the type of object being indexed. If NIL, this stands for an English term. Other types are used for indexing lisp functions, variables, etc. Note that upper/lower case is important. Examples:
 Type(): Function
 Type(): Editor Command

Note: The special types CHAPTER and SUBSEC (all-uppercase) are used to create entries in the table of contents, as described below. These cannot be used as index entry types.

Text: Value is the name actually printed in the index. If NIL, the value of the "Name:" property is used. This is usually used when indexing non-uppercase items. Another place where this is useful is when you want an item containing strange characters to appear in a different place in the index. For example, if Name: = FOO and Text: = *FOO*, the item "*FOO*" will appear in the index among the "F" items instead of the "*" items.

Info(): Value is a list of "information" words, that mean something to the indexing programs. There are a few info words likely to be used by IMINDEX users. One is the word *PRIMARY*, which indicates that this is a primary reference. In the index, primary index page numbers are printed first, in a bold font. Another is the word *NOPAGE*, which indicates an index entry that should not be printed with a page number. This can be used to generate entries such as "FOO, see BAR". Example:
 Info(): *PRIMARY*

SubSec(): Probably not much use for IMINDEX users. Value is a reverse list of nested subsection and chapter numbers. For example,
 SubSec(): 4 2 99
 indicates subsection 4 inside subsection 2 inside chapter 99. This is not used with normal index entries, except that chapter numbers, if given, are used when generating the page number in the index. For example, if the index entry specified chapter 99, and it was on page 5, it would appear in the index as 99.5.

Another use for this field is if you use IM index objects to generate a table of contents, by creating index objects with type of CHAPTER or SUBSEC, as described below. In this case, the SubSec field is used to specify the subsection numbers.

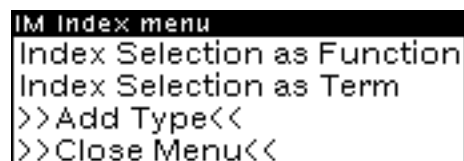
Page#: The value of this field is replaced with the page number when the Tedit textstream containing the index image object is hardcopied. Not much use to change, but it is possible.

SubName:
SubType():
SubText: Name, Type, and Text values for a sub-entry, if any, which appears under the main entry in the index.

SubSubName:
SubSubType():
SubSubText: Name, Type, and Text values for a sub-sub-entry, if any.

Adding New Types to the IM Index Menu

If all of the index entries in a document are terms, it is very easy to use the IM Index menu to index them. However, if there are a lot of Functions, variables, etc., you may need to edit each index entry to change the type. To make this easier, you can use the [>>Add Type<<] button in the IM Index menu to add new selections to the IM Index menu. If [>>Add Type<<] is selected, the system will prompt you to type a type name, and add a new item to the menu. For example, if you started with the initial menu, and added a new entry for the type "Function", the menu would be changed to look like the following:



```

IM Index menu
Index Selection as Function
Index Selection as Term
>>Add Type<<
>>Close Menu<<

```

If [Index Selection as Term] is selected, an index object will be created indexing the the current Tedit selection, with the type of Function.

Saving and Retrieving Tedit Documents with IM Index Objects

Tedit documents containing IM Index objects can be saved using the ordinary Tedit "Put" command, which will store all of the indexing information in the index objects. Note, however, that it is necessary to load the IMINDEX package before editing any Tedit documents containing IM Index objects -- otherwise Tedit will print the message "WARNING: Document contains unknown image objects", and all index objects will appear as the following:



```

Unknown IMAGEOBJ type
GETFN: IM.INDEX.GETFN

```

If this happens, it will not be possible to use these index image objects, or re-save the Tedit document. Close the Tedit window, load IMINDEX.DCOM, and call Tedit to edit the document again.

Hardcopying a Tedit Document Containing Index Objects

A Tedit document containing index objects can be hardcopied using the normal [Hardcopy] menu button, or from the Filebrowser. Index image objects will not appear in the final hardcopy. However, while the document is being formatted, the index objects will put indexing information in an auxilliary file with the extension "IMPTR". The first time an index object is formatted, it will try to open an imptr file on the connected directory, printing a message in the prompt window: "Opening index pointer file: {DSK}<LISPFILES>FOO.IMPTR...done". When the formatting is completed, another message will appear: "Closing index pointer file: {DSK}<LISPFILES>FOO.IMPTR;1...done".

The directory used for the imptr file is the currently-connected directory (which can be changed using the CONN command). The file name used is the file name of the Tedit file, if there is one. For example, if the Tedit file is XYZ.TEDIT, the imptr file would be XYZ.IMPTR. If the Tedit text stream doesn't have a name (for instance, if the Tedit window has been brought up but never saved) the file name NONAME.IMPTR is used. The extension is always IMPTR.

For Hackers Only: If there is a need to use a different filename for the imptr file, it is possible to specify this programmatically by changing textstream properties of the Tedit textstream before the hardcopy operation. If the value of the textstream property IM.INDEX.PTRFILENAME is non-NIL, it is used as the file name for the IMPTR file. If the value of the textstream property IM.INDEX.PTRFILE is non-NIL, it is the stream used for the IMPTR information.

Creating Indices and Tables of Contents From IMPTR Files

The file IMTOOLS.DCOM contain the routines used to create indecies and tables of contents for documents in IM format. These will also produce indicies and tables of contents from IMPTR files created by IM index objects. Loading this file will automatically load IMINDEX.DCOM, if it is not already loaded. Note, however, that you can load IMINDEX (for the purpose of editing Tedit documents with IM index objects) without loading IMTOOLS.

IMTOOLS contains the following useful functions:

(MAKE.IM.INDEX OUTFILE.FLG VOLUME.INFO IMPTR.FILES IMPTR.TYPES) [Function]

MAKE.IM.INDEX takes a number of IMPTR files, and produces an index, formatted like the one in the IRM. If OUTFILE.FLG is NIL, the output file is just sent to the default printer. If OUTFILE.FLG is T, the outfile textstream is simply returned. If OUTFILE.FLG = anything else, it is taken as a file name of an interpress file which is created <but not printed>.

VOLUME.INFO is a list of lists specifying which chapters are associated with which volumes in a large multi-volume document (like the IRM). For an example of the format, see the variable IM.MANUAL.VOLUMES in IMTOOLS. If NIL, no volume numbers are printed in the index.

IMPTR.FILES is a list of IMPTR files to be used to create this index. If NIL, any index info loaded in by explicitly calling GRAB.IMPTR is used (see IMTEDIT.TEDIT). When doing the IRM, all of the IMPTR files are loaded into global variables using GRAB.IMPTR, but users of IMINDEX probably want to specify the values specifically.

IMPTR.TYPES should be a list of IM index types to be included in the index, specified as lists. For example, if IMPTR.TYPES = ((Function)(Variable)), only function and variable entries would be listed in the index. If IMPTR.TYPES = NIL, all index entries in the IMPTR files are used.

(MAKE.IM.TOC OUTFILE.FLG CHAPTER.NUMBERS IMPTR.FILES) [Function]

MAKE.IM.TOC takes a number of IMPTR files, and produces a table of contents, formatted like the one in the IRM. All im index entries whose type is CHAPTER or SUBSEC are used as chapter and subsection pointers.

OUTFILE.FLG and IMPTR.FILES are interpreted similar to MAKE.IM.INDEX.

CHAPTER.NUMBERS is either: NIL, meaning to generate TOC of ALL data in the specified imptr files; a single number, meaning to generate a chapter TOC for that chapter; or a list of numbers, meaning to generate a TOC for those chapters.

```
author: Michael Sannella
file: {Phylum}<LispUsers>IMNAME. (& .DCOM)
loads file: {Phylum}<LispUsers>HASH.DCOM
loads file: {Phylum}<LispUsers>IMTRAN.DCOM [loaded by IMNAME.UPDATE.HASHFILE]
```

***** The IMNAME Database Package *****

IM format is the text formatting language that the Interlisp Reference Manual is represented in. It is somewhat like TEX source code, in that there are keywords, and brackets are used to delimit text. However, IM format was specifically designed for representing the Interlisp Manual, so the "text objects" used are semantically meaningful objects within the manual, such as "function definition", "lisp code", "subsection".

IMNAME is a package designed to help people who may frequently need to modify IM format files. Usually, a large document (such as the Interlisp Manual) is stored in a number of separate files, and it is difficult to know which file contains a particular piece of information. IMNAME contains functions for analyzing a set of IM format files and building a database of "IM Names" (functions, variables, property names, etc) with pointers to the files where they are defined. Using this database, other functions allow the user to specify an IM name, and bring up a Tedit window on the appropriate file, with the cursor positioned at the right place. This tool has been very helpful to the people updating the Interlisp Reference Manual.

***** Using IMNAME to access IM format files. *****

The normal way of using an IMNAME database is by creating an "IM name inspector". This can be done using either of the following two functions:

```
(MAKE.IM.INSPECTOR hashFileName)
or
(IMNAME hashFileName)
```

hashFileName should be the name of the IMNAME database hashfile to use for this inspector. One can create multiple IM name inspectors point to the same hashfile, or to different hashfiles. Currently, I know of only two IMNAME hashfiles; the one for the Interlisp Manual and one for the Loops Manual. If hashFileName is the atom INTERLISP or LOOPS, this Inspector will point to the appropriate hashfile. If hashFileName is omitted, it will assume that the Interlisp Manual hashfile is desired for people whose default host is Phylum, and the Loops Manual for people on Ivy. (Other people can set up their own default IMNAME hashfile by setting the global variable IM.NAME.DEFAULT.HASHFILE).

MAKE.IM.INSPECTOR sets up an "IM Inspector Window", which contains a menu. Initially, this contains the single selection "Type an IM name", which (when buttoned) prompts the user to type a name which will be looked up in the database. If an IM name is found in the hashfile, another window will appear below the first one, containing all of the "types" that the given name is known by. For example, a name may be known as both a variable and a function. If one of these types is selected, a third menu will appear below the second, listing references to the name in different files. Selecting one of the references will move the cursor in a TEDIT window (if there exists an active TEDIT window to the appropriate file), or create a new TEDIT window to the file. [Note: If a name is only known to be of one type, the "type menu" step is omitted.]

Other functions:

```
(INSPECT.IM imname hashFileName)
```

This allows the searching for a single IM name, using several pop-up menus. In general, it is easier to use the IM name inspector.

```
(GET.IM.NAME.LIST hashFileName)
```

Returns a list of all of the IM names known within the specified hashFileName.

***** Updating an IMNAME database. *****

IMNAME works by referencing a hashfile database containing IM names and pointers to files where these names are referenced. As a set of files is modified, this information will become obsolete. In particular, simply adding a few characters to a file will invalidate all pointers to positions later in the file. This can be tolerated for awhile (it is rare that IM format files will be changed enough such that the pointers are totally off), but eventually, it is necessary to update the IMNAME database, using the following function:

```
(IMNAME.UPDATE.HASHFILE oldHashFileName addFileList deleteFileList flushDisappearedFlg )
```

This function updates an IMNAME database hashfile. First, it looks at the list of files referenced in the hashfile, and determines which of these files have been updated, by comparing version numbers. Next, every updated file is reanalyzed with IMTRAN, and updated index information is stored into an in-core hasharray. Finally, the entries in the old hashfile are read in, merged with the new info, and written out to a new hashfile.

oldHashFileName is the name of the hashfile to be updated (Note that this file must be named explicitly --- no file searches are done so that the user will not inadvertently start updating the main manual database). The new hashfile will be created as the new version of the same file name. addFileList is a list of files that will be analyzed, and added to the database. deleteFileList is a list of files that will be deleted from the database. addFileList and deleteFileList can be used to "manage" a database, as new files are added to a document, and old ones are removed, split up, or renamed. flushDisappearedFlg determines what IMNAME.UPDATE.HASHFILE will do if it finds that some of the files in the database have disappeared (and they are not named on deleteFileList). If flushDisappearedFlg = T, the info for those files will simply be deleted. If flushDisappearedFlg = ERROR, IMNAME.UPDATE.HASHFILE will return without doing anything if files have disappeared. If flushDisappearedFlg = <anything else>, the info on the disappeared files will simply be retained.

To create a new IMNAME hashfile, pass a non-existent file name as oldHashFileName, and give a list of files as addFileList. In this case, a new hashfile will be created just from the internal hasharray info.

***** The IMTEDIT IM-to-TEDIT translation program. *****

author: Michael Sannella
file: IMTEDIT.DCOM
loads file: IMTRAN.DCOM
related files: IMTOOLS.DCOM

IM format is the text formatting language that the Interlisp Reference Manual is represented in. It is somewhat like TEX source code, in that there are keywords, and brackets are used to delimit text. However, IM format was specifically designed for representing the Interlisp Manual, so the "text objects" used are semantically meaningful objects within the manual, such as "function definition", "lisp code", "subsection". IM format is described in detail below.

IM format files are easy to edit using Tedit, but they don't look very pretty. To produce the manual, use the function IM.TEDIT, which translates IM format files to formatted Tedit text streams. These text streams can be proofread and edited by the user, or automatically printed.

A useful feature of IM.TEDIT is that is very forgiving about errors in IM format, even misplaced bracket errors! This is not to say that the output will be pretty, but at least the translation program will not bomb out on you.

***** Formatting a File with IM.TEDIT *****

To translate an IM format file into a formatted Tedit text stream, use the following function:

(IM.TEDIT INFILE.NAME OUTFILE.FLG) [Function]

This function takes an IM format file, and produces a formatted Tedit text stream. *INFILE.NAME* is the name of an IM format file. *OUTFILE.FLG* determines what happens to the translated textstream. If *OUTFILE.FLG* = T, the Tedit textstream is returned by IM.TEDIT. A Tedit window showing the translated document can be created by typing **(TEDIT (IM.TEDIT xxx T))**. If *OUTFILE.FLG* = NIL, the document is immediately sent to the printer. If *OUTFILE.FLG* = anything else, it is taken as a file name for the Interpress file which is created <but not printed>.

As IM.TEDIT runs, it prints out warning messages. These messages are also saved in the file <infile>.IMERR.

Important note: It is necessary to understand that IM.TEDIT produces a totally separate document from the IM format original. Any edits to this document will NOT be reflected in the original. In general, all of the editing should be done to the IM format files, to insure that there are no inconsistencies.

Note: If IM.TEDIT is called with *OUTFILE.FLG* = T to produce a textstream, and **IM.INDEX.FILE.FLG** (below) is set to T to add index information to the textstream, the textstream cannot be stored or printed. **IM.INDEX.FILE.FLG** should only be set to T if IM.TEDIT is called with *OUTFILE.FLG* not T, so it automatically prints the document to a printer or interpress file.

***** Variables Affecting IM.TEDIT *****

The operation of IM.TEDIT affected by a number of variables:

IM.NOTE.FLG [Variable]

If T, notes will be printed out, otherwise they will be suppressed. Initially NIL. (Note: If this is T, the translation programs will print out a message to remind you that notes will be printed.)

IM.DRAFT.FLG [Variable]

If T, the output will have "--DRAFT--" and the date printed on the top and bottom of every page. Initially NIL.

IM.CHECK.DEFS [Variable]

If T, checks whether variables and functions are bound/defined in the current Interlisp environment, and prints a warning if not. For functions, will also check arg list consistency. Initially NIL.

IM.EVEN.FLG [Variable]

If T, IM.TEDIT will add an extra page at the end of the file saying "[This page intentionally left blank]". This can be used if you need a document with an even number of pages (for double-sided copying). Initially NIL.

The following FLGs are only of interest when generating an index:

IM.INDEX.FILE.FLG [Variable]

If T, index information will be added to the formatted Tedit text stream, and the file <infile>.IMPTR will be generated containing index information when the formatted Tedit textstream is printed. Initially NIL.

IM.REF.FLG [Variable]

If T, the translation program will try to resolve cross-references by looking at various hash tables. Initially NIL.

IM.SEND.IMPLICIT [Variable]

If T, send "implicit references" for functions and variables (if not in index hash array). Initially NIL. {fn...} or {var...} objects generate "implicit references" if they are not contained in the index hash tables. This can be used to find variables and functions that are not formally defined, but only mentioned.

***** Producing an Index and Table of Contents *****

The file IMTOOLS.DCOM contains functions for gathering index information, generating an index and a table of contents, and using index information to resolve cross-references.

Currently, these tools are rather primitive. Eventually, it is hoped that they will be improved to make this process easier.

To produce an index and TOC:

(1) Evaluate (SETQ IM.INDEX.FILE.FLG T), which tells IM.TEDIT to create index pointer files <XX>.IMPTR.

- (2) Run (IM.TEDIT <file> <ipfile>) on all files in the document
- (3) Evaluate (INIT.INDEX.VARS) to clear the index.
- (4) For each of the files in the document, load the index information by evaluating (GRAB.IMPTR xxx.IMPTR).
- (5) Generate an index by evaluating (MAKE.IM.INDEX OUTFILE.FLG), where the argument OUTFILE.FLG is interpreted the same as in IM.TEDIT.
- (6) Generate a table of contents by evaluating (MAKE.IM.TOC OUTFILE.FLG), where the argument OUTFILE.FLG is interpreted the same as in IM.TEDIT.

To generate formatted files with cross-references resolved, do the following:

- (1) Evaluate (SETQ IM.INDEX.FILE.FLG T), which tells IM.TEDIT to create index pointer files <XX>.IMPTR.
- (2) Run (IM.TEDIT <file> '{NULL}FOO.IP) on all files in the document
- (3) Evaluate (INIT.INDEX.VARS) to clear the index.
- (4) For each of the files in the document, load the index information by evaluating (GRAB.IMPTR xxx.IMPTR).
- (5) Evaluate (SETQ IM.REF.FLG T), which tells IM.TEDIT to resolve cross references using the loaded index information.
- (6) Format all of the files using (IM.TEDIT <file> <ipfile>).

Note that creating a formatted file with cross-references requires formatting the file twice.

The process of formatting a large set of documents such as the IRM can be automated using the following function:

(DO.MANUAL CHAPNAMES MAKE.INDEX.FLG GET.REFS NO.IP.FLG) [Function]
 Formats the IRM manual chapters specified by CHAPNAMES, producing Interpress files, "imptr" files (explained below), and error files. Uses the global variable **IM.MANUAL.DIRECTORY** (initially {erinyes}<lispmanual>) to indicate where the IRM files are taken from, and where the processed files should go.

Note: **DO.MANUAL** initially puts all files on {DSK}, and then copies them to the value of **IM.MANUAL.DIRECTORY**. Therefore, you should have at least 3000-4000 free pages on DSK.

"IMPTR" files are files with the extension "**IMPTR**" (such as ChapLitatoms.IMPTR), which contain index info, including the page number where each index appears. These are generated whenever a chapter is formatted. These must be read in to generate an index, or format a chapter resolving cross-references.

CHAPNAMES indicates which chapters should be processed. If *CHAPNAMES* = a list of chapter file names such as (ChapLitatoms ChapStack), only these chapters are formatted. If *CHAPNAMES*=**T**, all chapters in the IRM are formatted. If *CHAPNAMES*=**NIL**, all chapters that have been modified since the corresponding Interpress files were made are formatted. The global variable **IM.MANUAL.CHAPTERS** specifies all of the chapters, and all of the sub-files in each chapter, and the chapter numbers.

If *MAKE.INDEX.FLG* is non-**NIL**, after processing the chapters, the index, title pages, and many tables of contents are generated. Note that if this is done, all of the IMPTR files for all of the chapters are loaded again.

If *GET.REFS* is non-**NIL**, all of the IMPTR files are loaded before any of the chapters are processed, and all cross-references in the chapters are resolved.

If *NO.IP.FLG* is non-**NIL**, the chapters are formatted to the interpress file **{NULL}FOO.IP**, and thus not kept. This can be used when you simply want to generate all of the IMPTR files, but do not want the interpress files.

Ideally, all you should need to do to re-format the IRM is:

(DO.MANUAL T NIL NIL T) --- to generate the IMPTR files

(DO.MANUAL T T T) -- to generate the Interpress files, and the index.

If **DO.MANUAL** doesn't do exactly what you want, the code is fairly self-explanatory.

***** Modifying the dimensions of the formatted pages *****

Note on modifying the formatting tools for different paper sizes: All of the lengths that IMTEDIT uses (left margin size, page height, etc.) are stored in global variables, set in the filecoms of IMTEDIT. IMTEDITCOMS also includes comments documenting the meaning of each of these global variables. Modify these variables until you get what you want.

***** Known problems with IMTEDIT *****

Currently, IM.TEDIT produces a good-looking document. However, there are some features that I was not able to provide, because Tedit did not provide the formatting capability. These are as follows:

(1) Footnotes. Tedit does not supply footnotes. Currently, I "fake" footnotes by positioning them in-line after the paragraph wherein they were created. They also will not appear within definitions, tables, or lisp code.

(2) Tables. Tedit doesn't do table formatting. Currently, IM.TEDIT simply prints out the items in the table without formatting.

***** IM Format Description *****

This is a complete description of the syntax and vocabulary of "IM format". The general syntax is not likely to change, but the vocabulary will probably be extended as the Interlisp Manual is edited, and we discover new text objects that we need.

IM Format Syntax

An IM-format file consists of a linear string of visible, editable characters (and Tedit image objects, which are treated like characters). No funny control characters are allowed. "Text" is defined to be a linear string of characters, organized into paragraphs (delimited by blank lines), interspersed with any number of "Text Objects". Text Objects are used to specify the meaning of various pieces of text, which may be processed and formatted in different ways.

Text Objects can be divided into two types: those that take a single unnamed argument, and those that take a number of labeled arguments. All of the 'arguments' to Text Objects can be arbitrary text, organized in paragraphs, and including sub-text-objects nested to any level.

Text Objects within a file and arguments within a text object are specified using the characters "{" and "}". These characters are ALWAYS interpreted as Text Object delimiters or Text Object argument delimiters---there are special text objects for indicating that you really want a left or right bracket character as part of your text.

The format of Text Objects is:

```
{<TOname> <TOarg>}          -      single argument TO
{<TOname> {<TOargname1> <TOarg1>} {<TOargname2> <TOarg2>} .... }  -      multi-
argument TO
```

Between a TOname or TOargname and its TOarg, and between named TOargs, there can be any combination of spaces, tabs, and CRs. These can be used to make your file look better.

In order to help with the problem of matching brackets in large TOs (such as SubSecs, which can extend over many pages), I have introduced a new piece of syntax: Begin and End:

```
"{Begin <TOname> <tag>}"      is treated exactly like "{<TOname> "
"{End <TOname> <tag>}"        is treated exactly like "}"
except that additional checks are made when the file is processed that the "Begin"s and "End"s
match up. The (optional) "<tag>"s can be used as an additional check, to distinguish between
different TOs of the same type. If you use a Begin TO, you should use a matching End TO.
```

Note that Begin and End TOs can be used with TOs that have labeled args, for example:

```
{Begin SubSec <tag>}
{Title ----}
{Text
.
.
}
{End SubSec <tag>}
```

In order to allow the IM format translation program to recover from errors (like missing brackets), information has been included in the program about what TOs "should" appear in other TOs. In general, "complex" TOs such as LabeledLists can only appear in other complex TOs. "Simple" TOs such as Lisp can appear within anything. These rules are defined on a per-argument basis for TOs with multiple labeled arguments --- for example, a LabeledList can appear within the "Text" argument of a FnDef, but not within the "Name" argument. This feature should not cause any problems, as long as TOs are used "reasonably."

Other rules followed by the translation program: FnDef's, VarDef's, Def's, etc can only appear at the "top-level", or inside a SubSec or Chapter. (this rule is very useful, because it helps trap many bracket errors before they propagate too far) Footnotes may not be nested.

New feature <which may get changed>: If a TO name is unrecognized, it will be printed out surrounded by brackets. For example, {FOO} will actually print as "{FOO}". This allows simple expressions to be bracketed without actually using the "{bracket ...}" TO.

Text Objects currently defined:

(note: the order of arguments in multi-argument TOs IS significant.)

(note: No distinction is made between upper and lower case in the TO names and the argument names, or in Begin/End tags)

(note: Many TO names and argument names have synonyms. These are indicated below.)

***** Plain Text Text objects *****

<paragraph>

All plain text is organized into paragraphs, delimited by blank lines.

{Chapter {number <number>} {title <title>} {text <text>} }

Specifies the number, title, and text of a chapter. If the number is not specified, the IM format translator will ask you to supply a number.

{subsec {title <title>} {text <text>} }

This can be used to generate sections, subsections, etc. to any depth. Heirarchical numbering is done automatically.

{Comment <text>}

Used to insert comments (which won't appear in the final formatted output).

{Note <text>}

Inserts comments that may be printed in the final formatted output, depending on the value of the variable IM.NOTE.FLG. Should be used for comments such as {Note I should write something about X here}

{foot <text>}

Generates a footnote. Footnotes may not occur within other footnotes. Currently, Tedit does not support footnotes, so these are just printed after the paragraph in which they appear.

Synonyms: footnote -> foot

{sub <text>}

Subscripts <text>.

{super <text>}

Superscripts <text>.

{it <text>}

Used to italicize pieces of text.

Synonyms: italics, emphasize -> it

{rm <text>}

Used to print text in the default, "roman" font.

{lisp <text>}

The text object for normal single-line references to lisp code. This is used for writing things like: "Obviously, {lisp (CONS 'A 'B)} evaluates to {lisp (A . B)}."

{lispcode <text>}

The text object for multiline lisp code, which do not appear in the middle of text, and have to be formatted differently. Spaces, tabs, and carriage returns are significant within <text>.

***** TOs for Interlisp manual objects *****

{FnDef {Name <name>} {Args <args>} {Type <keywords>} {Text <text>} }

This is used to define all lisp system functions. It needs to know the name of the function, and the args, and the text of the function description. (If the function has 0 args, the {Args --} argument may be omitted.) {Type ...} is an optional argument used to specify the argument type of the function. If the keywords NLambda or NoSpread (case doesn't matter) are included in <keywords>, the function is specified to have the corresponding argument type.

Synonyms: FnName -> Name

 FnArgs -> Args

 FnType -> Type

{vardef {name <name>} {text <text>} }

Used to define system variables.

{propdef {name <name>} {text <text>} }

Used to define property names.

```
{MacDef {Name <name>} {Args <args>} {Type <keywords>} {Text <text>}}
```

Like FnDef, but for macros.

```
{arg <name>}
```

Used to talk about an abstract argument to a function, such as "x" or "y" or "number". It is used primarily within function definitions.

```
{fn <name>}
```

Used to talk about the name of a function. i.e.: "...it calls {fn CONS} to do something."

```
{var <name>}
```

Used to talk about a system variable.

```
{prop <name>}
```

Used to talk about a property name.

```
{Mac <name>}
```

Like Fn, but for macros.

```
{EditCom <name>}
```

Used to talk about an edit command.

```
{BreakCom <name>}
```

Used to talk about a break command.

```
{PACom <name>}
```

Used to talk about a programmer's assistant command.

```
{FileCom <name>}
```

Used to talk about a file package command.

***** Indexing Text objects *****

The following TOs (for indexing, defining, and referencing) all deal with objects with specific "object-types." The Interlisp Manual contains a very large number of names (CONS, NIL, FOO, etc). It is not enough just to list the name in the index --- it is also important to indicate WHAT the name is (a function, a variable, an error message, etc.) An "object-type" is essentially the description that would be printed in the index to describe a particular name. In IM format, such a description is given by a list of words, within parenthesis. [note upper/lower case are NOT distinguished in "object-types"] For example:

(Function)
 (Error Message)
 (Transor Command)
 (Compiler Question)

Some commonly-used object-types may be abbreviated with single words:

FN	->	(Function)
Var	->	(Variable)
Prop	->	(Property Name)
BreakCom	->	(Break Command)
EditCom	->	(Editor Command)
PACom	->	(Prog. Asst. Command)
FileCom	->	(File Package Command)
Error	->	(Error Message)
Litatom	--,	
Atom	--->	(Litatom)

[note: In {PageRef ...} and {SectionRef ...}, the word TAG can be used to refer to tags, as specified below. The word FIGURE is a synonym for TAG; this can be used to refer to a figure tag. The word TERM can be used in {index ...} to index an English term.]

{index <text>}

Creates an index reference. Some of the text objects (such as function definitions) will automatically create an index reference, but it is useful to be able to create one explicitly. This should have the format:

{index <keywords> <object> <object type>}

<keywords> (optional) can be any combination of the words

BEGIN

END

PRIMARY

<object> can be any number of words.

<object type> should be an object-type as specified above.

If this is omitted (<text> does not end with ") and

the last word in <text> is not one of the special object-type words), then this is the index of a term.

This can also be specified by using the word TERM.

Examples: {index *PRIMARY* BLOBBY (Transor Command)}
 {index SELF-DESTRUCT SEQUENCE INITIATED Error}
 {index *BEGIN* *PRIMARY* file names}

{indexX {name <name>} {type <type>} {info <info words>} {text <text>}}

{subname <name>} {subtype <type>} {subtext <text>}

{subsubname <name>} {subsubtype <type>} {subsubtext <text>}}

Used for creating special index references whose printname is different from the "index name", or who have sub-index entries, or should not have page numbers in the index. <name> is the index name, used for referencing the object, and alphabetizing the index entry. <type> is the object type. <info words> (optional) can contain one or more of the keywords *BEGIN*, *END*, *PRIMARY*. <text> is the text printed in the index for this index entry.

Example: {indexX {name +} {type (Infix Operator)} {text {lisp {arg X}+{arg Y}}}} could be used to index "+" as an infix operator so that it would appear in the index alphabetized near other "+"s, but printing as "A+B".

The arguments {subname <name>} {subtype <type>} {subtext <text>} {subsubname <name>} {subsubtype <type>} {subsubtext <text>} are all optional. These are used to specify 1st and 2nd level subentries in the index. The subentry can have their own type and text definition, although these are mostly terms.

If <info terms> includes *NOPAGE*, this indicates that the index entry should not have a page number associated with it. This can be used with subentries to create "See also..." notes. Within a list of subentries, the *NOPAGE* entries will always be listed last.

Example: The following set of indexx commands:

```
{indexX {name FOO} {type (Big Command)} {text {lisp /FOO/}}}  
{indexX {name FOO} {type (Big Command)} {subname ZZZ}}  
{indexX {name FOO} {type (Big Command)} {subname BAZ}}  
{indexX {index *NOPAGE*} {name FOO} {type (Big Command)} {subname See also BAR}}  
{indexX {name FOO} {type (Big Command)} {subname BAZ} {subsubname QWERTY}  
{subsubtype VAR}}}
```

Would produce the index entries:

```
/FOO/ (Big Command) x.xx  
  BAZ x.xx  
    QWERTY (Variable) x.xx  
  ZZZ x.xx  
  See also BAR
```

```
{Def {type <type>} {name <name>} {printname <pname>} {args <args>} {parens} {noparens}  
{text <text>} }
```

May be used to specify the definition of anything. <type> should be an object-type, as specified above. <name> should be the name of the object, ideally a single word. If {printname <pname>} (optional) is given, it will be used for printing the object in the top line of the definition and in the index. If {printname <pname>} is not given, the printed name is specified by {name <name>}, {args <args>}, {parens}, and {noparens}. <args> should be the "arguments" of this object. [{args <args>} is optional.] Normally, {Def will put parenthesis around the object if and only if {Args <args>} is given. {parens} and {noparens} (if given) can be used to change this default behavior. Examples:

```
{Def {Type (Blobby Command)} {Name DOBLOB} {Args A B}  
  {Text ....}}  
{Def {Type (CLISP Character)} {Name +} {PrintName {lisp {arg X}+{arg Y}}}  
  {Text ....}}}
```

```
{Tag <tag>}
```

This may be used to associate <tag>, a single word (upper/lower case doesn't matter), with a particular place in the text. This can be referenced by {SectionRef and {PageRef as described below.

```
{SectionRef <object-type> <object>}  
{PageRef <object-type> <object>}
```

These TOs are used to provide a cross-reference facility. {SectionRef will print "section " and the section that the "main" occurrence of the object with the given type occurs. <"main"

shall remain undefined for now> {PageRef prints "page " and the appropriate page number. If <object-type> is "Tag", these TOs can refer to tags generated with {Tag <tag>}. Examples:
 {SectionRef Fn CONS} could print "section 3.1.1"
 {PageRef (File Package Type) Expressions} could print "page 14.34"
 {PageRef Tag CompilingCLISP} could print "page 20.42"

Note: {PageRef...} and {SectionRef...} can only be used to reference 1st level index references. They cannot be used to reference subentries generated with {indexx...}

{Term <text>}

Prints <text> exactly as given, and also puts it into the index. Equivalent to "<text>{index <text> Term}".

{Figure {tag <tag>} {text <text>} {caption <text>} }

Prints a simple "figure", with figure text and caption as specified by the "text" and "caption" arguments. "{tag <tag>}" is an optional argument which, if specified, generates a tag at the beginning of the figure. This tag can be used to refer to the page or section that this figure appears on (using {PageRef Figure <tag>} or {SectionRef Figure <tag>}), or to refer to the figure number (using {FigureRef <tag>}).

{FigureRef <tag>}

Prints "figure" and the figure number of the figure with the specified tag.

***** Complex Text objects *****

{numberedlist {item <text>} {item <text>} {item <text>} ...}

Used for making numbered lists of items.

Synonyms: Text -> Item

{unnumberedlist {item <text>} {item <text>} {item <text>} ...}

Like {numberedlist ...}, except that it uses bullets to mark each item in the list.

Synonyms: UnlabeledList -> Unnumberedlist

Text -> Item

{labeledlist {name <text>} {item <text>} {name <text>} {item <text>} ...}

Used for making a table associating a set of labels with descriptions of their meanings.

The argument {LName <text>} can be used to add a label left-justified on the page.

Either an LName or a Name, or both, can appear as labels for an item. If an LName is supposed to be on the same line as a Name, it should appear first: {Labeledlist {LName xx} {Name yy} {item zz} ...}

Synonyms: Label -> Name

Text -> Item

{Table {Column} ... {Column} {First <text>} {Next <text>} {Next <text>} ... {First <text>} ... }

Used for formatting multi-column tables. The number of {Column}'s before the first {First or {Next indicates how many columns the table should have. If no {Column}'s are given,

the default is three columns. There are other formatting arguments, but they will probably change. <<<Note: This is not currently supported in Tedit. Currently, this TO only prints out the values of each {First} or {Next} argument, without formatting. Do not use tables>>>

***** Other Text objects *****

{Include <filename>}

Used to include the text of one file within another. Files may be nested arbitrarily deep.

{Lbracket}

{Rbracket}

{bracket <text>}

These are used to insert the curly-bracket symbol into the text. {Lbracket} and {Rbracket} insert left and right brackets, and {bracket <text>} puts brackets around the given piece of text.

The following TOs are used to specify certain non-typeable characters.

{CRsymbol} or {cr} - carriage return symbol

(currently a superscripted 'CR')

{pi}	-	Greek letter pi
{plusminus}	-	Plus or Minus ("+" + "-")
{GE}	-	greater than or equal (">" + "=")
{LE}	-	less than or equal ("<" + "=")
{NE}	-	not equal ("=" + " ")
{endash}	-	short dash
{emdash}	-	long dash
{ellipsis}	-	ellipsis (...)
{bullet}	-	bullet
{anonarg}	-	long dash (should be used in the argument list of a function definition to specify the "unspecified arguments" specified in the current manual with a dash)

FORMATTING AN INDEX

Creating a properly formatted index takes a certain amount of work (about 2 hours).

- Type in your Executive (TEDIT (MAKE.IM.INDEX T NIL '(filenames) NIL))
- Save the resulting file in {ERIS}<Doc>LoopsBeta>Ref>RefIndex-Raw.tedit.
- Now start dithering. The task is to change all the separators in the file. For example, the index program returns pages in the form 1.2; 10.5,38; 18.3-4. The Index format we use is in the form 1-2; 10-5; 10-38; 18-3. Unfortunately, you can't make all the necessary changes in a global substitute, since many of the separators are also in the text and the comma requires that you repeat the chapter number. Here goes:
 - Fix the commas. Do a Find on a comma (.). Everywhere a comma occurs in a page number, change it to a semicolon (;) and repeat the chapter number and a dot before the page number. For example, 10.5,38 becomes 10-5; 10-38.
 - Fix the dashes. Do a Find on a dash (-). Everywhere a dash occurs in a page number, delete it and the following page number.
 - Fix the period. Except for the form #., you can use a global substitute from a period to a dash (. to -). This takes about 20-30 minutes, as there are about 700 substitutions.

I really recommend saving the file at this point.

- Hardcopy the index. Make whatever page breaks and other changes you feel are necessary. Make a final hardcopy and save the file.

ARCLEANUP Package --- Michael Sannella
(and augmented by Susana Wessling, 26-mar-87)

The file {eris}<lispcore>internal>library>ARCLEANUP.LCOM contains a few functions used to update the AR database and to print summaries.

```
(AR.CLEANUP UPDATE.FLG INDEX.LOCAL.DIR SUMMARY.FLG SUMMARY.LOCAL.DIR ]
```

This is the main function, which should be run every few days, preferably just as you go out the door (so your machine can crunch on it overnight). If UPDATE.FLG is non-NIL, all of the ARs that have been touched since the last update are scanned, and the AR database is updated. If SUMMARY.FLG is non-NIL, a set of summaries is generated on {eris}<lispars>summaries> and press-fied.

INDEX.LOCAL.DIR and SUMMARY.LOCAL.DIR are used if you want the AR index and/or the summary reports to be cached on a local disk. This speeds up the process considerably, along with reducing ethernet load. If non-NIL, each of these should be a host/directory pair, which will be PACKed on the front of a filename to generate the caching filename. These are not just flags, with caching defaulting to {DSK}, so that I can use multiple partitions for caching. The issue is space: Sometimes the local disk does not have enough room for two copies of the AR index, or for both the txt and press version of the grang summary.

Example:

```
AR.CLEANUP (T NIL T NIL)
-- does cleanup with no caching

AR.CLEANUP (T {DSK} T {DSK2})
--- caches index on {DSK}, and the summaries on {DSK2}

AR.CLEANUP (T {DSK})
--- just does index update, caching on {DSK}.
```

Another useful function:

```
AR.CLEANUP.REDO.SUMMARIES: just generates summaries, does not do a new cleanup.
```

The AREDIT Interlisp bug database system

author: Michael Sannella

files: {eris}<lispcore>library>AREDIT.DCOM

doc: {eris}<lispcore>library>AREDIT.TEDIT

uses: All Tedit files

The file AREDIT.DCOM contains a number of tools useful for examining, editing, and submitting ARs ("Action Requests") related to the Interlisp-D system. These tools are loosely based on the "Adobe" tools in the Tajo environment. The Interlisp-D support group uses this system to keep track of the state of outstanding bug reports. There are currently over 2000 ARs in the database.

These tools can be used from any machine running Interlisp-D which can establish a leaf connection to the Phylex: file server, where the database files are currently stored, and the ERIS file server.

After loading AREDIT.DCOM, the user can create two types of windows: AR edit forms and AR Query forms.

The AR Edit Form

An AR edit form is used to examine, edit, and submit ARs. To create an AR Edit Form, evaluate (AR.FORM). Interlisp will prompt you to specify a region for the form window -- the best size to give it is one about half the width of the screen and at least half the height of the screen. The form window which will appear contains three subwindows: (1) On the top is the message subwindow, where prompts and status messages are printed; (2) in the middle is the command subwindow, a menu of commands for editing / submitting ARs; (3) on the bottom is the form subwindow, where the information in an AR is displayed.

The command subwindow contains the following commands:

New -- Buttoning this word clears the fields of the AR in the form subwindow. Some fields (**Source**, **Submitter**, **Status**) are initialized to appropriate values for a new AR.

Get -- Buttoning this retrieves the AR whose number follows "**Number:**" in the command subwindow.

Put -- Buttoning this will either store an edited AR, or submit a new AR. Which one (submit new or store old) depends on whether the last operation was "**New**" or "**Get**". If the current AR displayed was retrieved with "**Get**", then "**Put**" will store it as the old AR. If this AR was built up from scratch after buttoning "**New**", then "**Get**" will submit it as a new AR. The title of the form subwindow gives an indication of what state the form is in: if it says "New Bug Report", then "**Put**" will submit it. If it says "Editing AR xxx", then "**Put**" will store it. [There are plans to improve this interface]

Number: -- This is a text field just like text fields in the form window (see below) used to specify the number used by "**Get**". Buttoning the word "**Number:**" will pending-delete-select the value of the field, so you can delete it and insert a new number. If the character carriage-return is typed, then a "**Get**" is automatically done on the value of this field. This is faster than typing a number, and buttoning "**Get**".

The form subwindow contains a large number of fields. The meaning of these fields is described in XXX. The value of these fields can be edited as follows:

"Enumerated fields" can only contain certain values. These are indicated in the form subwindow by field names followed by curly-braces "{}". To change the value of one of these fields, button the field name; a menu of permissible values will appear; select a value; it will be inserted between the braces.

[note: Some of the enumerated field values are dependent on other fields. For example, the values of "Subsystem:" depend on the value of "System:" -- if the "System:" value is changed, the "Subsystem:" value is automatically set to NIL. The fields with this relationship are System:/Subsystem: and Machine:/Disk:.]

"Text fields" can contain arbitrary text. These fields do not have braces after the field name. The text can be edited using normal Tedit editing. Buttoning the field name will pending-delete-select the entire field value, which allows the whole field to be easily deleted.

[note: Currently, stored ARs only contain straight text. Any tedit formatting information put into an AR will be lost when the AR is stored. Image objects (like bitmaps) are also not stored.]

[note: A few of the text fields, like "Number:", are read only --- they cannot be edited by the user.]

[note: in older versions of AREDIT, some text fields (such as "Attn:") could only contain a certain number of characters. The user could type as many characters as he wanted, but an error would occur when the "Put" command was executed. This has now been changed --- any text field can contain an arbitrary number of characters.]

The AR Query Form

An AR Query Form is used to search the AR database for all ARs with particular characteristics. One can search for all ARs with a given name in the "**Attn:**" field, all ARs which have **Status:** = Open, etc. These ARs can be sorted, and a summary of the selected ARs can be printed into a file.

To create an AR query form, evaluate (AR.QFORM.CREATE). Interlisp will prompt for a region (the default size is ok), and create a window with three subwindows: (1) on top, a message subwindow, for printing prompts and messages; (2) in the middle, a browser subwindow, used for displaying the ARs selected by a query; (3) on the bottom, the AR query command subwindow, containing a number of commands and fields.

The Ar Query command subwindow contains the following fields/ commands:

Query List: -- This field is used to specify which ARs the "**Query**" command will search for. This field should be filled with an AR query spec, which has one of the following forms:

(<field> HAS <val>) searches for all ARs whose text field <field> contains <val>. <string> may either be an Interlisp string or an atom. The search is case-independent: foo matches Foo matches FOO.

(<field> IS <val>) searches for all ARs whose enumerated field <field> has the value <val>.

(AND <spec1> <spec2> ... <specN>) returns all ARs satisfying ALL of the given specs.

(OR <spec1> <spec2> ... <specN>) returns all ARs satisfying ANY of the given specs.

[note: an implicit (AND) is wrapped around the value of the "Query List:" field, so just giving a number of specs will AND them together.]

Not every AR field can be searched for in the same way: some can only be searched with HAS, some can only be searched with IS, and some (like the Description: field) cannot be searched at all. To find out the possible query specs, button the words "**Query List:**" --- this will put up a menu of all of the permitted searching options. Some of these menu items have submenus. When one of the options is selected, it is added at the end of the value of this field.

Examples:

Query List: (Subject: HAS foo)

Searches for all ARs whose subject contains the string "foo".

Query List: (Status: IS Open) (Attn: HAS sannella)

Searches for all open ARs which have "sannella" in the Attn: field.

Query List: (OR (Status: IS Declined) (Status: IS Superceded))

Searches for all ARs with Status: either Declined or Superceded.

Sort List: -- This field determines how the Query-ed ARs will be sorted. Currently, ARs can only be sorted by the values of enumerated fields. Buttoning the words "**Sort List:**" will put up a menu of the permitted field names -- selecting one will add it to the value of this field.

Example:

Sort List: Status: Priority:

This will sort first by the Status: field, and then by the Priority: field.

[note: After sorting by all given fields, if two ARs are the same, they are sorted by AR number. Therefore, if this field is left blank, the queried ARs will be in numerical order]

Query -- Buttoning this command will initiate the query specified by the "**Query List:**" field, and sort it according to the value of the "**Sort List:**" field. While the query is in progress, the AR query command subwindow is greyed-out. When the query is completed, the numbers, subjects, etc of the ARs which have been found are displayed in the AR query browser subwindow. This window can be scrolled both vertically and horizontally.

Print File: -- This field can be filled in with a file name, which is used to specify the file that the **Print** command should store a report. If left blank, a window will pop up on the screen, and the information will be displayed there.

Print -- This prints a detailed summary of all of the ARs from the last **Query** into the file given in the **Print File:** field.

To generate and print a summary of a selected group of ARs, fill in the **Query List:** and **Sort List:** fields, select **Query** and wait for the query to complete, fill in the **Print File:** field, and select **Print**. The summary is rather wide -- it may be a good idea to use the LANDPRESS package to print it out sideways on a printer.

The AR query browser window:

This window shows a short summary (one line each) of the ARs that have been queried. Left-buttoning one of the AR lines will call AR.SHOW on that AR, to display it. Middle-buttoning an AR line will "**Get**" that AR into a specified AR edit form window.

Background menu commands:

When AREDIT is loaded, the item "AREdit" is added to the background menu, with a number of subitems. These are interpreted as follows:

AREdit -- Creates a new AR edit form, initially cleared.

New AR Form -- Same as AREDIT

Load AR Form -- prompts the user for a number, and creates a new AR edit form, with the specified AR number loaded initially.

AR.SHOW -- prompts the user for a number, and calls AR.SHOW, an old version of AREDIT which quickly displays the contents of a given AR. It prompts for a window region the first time it is used -- thereafter it uses the same window.

AR Query Form -- Creates a new AR query form.

Auxiliary AR edit form commands.

Pressing the left mouse button in the title bar of the AR Edit form command subwindow will bring up a menu of less-used commands:

Clear -- Clears ALL the fields of the current AR. Similar to **New**, except that none of the fields like **Source**:, etc., are filled in. This is useful when you are submitting an AR for someone else.

New

Get

Put -- the same as the **Commands** in the edit command subwindow.

Put&Get -- prompts the user for a number, **Puts** the current AR, and **Gets** the given numbered AR. Useful when scanning through a number of ARs.

GetFromFile -- Prompts the user for a file name, and loads the information from that file into the AR edit form subwindow. If this file is not in the right format, an error will be generated.

PutToFile -- Prompts the user for a file name, and stores the information from the AR into that file.

Locally caching the AR index.

All AR query operations use the information in the "AR Index" file. This file, which is updated every few days, is stored as {eris}<lispars>AR.INDEX. To speed up Query operations, this file can be copied to a local file server, or the local hard disk. Warning: this file is currently ~700 IFS pages long, and it will undoubtedly get larger. Also, it is the responsibility of the user to make sure that they update their local version of AR.INDEX when the master copy is updated.

To use a local version of AR.INDEX, give the file name as an argument to AR.QFORM.CREATE:

(AR.QFORM.CREATE '{DSK}AR.INDEX).

Global Variables that control AREDIT

The following are global variables that can be set to alter the operation of AREDIT. These are the only global variables that it is safe to change.

AR.ENTRY.LIST.WINDOW.FIELDS -- Controls which fields are displayed in the AR query browser window, along with the widths of the fields.

AR.ENTRY.LIST.PRINT.FIELDS -- Controls the fields displayed by the **Print** command of the AR query form.

AR.ENTRY.LIST.PRINT.MULTILINE.FLAG -- if non-NIL, the **Print** command of the AR query form will print all of the characters in each field, using multiple lines for those field values bigger than the field width allowed. If NIL, each AR will use only one line, truncating any field values that are too big.

DATEPATCH

By: Bill van Melle
(vanMelle.pa@Xerox.com)

DATEPATCH fixes some bugs and extends the functionality of the date functions DATE, GDATE and IDATE.

Date Parsing

The date parser (IDATE) now handles all dates legal in RFC822 syntax (except the silly single-digit military time zones). In addition, it handles months spelled out, months abbreviated with a period, and ignores initial strings of the form "{letter}*", assuming these to be specifying a day, as in "Monday, May 1, 1989". In addition to the official time zone specifications, it also recognizes any in the list TIME.ZONES, whose format has changed slightly:

TIME.ZONES [Variable]

An association list whose elements are of the form (*offset regzone dstzone*), where *offset* is the number of hours west of GMT (note that this, unfortunately, is opposite in sign to the RFC822 standard, but is strictly an internal matter), *regzone* is a string specifying the time zone normally, and *dstzone* is a string specifying the zone when daylight savings time is in effect. If *dstzone* is omitted, then there is no representation for that zone in daylight savings time, and DATE is forced to use absolute syntax (e.g., +0400).

The initial value of TIME.ZONES is

```
( ( 8 "PST" "PDT" )
  ( 7 "MST" "MDT" )
  ( 6 "CST" "CDT" )
  ( 5 "EST" "EDT" )
  ( 0 "GMT" "BST" )
  ( 0 "UT" )
  (-1 "MET" "MET DST" )
  (-2 "EET" "EET DST" ) )
```

IDATE also accepts an optional argument DEFAULTTIME, which is interpreted as a number of seconds past midnight. If the date string does not contain a time, DEFAULTTIME is used; if DEFAULTTIME is NIL, IDATE returns NIL in this case (as it always has).

Date Output

The date printers (DATE and GDATE) now produce appropriate time zones outside of the U.S. Given a choice of time zones, they take the first entry found in TIME.ZONES. In addition, they support a few more DATEFORMAT options:

MONTH.LONG [DateFormat Option]

Provides for full names of months rather than the first three characters. For instance, (DATE (DATEFORMAT MONTH.LONG SPACES NO.TIME)) might produce "20 February 87".

MONTH.LEADING

[DateFormat Option]

Causes the month to be produced as a word before the day and the day to be followed by a comma. For instance, (DATE (DATEFORMAT MONTH.LEADING MONTH.LONG YEAR.LONG NO.TIME)) might produce "February 20, 1987". MONTH.LEADING implies SPACES and inhibits NUMBER.OF.MONTH.

CIVILIAN.TIME

[DateFormat Option]

Specifies 12-hour time instead of 24-hour (military) time. For instance, (DATE (DATEFORMAT CIVILIAN.TIME NO.DATE NO.SECONDS)) might produce "11:34pm".

For completeness, listed below are all the DATEFORMAT options currently supported.

Those affecting the date portion:

NO.DATE	Omit the date portion (month, day, year, day of week).
NUMBER.OF.MONTH	Use a number for the month instead of spelling it.
MONTH.LONG	Spell the month out instead of abbreviating it.
MONTH.LEADING	Month before day, spelled out, comma after day.
YEAR.LONG	Use 4 digits for year instead of 2.
DAY.OF.WEEK	Include day of week (it appears at the end of the string, in parentheses).
DAY.SHORT	Use 3-letter abbreviation for day.
SLASHES	Separate parts of date with slashes instead of hyphens.
SPACES	Separate parts of date with spaces instead of hyphens.
NO.LEADING.SPACES	Omit leading spaces (default is a fixed format that always "lines up"). This also affects the hour when CIVILIAN.TIME is specified.

Those affecting the time portion:

NO.TIME	Omit the time portion (hour, minutes, seconds, zone).
NO.SECONDS	Omit seconds.
CIVILIAN.TIME	12-hour instead of 24-hour time.
TIME.ZONE	Include time zone specification.

FILEBANGER

Filed as {eris}<lispcore>internal>library>FileBanger.*
Not for customer release

Basic Functions

(DOFILEBANGER *Destination Length NoBreak*)

Starts a new file banger in its own process. The device and directory specified by *Destination* will be where the test files are created during the testing. Each file will be *Length* bytes long. The file banger performs consistency checks on all the files it creates after each operation. If those consistency checks fail, it will print a message indicating what it found wrong, and if *NoBreak* is NIL will open a break window.

To stop the file banger process, one may (KILL.PROCESS 'FILEBANGER), provided that only a single file banger is in operation.

(FILEBANGER *TestFile Destination MakeWindow NoBreak InParms OutParms*)

Enters a loop of creating, manipulating, and deleting files, making consistency checks after each pass. *TestFile* may be the name of a file (whose contents are used as the contents of the test files), or an integer, in which case a test file of that many random bytes is created. *Destination* is the device and directory to be tested (normally this should be {DSK}<LispFiles>). *MakeWindow*, if T, causes a fresh window to be created; that window is used for progress messages as the test runs. If *MakeWindow* is NIL, the regular TTY window for the process is used (and created, if need be!). If the test's consistency checks fail, FILEBANGER will print a message in the status window; if *NoBreak* is NIL, it'll also stop and open a break window. If you are interested in really stressing the file system, you may specify PARAMETERS arguments to be given to OPENSTREAM when files are opened. *InParms* is used when opening for input, and *OutParms* when opening for output. We don't recommend the use of these arguments.

Generally speaking, the test file should be of a good length (5000 bytes or more), to assure that the files spread out to cover the disk volume you're testing.

FILEBANGER continues to run until interrupted from the outside somehow. The easiest way to arrange that is to run it in its own process (using DOFILEBANGER) and use KILL.PROCESS.

Recommended Operation

(DOFILEBANGER '{DSK}<LispFiles> 5000 NIL)

This will open a TTY window for the FILEBANGER process, and print small ongoing status messages there; you will need to do something to prevent the scrolling from hanging things up when the TTY window is full (either type ahead some spaces in that window, or arrange for the window not to pause when it fills up). To stop the file banger when it has run long enough, do (KILL.PROCESS 'FILEBANGER) in the top-level TTY window.

Likely Error Messages

HARD DISK ERROR

The low-level disk code hit a hard error on the disk, and was able to complete its operation to the extent of recognizing the error. Indicates a bad spot on the disk.

VERIFY ERROR

The disk-page label checking code found an inconsistency. While this might be software, it is most likely to be controller trouble or a bad spot on the disk.

(infinite uninterruptible loop)

The low-level disk code hit a snag so severe that it could never complete the disk operation at all. This should never happen. If it does, suspect IOP trouble.

File1 and File2 differ at byte xxx

Two files should have been identical but weren't. Suspect software problems, unless the error follows a particular drive, processor, or IOP.

File1 has length xxx, but File2 has length yyy.

Two files should have been identical but weren't. Suspect software problems, unless the error follows a particular drive, processor, or IOP

Files in this directory /usr/local/lde/internal/
where copied from {eris}<lispcore>internal>library>
31-Jan-90

LISP DIAGNOSTICS

Date : 05 - 21 - 1985

This document describes Lisp Diagnostics program for Xerox 1108 and Xerox 1109 workstations.

To start it, load LISPDIAGNOSTICS.DCOM from the Lisp Library.

As soon as it finishes loading the file, a menu will appear.
You will have 2 choices, i.e. : Start Exercise and Stop Exercise.

If the file has previously been loaded, you'll only have to type :
(MAKEDIAGNOSTICSMENU) to make the menu appear.
When you select Start Exercise, the following message will appear on the
Top level typescript window :

Legend

! -> Completed !DIAGNOSE of MACROTEST
@ -> Completed TANSPEED benchmark
-> Completed BROWSE benchmark
□ -> Found a Clearing House on the Ethernet
- -> Looked, but failed to find a Clearing House
[xxx] -> Tried 32 retrievals from CH and got xxx failures
{xxx} -> Copied and deleted xxx copies of the Disk file
(xxx) -> Finished with xxx'th run of the EMUPROC loop
GDATE on new line marks release of working set pages

In addition to that, a window appears on top of the menu, it describes the
current activity indicated by a black rectangular cursor.

They are :

1. Swap out working set.
2. Ethernet activity.
3. Disk Activity.
4. Benchmark.

To stop the diagnostics, simply click the Stop Exercise selection.

(This package was superficially tested on Intermezzo>Full.sysout on
05 - 21 - 1985 by G. Santosa)

NSMAIL

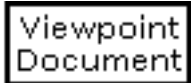
INTRODUCTION

The module NSMAIL implements the protocols to allow Lafite to be used to send and retrieve Xerox NS Mail. Load the file NSMAIL.LCOM. To run this in Lyric, you must have loaded the LispUsers module NSRANDOM as well (q.v. for important loading information). If you don't have NSRANDOM loaded, you can't use the "Put to file" command described below.

If you have both Grapevine and NSMAIL implementations loaded, you must set Lafite's mode to NS. Use the "NS Mode" subitem underneath Lafite's Quit command, or call (LAFITEMODE 'NS). You must also be a registered NS user, and have a mailbox.

ATTACHMENTS

The main difference between this and earlier versions of NSMAIL is that "attachments" are no longer left in your mailbox to be read later with, for example, Viewpoint. Instead, Lafite retrieves the entire attachment and encapsulates it into an image object that is enclosed as part of the text message, immediately following the header. A typical attachment appears in a mail message as:

Attachment: 

If you click inside the object with any mouse button, you are offered a menu of things you can do with the attachment. The choices vary according to the type of attachment:

View as text	This brings up a window in which is displayed the raw content of the attachment as ascii bytes. Runs of non-ascii bytes are replaced by nulls to reduce the amount of garbage. Some attachments are utter gibberish, but some, such as Viewpoint documents and Interpress masters, contain sections that are plain text. With this command, you may be able to decide whether you care to do anything further with the attachment. (Sorry, there is no Viewpoint to TEdit converter, nor are there plans for one.)
Put to file	This prompts you for a file name, and creates a file to contain the attachment. The file must be on an NS file server for this command to be very useful; otherwise, information will be lost. Once the file is so stored, you can retrieve it from Viewpoint and manipulate it just as if you had originally retrieved it as mail in Viewpoint.

Send to Printer	This command is only available for attachments that are in the form of an Interpress master. The command prompts you for a printer (must be one that accepts Interpress, of course), and sends the attachment to it for printing.
Expand folder	This command is only available for attachments that are in the form of a "folder". A folder is a mechanism for collecting several objects into a single one. The Expand folder command splits the attachment up into its component objects, each of which can be manipulated in the same way as a top-level attachment. For example, if the folder contains an Interpress master, you can print it.

If you use the Put to file command on a folder, the name component of the file name you type will be treated as the name of a new subdirectory, and the components of the folder will appear as files in that subdirectory. For other types of attachments, Put to file (usually) produces an ordinary (non-directory) file.

Messages containing attachments are otherwise just like formatted messages—you can move them to other folders, and you can forward them (assuming the mail is received by another Lafite recipient and did not have to pass through a mail gateway).

There is currently no mechanism for creating your own attachments to send to other users.

MISCELLANY

If you prefer the old behavior of leaving the attachments behind in the mailbox, set the variable NSMAIL.LEAVE.ATTACHMENTS to T, but this use is discouraged. You must take care to regularly retrieve your mail from somewhere (such as Viewpoint) that will flush out all the mail; otherwise, the mail with attachments, whether you want them or not, accumulate on the server.

When in NS mode, Lafite will want your NS login identity. Normally, if your NS password differs from your default password, you will be prompted to login. You can also call (LOGIN 'INS::) yourself to set your NS login.

You can freely intermix Grapevine and NS mail in the same mail folder if you like, but the Answer command always treats the selected message as if it were one in the current mode. So if you try to answer a Grapevine message while in NS mode, some confusion may result. Also, the status window always shows you mail status only of the current mode.

SOURCELOOKUP

INTRODUCTION

This module provides a mechanism to locate the source file and floppy in which a particular system function is defined. It is designed for use with the set of Xerox LISP source files distributed on floppy with the Lyric release. This module uses the WHEREIS module distributed with Lisp Library documented in Section 23 of the *Interlisp Reference Manual*.

LOADING THE MODULE

SOURCELOOKUP requires WHEREIS.DCOM and HASH.DCOM to be loaded. The variable WHEREIS.HASH must be set to the hash file LYRICSOURCES.WHEREIS. All these files are distributed on the floppy labeled Lyric Sources #10. The hash file must reside on a random access device. To use the hash file on FLOPPY do the following:

```
(SETQ WHEREIS.HASH '({FLOPPY}LYRICSOURCES.WHEREIS))
```

The variable LyricSourceIndex will be set to a list where each element is a list whose CAR is the name of a Lyric Source floppy and CDR is the list of files contained on the floppy.

USER FUNCTION

```
(LOCATE.FUNCTION function)
```

If *function* is defined in a distributed source file prints out the appropriate file name and floppy and returns the file name; else prints "*function* not found" and returns NIL.

EXAMPLES

```
(LOCATE.FUNCTION 'SETQ)
```

The function **SETQ** is defined in the file **LLINTERP** located on floppy **Lyric Source #5**.

```
(LLINTERP)
```

```
(LOCATE.FUNCTION 'UNDEFINED)
```

UNDEFINED not found.

NIL

SETUP instructions for
PC Emulation virtual hard disk and real PC floppy disk access

1. Make sure all of the normal PCE .DCOM files are loaded to run PCE:
(PCEWINDOW, PCE, PCEERD, PCEDISPLAY, PCEKEYBOARD).
2. Select PC Emulation on the background MENU. This may take a while to load the PC fonts.
3. Put MSDOS disk in floppy drive. Select BOOT in the command window.
4. If the PCE boots ok, then select QUIT in the the command window.
5. Now we will make a virtual hard disk for the PCE.

From LISP do the following:

- ```
1_(PCE.CREATE.ERDFILE 'PCE.DISK 50)
 (* this will make a lisp file PCE.DISK which is nearly 2
 megabytes big - make sure you have enough disk space)
```
6. Reboot the PCE (BOOT command) and make sure configuration has the right file name PCE.DISK for the ERDFILE using the CONFIG command.
  7. From the PC window:  
  
-FDISK  
 (\* create DOS partition using all of the disk )  
  
-FORMAT C:/V/S  
 (\* format the rigid disk, put a syste on it, and label)  
  
-COPY A:\*. \* C:  
 (\* copies all of the system files from floppy to ERD)
  8. Select QUIT from the COMMAND window.
  9. You may now boot from the ERD (emulated rigid disk) by selecting FIXED DISK as boot device and Set-Config from the configuration window.
  10. From LISP exec:  
  
2\_ LOAD(VPCDISK.DCOM) (\* loads in LISP virtual PC disk/floppy and real PC floppy access)  
3\_ (VPCDISK.CREATE.DEVICE 'PCDISK 'PCE.DISK T) (\* associates ERDFILE PCE.DISK with a device name PCDISK)  
4\_ DIR {PCDISK}PCDISK:\* (\* directory listing of the PC DISK)  
5\_(SETQ PC.TEXTFILE.EXTENSIONS '(TXT PAS BAS 1 2 3] (\* list of PC file extensions which are treated as TEXT type files)  
6\_(VPCDISK.CREATE.DEVICE 'PCFLOPPY '{PCFLOPPY}) (\* associates real PC floppy driver with {PCDISK} device PCFLOPPY)  
 (\* make sure real floppy is in the disk drive when you execute this)  
7\_DIR {PCDISK}PCFLOPPY:\* (\* directory listing of the real floppy)
  11. At this point you are set up to do any LISP commands you want such as Filebrowser, Hardcopy, TEdit, etc. using the {PCDISK}PCDISK: for the virtual hard disk, and {PCDISK}PCFLOPPY: for real PC floppies.
  12. Try using Filebrowser and then selecting a textfile that has one of the extension names you setup in PC.TEXTFILE.EXTENSIONS above, and selecting HARDCOPY.
  13. Try using TEDIT on a floppy file such as {PCDISK}PCFLOPPY:README.1 and then editing the file and writing (PUT) it back. and then reselect it from filebrowser and selecting hardcopy. (Make sure floppy is not write-protected.)
  14. You may also access files on sub-directories. IE, {PCDISK}PCDISK:<DIR>FILE - However, you cannot delete or create directories from LISP. You have to do this from the PC window.

## MASTERSCOPE EXTENSIONS

1. More than one type of executable thing.

All possible executable thing types are stored on MSFNTPES, which is a list of MSANALYZABLE records:

(filepkgname setname getdef-fn)

Vanilla Masterscope starts with it set to

((FNS FNS GETDEF))

and new things get added to it through MSADDANALYZE (see below).

The filepkgname must be the File Manager name of the new executable thing. MSADDANALYZE will barf if filepkgname is not on IL:FILEPKGTYPEES.

The setname must be the name Masterscope uses for a set of the new executable things.

The getdef-fn has to take the same arguments as GETDEF and return NIL or a definition suitable for Masterscope munching.

MSGETDEF, MSEDITF, MSSEEKTYPE, MSMARKCHANGED, and MSADDANALYZE look at MSFNTPES

Subject: New Lispcore>Library package: MesaTypes  
To: Lispcore^, Lispsupport, Sheil, Cooper, Purcell

Announcing a new Lispcore>Library package: MesaTypes

By Tayloe Stansbury with help from Richard Burton.

This package introduces three new clisprecordtypes which allow you to describe any block of bits with an arbitrarily nested datatype. You can define multidimensional, nonstandardly indexed arrays with multi-word elements; records with multi-word fields; and multi-word types. Special accessfns cover up the necessary \BLTs and LOCFs. Appropriate create methods are automatically provided. The package also provides a number of macros for manipulating instances of such types.

Anyone who wants to use graceful datatypes to describe some arbitrary chunk of memory (e.g. ethernet, rs232, nsfiling, any file system) will probably find this package useful; most of the 1108 file system now depends on it. People translating Mesa system code into Lisp will find it particularly useful.

This message is stored as Lispcore>Library>MesaTypes.tedit. Proper external documentation of this package will follow release of the 1108 file system. Extensive examples of its use can be found in the first few pages of stansbury>newdlionfs>dlionfs.

-- Tayloe.

## Common Lisp package code and symbol conversions

Ron Fischer

[This document is for developers only. It is not intended to be part of any external software release.]

The package code can behave mighty strangely if not operated "just so." Please read this document carefully.

### Loading

The package code is now loaded in the full.sysout. It lives in {Eris}<LispCore>Library> on

CMLSYMBOL.DCOM

CMLPACKAGE.DCOM

### Initializing

Packages, as you might expect, must be bootstrapped carefully. At its simplest, initialization is accomplished by calling just one function:

```
(package-init t)
```

#### simple startup

```
(package-init &optional (convert? nil))
```

[Function]

Clear, make structures of, initialize & convert symbols (if convert? is t) to, and enable use of the symbol package system. On a DandTiger this takes about 25 minutes.

The bootstrap actually takes place in three steps. The first one creates all the package structure needed. The second passes over all the atoms in the sysout to "tag" them with packages. The third enables the package world.

#### step 1: make system packages

```
(package-make)
```

[Function]

Create, but do not fill with symbols, the base packages that need to exist. Also enables the package qualifier characters in the readtables and saves the old definitions of \READ.SYMBOL and \MKATOM.

#### step 2: convert existing symbols

```
(package-hierarchy-init &optional (convert? nil))
```

[Function]

Fill all the initial system packages with their proper symbols, moving litatoms into appropriate places and such. If convert? is non-nil then symbols whose pnames have fake package qualifiers, like cl:length, will be converted IN PLACE to remove the qualifier. If conversion takes place you cannot fully disable the package system.

#### step 3: enable package system, remove old hacks

The third part of the bootstrap enables the use of packages by the reader, printer and friends. This involves redefining \MKATOM and \READ.SYMBOL. Also, if \*package\* is set to nil things will not go well. Please use package-enable and package-disable to start and stop, as these fellows know what they're doing. These faithful functions are available on a menu by calling package-menu.

```
(package-enable &optional (package *interlisp-package*))
```

[Function]

Turn on the package system, making package the current one and redefining \READ.SYMBOL and \MKATOM appropriately.

```
(package-disable)
```

[Function]

Turn off the package system and restore the old definitions of \READ.SYMBOL and \MKATOM. After disabling, symbols interned under the package system will not be eq to symbols of the same name reread.

```
(package-menu)
```

[Function]

Make a menu that allows turning the package system on or off without using the reader.

```
(package-hacks-disable)
```

[Function]

Eliminates package simulation hacks when loading over an old sysout. These hacks cannot be re-enabled.

(pkgconvert-enable) [Function]

Enables a change in the behavior of the function `\\READ.SYMBOL` (using the function `check-symbol-namestring`), which converts symbols being read based on their "looking like" a package prefixed symbol name. "Looking like" is defined by a table in the global `litatom-package-conversion-table`.

litatom-package-conversion-table [Variable]

a global variable containing a list of clauses used by the functions `package-hierarchy-init` and `check-symbol-namestring` to determine if a symbol "looks like" is trying to be package qualified. The list contains clauses with the following structure:

*(prefix-string list-of-exception-strings package-name-string where)*

*prefix-string* - string containing the prefix of symbols which this clause converts, eg "CL:"

*list-of-exception-strings* - list of strings naming symbols which this clause should leave alone, eg ("CL:FLG")

*package-name-string* - string containing the name of the package that symbols converted by this clause should wind up in, eg "LISP"

*where* - either :INTERNAL or :EXTERNAL, indicates whether symbols converted by this clause should be external or internal in their package.

The initial value of this variable is:

```
(
 ("CL:"
 ("CL:FLG" "CL:MAKE-SYMBOL" "CL:COPY-SYMBOL" "CL:INTERN"
 "CL:MAKE-KEYWORD" "CL:GENTEMP" "CL:KEYWORDP")
 "LISP" :external)
 (":" nil "KEYWORD" :external)
)
```

## Notes & cautions

The read functions may return strings if you use package qualifier syntax and `*package*` does not contain a package. This is part of debugging code that Bill put into them. Beware especially of making `*package*` NIL without calling `package-disable`.

The list of conversion clauses is searched linearly, hence longer prefixes should come before shorter ones with the same chars in them, ie put a clause for "CL: " before one for "CL:".

## Missing features

Cannot be placed early into loadup due to dependancies on CL files.

Apropos (and other Interlisp litatom functions) are not redefined to operate with packages.

## Performance

There have been some small improvements since these timings were taken.

```
FILEDATES : (("21-Jul-86 14:12:17" .
{ERIS}<LISPCORE>CML>LAB>CMLPACKAGES.;129))
```

Testing symbol / litatom creation. Old array package. There are 6 random symbols made (new ones). intern is a factor of 16 slower, conses heavily, makes 3 strings for each call.

```
85#(TIMEALL (TIME.MKATOM))
Elapsed Time = .336 seconds
SWAP time = .32 seconds
CPU Time = .016 seconds
PAGEFAULTS = 8
LISTP
9
```

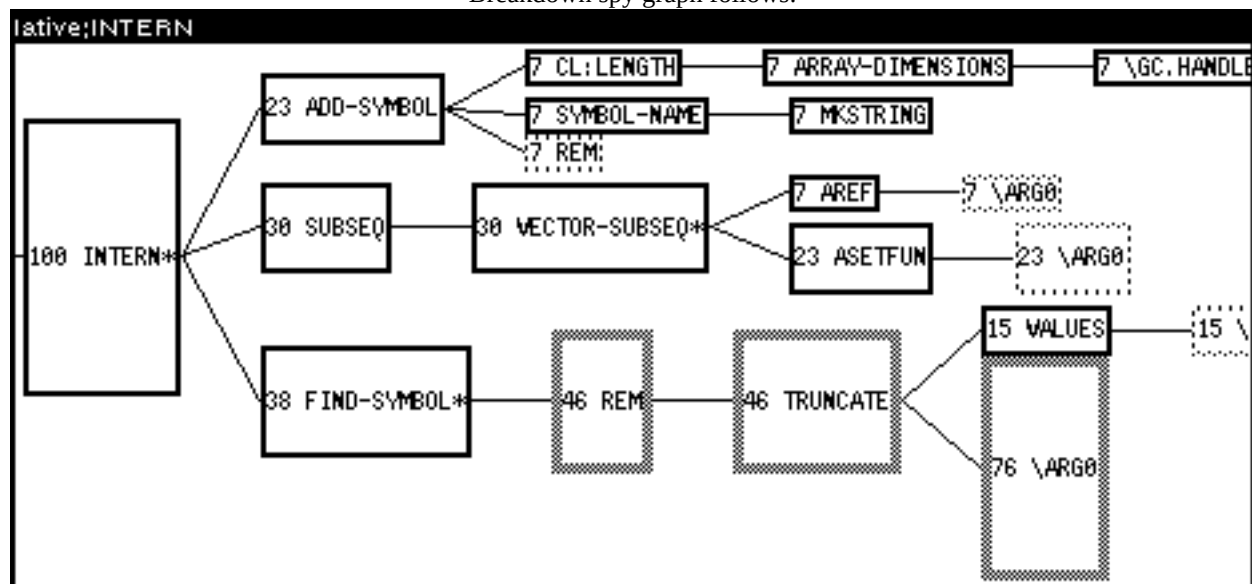
```
86#(TIMEALL (TIME.INTERN))
Elapsed Time = .449 seconds
SWAP time = .188 seconds
```

```

CPU Time = .261 seconds
PAGEFAULTS = 5
LISTP STRINGP
423 18

```

Breakdown spy graph follows:



---

---

**Peano**

---

---

Maintained By: Frank Shih (Shih.envos@Xerox.com)

This document last edited on 8-Nov-88

**INTRODUCTION**

Peano is an ancient graphics demo.

**OPERATION**

(PEANODEMO *LEVEL SCALE*)

[Function]

Runs a demo in PEANOWINDOW (set first time), drawing Peano curves to level *LEVEL* with lines *SCALE* units wide.

## Read and Print state profile

There are a large number of special variables that control reading and printing. Taken together these comprise a "mode of operation" for the reader and printer. Because there are a fair number of such variables a method for capturing and restoring their state has been provided.

```
(xcl:make-read-print-profile &key (readtable *readtable*) (read-base *read-base*) (read-suppress *read-suppress*)
(package *package*) (read-default-float-format *read-default-float-format*) (print-escape *print-escape*) (print-
pretty *print-pretty*) (print-circle *print-circle*) (print-base *print-base*) (print-radix *print-radix*) (print-case
print-case) (print-gensym *print-gensym*) (print-level *print-level*) (print-length *print-length*)) [Function
]
```

Creates a read-print-profile object. The default values of its components are taken from the current special bindings of the variables shown in the argument list above.

```
(xcl:copy-read-print-profile profile) [Function]
```

Creates a new read-print-profile object and copies the values in the slots of profile into the new one.

```
(xcl:read-print-profile-p object) [Function]
```

Returns true if the object is a read-print-profile, otherwise false.

```
(xcl:read-print-profile-readtable profile) [Function]
(xcl:read-print-profile-read-base profile) [Function]
(xcl:read-print-profile-read-suppress profile) [Function]
(xcl:read-print-profile-package profile) [Function]
(xcl:read-print-profile-read-default-float-format profile) [Function]
(xcl:read-print-profile-print-escape profile) [Function]
(xcl:read-print-profile-print-pretty profile) [Function]
(xcl:read-print-profile-print-circle profile) [Function]
(xcl:read-print-profile-print-base profile) [Function]
(xcl:read-print-profile-print-radix profile) [Function]
(xcl:read-print-profile-print-case profile) [Function]
(xcl:read-print-profile-print-gensym profile) [Function]
(xcl:read-print-profile-print-level profile) [Function]
(xcl:read-print-profile-print-length profile) [Function]
(xcl:read-print-profile-print-array profile) [Function]
(xcl:read-print-profile-print-structure profile) [Function]
```

profile must be a read-print-profile object. Returns the named slot of the read-print-profile. A corresponding setf method is provided for each slot.

```
(xcl:save-read-print-profile profile) [Function]
```

Capture bindings of special read & print variables into the profile. Returns profile.

```
(xcl:with-read-print-profile profile-form &body forms) [Macro]
```

Binds all the special read & print variables to the values in the profile and executes the body forms as in a let. Returns the value of the last of the forms.

```
(xcl:restore-read-print-profile profile) [Function]
```

Restore values of special read & print bindings from profile. Sets current bindings. Returns T.

```
xcl:*default-read-print-profile* [Variable]
```

Holds a simple default read-print-profile. When possible programs should default to the current settings of the read-print variables by capturing them with save-read-print-profile instead.

```
(xcl:find-read-print-profile name) [Function]
```

Since read-print-profiles enclose readtables and packages, which are available by name, named read-print-profiles



are also available.

This function will retrieve one of the following "standard" read-print-profiles:

LISP: LISP readtable and USER package, others nominal.

XCL: XCL readtable and XCL-USER package, ditto above.

INTERLISP: INTERLISP readtable and INTERLISP package, ditto.

This function is case insensitive. It returns NIL if a profile by that name is not found.

(xcl:list-all-read-print-profile-names) [Function]

Returns a list of strings containing the names of all available read-print-profiles.

## WHEREIS

The WHEREIS package extends the function WHEREIS such that, when asked about a given name as a function, WHEREIS will consult not only the commands of files that have been noticed by the file packages, but also one or more hash file data bases that associate function names with file names.

(WHEREIS NAME TYPE FILES FN) [Function]

Behaves exactly like the standard WHEREIS function (see the *Interlisp-D Reference Manual*) unless *TYPE*=FNS (or NIL) and *FILES*=T. In this case, WHEREIS will consult, in addition to the files on FILELST, the hash files that are on the value of WHEREIS.HASH. Note: normally the user will just enter the full name of a file onto WHEREIS.HASH, but as WHEREIS begins to use it, it will convert the entry into a cons of the name and the hash file handle.

Many system functions, such as the editors, call WHEREIS with *FILES*=T, so loading this package automatically makes any information contained in the WHEREIS data base files available throughout the system.

---

## Creating a Where Is Data Base

---

Information may be added to an existing WHEREIS hash file, or by creating new data bases. It is often useful to have a separate data base for large user systems, explicitly calling the following function:

(WHEREISNOTICE FILEGROUP NEWFLG  
DATABASEFILE) [Function]

Inserts the information about all of the functions on the files in *FILEGROUP* into the WHEREIS data base contained on *DATABASEFILE*. If *DATABASEFILE* is NIL, the first entry on WHEREIS.HASH is used.

*FILEGROUP* may be simply a list of files, in which case each file thereon is handled directly; but it may also be a pattern to be given as a file group argument to DIRECTORY, so \* may be used.

If *NEWFLG* is non-NIL, a new version of *DATABASEFILE* will be created containing the data base for the functions specified in *FILEGROUP*. If *NEWFLG* is a number, the hash file will be created with *NEWFLG* entries. Otherwise, it will be created to allow 20,000 entries.

Example:

The following sequence of actions will cause all of the files on the PROJECT directory to be noticed by the WHEREIS package.

```
(WHEREISNOTICE ' <PROJECT>* . T ' <PROJECT>PROJECTWHEREIS.HASH)
(push WHEREIS.HASH
 (FINDFILE ' <PROJECT>PROJECTWHEREIS.HASH))
```

One of the key components of CLOS in inheritance. The CLOS Browser provides functionality for displaying this structure and for extending it. It also provides functions for displaying and changing the class definitions and method definitions which make up a system written in CLOS.

## Creating a Browser

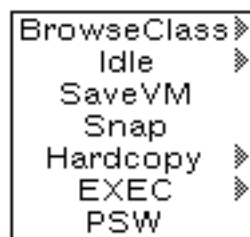
---

A browser can be created in two ways:

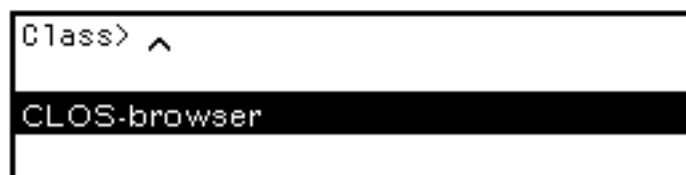
- Via a menu option from the Background Menu
- By calling the function CLOS-BROWSER:BROWSE-CLASS on a class

### Creating a browser via the Background Menu

When the CLOS-BROWSER module is loaded, an entry is added to the Background Menu, as shown below:



Selecting the menu item BrowseClass brings up a window, with a prompt for the name of the class to use as the root of the browser as shown below.



Type in the name of the class you wish to browse at the flashing cursor, and the class graph will be drawn in the window.

## Creating a browser programmatically

Browsers can also be created by calling the function BROWSE-CLASS:

**(BROWSE-CLASS &OPTIONAL CLASS-NAME-OR-LIST &KEY :WINDOW-OR-TITLE :GOOD-CLASSES :POSITION)** **[Function]**

This function brings up a browser on the class named or the list of classes named. If a window is supplied for the :WINDOW-OR-TITLE argument, then the browser is created in that window, else an appropriately sized window is created. The window is positioned at the :POSITION argument or, if not supplied, then the position is set via the mouse. If a text string is supplied for the :WINDOW-OR-TITLE argument, then that string is used for the window title, else the string "CLOS-browse" is used. If :GOOD-CLASSES is supplied, then only those classes in the list are displayed.

## Using the Class browser

---

Instances of CLOS-BROWSER are operated on through a mouse-based interface.

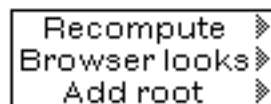
Buttoning on the browser will cause one of the following menus to be popped up:

- One menu appears when the left or middle button is pressed while the mouse is in the title bar. This menu has operations that apply to the browser itself.
- The other menu appears when the middle button is pressed when the mouse is on one of the nodes in the browser.

If the left button is pressed when the mouse is on a node, that node is boxed. This marks the node for some operations.

### Options in the title bar menu

The following menu appears when you left- or middle-button in the title bar.



### Recompute and it's suboptions

Selecting the Recompute option and dragging the mouse to the right causes the following submenu to appear:



Most of these items change the appearance of the browser, not the contents.

|                   |                                                                                                                                                                     |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Recompute         | Recomputes the browser from the starting objects. It does not recompute the labels for each node if those labels are cached in the Label-Cache slot of the browser. |
| Recompute Labels  | Recompute the browser from the starting objects, including the labels.                                                                                              |
| Recompute inPlace | Recompute the browser without affecting the scrolled location of the lattice within the window.                                                                     |
| Clear caches      | Clear the caches of the nodes.                                                                                                                                      |

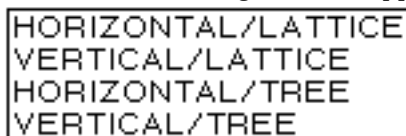
### Browser looks and it's suboptions

Selecting the Browser looks menu item and sliding to the right causes the following submenu to appear:



Selecting one of these options changes the looks of the browser.

|                  |                                                                                                                                              |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| Shape to hold    | Make the window for the browser just large enough to contain the browser.                                                                    |
| Change font size | Causes a menu of alternative font sizes to pop up. Selecting one of these causes the browser to be redrawn with the nodes at that font size. |
| Change format    | Causes the following menu to appear:                                                                                                         |



Horizontal/Lattice Lays out the grapher as an horizontal lattice.

Vertical/Lattice    Lays out the grapher as a vertical lattice.

Horizontal/Tree    Lays out the grapher as a horizontal tree.

Vertical/Tree    lays out the grapher as a vertical tree.

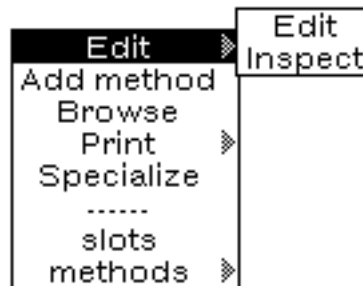
## Options in the Middle-button menu

The following menu appears when you middle-button over a node in the graph:



## Edit and it's suboptions

Selecting Edit causes an editor on the class definition to be brought up. Sliding the mouse to the right causes the following menu to appear:



Edit      Edits the class named by the node

Inspect      Inspects the class object named by the node.

## Add Method

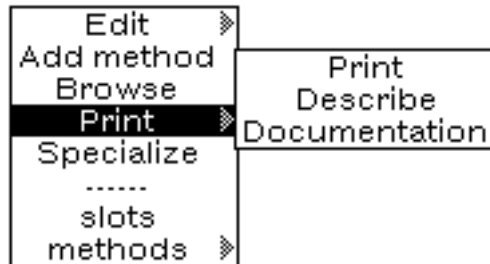
Selecting the Add Method option brings up an editor window with a template for a method to be added to that class. When the editor is done the method is installed for that class and the menu updated.

Browse

Selecting the Browse option causes a browser to be created starting with that class as the root.

## Print and it's suboptions

Selecting Print prints out the class definition. Sliding the mouse to the right causes the following menu to appear:



|               |                                                                                                                                           |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| Print         | Print's the class definition                                                                                                              |
| Describe      | Describes the class, listing it's metaclass, it's supers classes, it's subclasses, it's CPL, and the number of methods specialized to it. |
| Documentation | Print's the documentation string for the class                                                                                            |

## Specialize

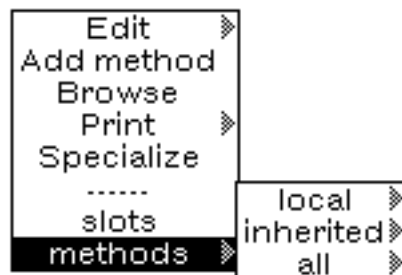
Selecting the Specialize option brings up an editor window with a template for a subclass to be added to that class. When the editor is done the class is installed and the browser updated.

## Slots

Selecting the Slots option is the same as selecting the Edit option, it brings up an editor on the class definition.

## Methods

The Methods option allows you to edit one of the methods defined for that class. Selecting it and sliding to the right brings up the following sub-menu:



|       |                                                                                  |
|-------|----------------------------------------------------------------------------------|
| Local | Bring up a menu of the local methods, ie methods directly defined for this class |
|-------|----------------------------------------------------------------------------------|

|           |                                                                                      |
|-----------|--------------------------------------------------------------------------------------|
| Inherited | Bring up a menu of the methods this class inherits from it's superclasses.           |
| All       | Bring up a menu of all the methods defined for this class, both local and inherited. |

Selecting an item with the left button from the resulting menu brings up an editor on that method. If there are multiple methods that apply, a gray triangle appears in the right edge of the menu next to that item. Sliding to the right brings up a menu of method specializers to select the appropriate method.



---

---

**COLOROBJ**

---

---

By: Greg Nuyens (Nuyens.pa@Xerox.com)

uses: COLOR

COLOROBJ provides an Interlisp-D imageobject which allows the color to be set on an imagestream. The default image looks like this:



Unknown IMAGEOBJ  
GETFN: COLOROBJ.GETFN

The color of the object can be changed by middle-clicking on the object.

(COLOROBJ.CREATE Color)

[Function]

Creates a single colorobj. Color is optional and defaults to COLOROBJ.DEFAULT.COLOR.

COLOROBJ.DEFAULT.COLOR

[Variable]

Determines the initial color of a colorobj. Default is RED.

---

---

**HostUp**

---

---

By: Johannes A. G. M. Koomen  
(Koomen.wbst@Xerox or Koomen@CS.Rochester.edu)

This document last edited on December 27, 1988

**SUMMARY**

This module provides the function HOSTUP? which attempts to find out if a given host is currently available. Contrast this to the system function HOSTNAMEP, which returns T if it has *at any time* successfully opened a connection to given host, whether or not it is currently available.

**DETAILS**

(HOSTUP? hostname) [Function]

Returns T if and only if the given host is currently responding. No distinction is made between dead and non-existing hosts.

HOSTUP.TIMEOUT [Global variable]

The function HOSTUP? returns NIL if no response is received from the given host within HOSTUP.TIMEOUT milliseconds. Default value is 15,000.

HOSTUP.RETRYCNT [Global variable]

This variable indicates the number of times the function HOSTUP? sends requests to the given host. Each time through the loop the function waits longer by a geometrically increasing amount of time, such that the total time does not exceed HOSTUP.TIMEOUT. Default value is 5. Hence, in the default case, a call to HOSTUP? with a dead host ends up sending a request to the host 5 times, waiting for an answer about 500, 1000, 2000, 4000 and 8000 milliseconds, respectively.

---

---

## IRIS

---

---

By: Nuyens (Nuyens.pa@Xerox.Com)

Requires: COLOR and COLOROBJ

This document last edited on January 31, 1987.

### INTRODUCTION

The LispUsers module IRIS is a collection of functions for using an IRIS (Integrated Raster Imaging System) with Interlisp-D. The IRIS is a sophisticated 3-d color graphics workstation produced by Silicon Graphics Incorporated. Much of its imaging system is implemented in special purpose VLSI, making it a very powerful graphics engine.

IRIS provides three separate ways to exploit the IRIS in Interlisp-D. The first is to use the IRIS as an Interlisp-D imagestream. In this capacity the IRIS serves as the output device for a program which needn't even know that it is outputting to the IRIS. Thus standard system utilities (such as GRAPHER, SKETCH, and TEDIT) can make use of the IRIS without change. The images produced by these system utilities can then be manipulated with the special abilities of the IRIS (3-d rotation, color updates, double buffering, object definitions, etc.)

The second mode of use for the IRIS is to directly access the IRIS graphics library. In this mode, Interlisp-D provides stubs for the functions in the IRIS graphics library. Thus, Interlisp-D provides the illusion of having the IRIS as a direct device like the standard screen. However, it is actually a separate computer accessible across the 10-MB Ethernet.

The third mode is to use the view controller to interactively view scenes created with either of the two modes. The view controller allows the user to create an object once (interactively or via program) but then view the object many times on the IRIS without recreating the object. This object downloading allows real-time rotation of the object displayed on the IRIS.

This package also implements boot service for the IRIS workstation. The D-machine supplies the boot program to the IRIS over the Ethernet, eliminating the need for a floppy-disk on the IRIS.

This package benefited substantially from comments and improvements by Stu Card, Michel Desmerais and Lennart Lovstrand.

### FILE DESCRIPTION

The implementation includes the following files:

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LOADIRIS    | after loading this file, the user will be prompted to place the IRIS icon. Buttoning this icon will produce a menu including the choice "create loadup panel". Choosing this item will produce a panel which can load the IRIS files. Button the "Standard" entry in the loadup panel, then button on the small SGI logo next to "Load". Prior to being able to run any applications, the IRIS must be running the terminal program. This can be booted from the floppy on the IRIS or from the Lisp workstation. To use the workstation as a bootserver, the boot file for the IRIS terminal program (see msg below) must be copied to {DSK}irisbootfile (or a directory in the list IRISBOOTDIRECTORIES) on the Dandelion that is to be the bootserver. At that point, hit the reset button on the IRIS and type 'n' to the monitor prompt on the iris. That will initiate a net boot from the Dandelion. |
| IRISSTREAM  | The file which contains the Interlisp-D imagestream definition for the IRIS. Thus, the standard Interlisp-D graphics facilities can be applied to an open IRIS imagestream. After this is loaded, images which are processed through the standard hardcopy menu can be sent to the IRIS. "Iris" will appear as a printer when the "to a printer" choice is made from the roll-off menu entry "Hardcopy".                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| IRISLIB     | this file contains the stubs for the functions in the IRIS library. For instance, corresponding to the function circf (found in the IRIS documentation), there is a function IRIS.CIRCF . The arguments are as listed in the IRIS documentation with the addition of the argument SPPstream. This is the SPP connection opened to the IRIS. (If omitted, it defaults to the value of IRISCONN, which is set by OPEN.IRISCONN). Where an argument is a matrix, it is passed to the lisp function as a list of the rows, each row itself a list. When a library fn returns a matrix, the appropriate (usually floatp) matrix is passed in and the elements are set.                                                                                                                                                                                                                                           |
| IOLIB       | this file contains the communication primitives which are used by the fns on IRISLIB.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| IRISNET     | this file contains the network support for the iris. The IRIS must be on the same network as the D machine (the IRIS doesn't handle routing through the gateway properly). If the IRIS is being bootservered from the Lisp workstation, the boot server software will set IRISNSHOSTNUMBER automatically. (The diagnostic messages can be inhibited by setting \IRIS.VERBOSE to NIL. They are left in because they can be useful when initially getting the IRIS to boot).                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| IRISVIEW    | This file contains the functions supporting the interactive viewing menu, together with the object definition facility it uses.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| IRISDEMOFNS | This file contains an example 3-d function for the IRIS. It is a function called TETRA which draws a colored, 3-d recursive tetrahedron.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| IRIS.TEDIT  | This file.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

## GETTING STARTED

**N.B.** All variables and functions in this document are written as in the old Interlisp readtable. To type in the examples in this document, bring up an "Old-Interlisp" Exec. (available from the background menu by rolling off "exec" and then "Interlisp".)

-1) type (FILESLOAD LOADIRIS)

It will ask you to position the following (SGI) icon:



(it can be recreated later by typing (IRIS.CREATE.ICON) at the Interlisp executive).

**0) Next type (FILESLOAD IRISSTREAM IRISVIEW)**

### **1) Boot the IRIS**

If you have a floppy drive for your IRIS, use it to boot the IRIS GL2 (XNS) Terminal program. ( If your IRIS is a workstation, not a terminal, you need to run the WSIRIS program available from SGI).

If you do not etherboot, you must set the variable IRISNSHOSTNUMBER. (see variables section below.)

If you have no floppy, you are going to "etherboot" the IRIS. To begin ensure that IRISBOOTDIRECTORIES is set properly. (see Magic Variables section below).

Press a mouse button in the SGI icon. Choose "Enable bootserver". Now boot the IRIS (by pressing the black button labelled "reset" on the IRIS display). To the prompt on the IRIS, type (on the iris keyboard) N (followed by carriage return). To the eventual prompt "connect to which host?" type < (carriage return). Etherbooting will automatically set the variable IRISNSHOSTNUMBER. Status messages will be printed to the promptwindow. The IRIS will beep when it is finished booting.

### **2) Open an IRIS stream**

Button in the IRIS stream and choose "Create IRISView panel". Confirm the command by clicking left button of the mouse. "Connected" should print on the IRIS screen, and then the screen should turn black and display axes. (**N.B.** If at any time the screen goes completely blank, touch any key on the IRIS keyboard. The IRIS has a dubious screen-saver "feature".)

## **Modes**

The modes of use of the IRIS are as follows:

### **Interlisp-D display stream**

In this mode, the IRIS is acting like a window on the Lisp machine. To see updates as they occur toggle the "Double Buffer" button on the IRISView Controller. It will respond "Single buffering". Then press the "2D-Home" button on the IRIS. To demonstrate this, try the use of commands like

```
(PROGN (DSPCOLOR 'BLUE \IRISSTREAM)
```

```
 (PRINTOUT \IRISSTREAM T "Let's here it for ")
```

```
(DSPCOLOR 'RED \IRISSTREAM)
```

```
(PRINTOUT \IRISSTREAM "COLOR." T)
```

```
(DSPCOLOR 'BLUE \IRISSTREAM))
```

NOTE that the first time a font family is defined it will download the font after loading the spline font definitions. This will take a few minutes.

```
(DRAWLINE 0 0 400 200 2 NIL \IRISSTREAM 'RED)
```

```
(FORCEOUTPUT IRISCONN)
```

Since the SPP stream buffers, sometimes it will be necessary to (FORCEOUTPUT IRISCONN). (There is a function simply called F that will do this for you.)

These simple instructions indicate the use of the IRIS as an Interlisp Device Independent Graphics (DIG) stream. See the IRM for further examples common to all DIG devices. For interactive graphics, the following sections contain other uses of the IRIS.

## Output stream for Sketch and TEdit

TEdit and Sketch can make use of the above display stream facilities. To see an example color document, do the following. (If you don't want to see the sketch in the document, you may omit loading sketch and sketchcolor, since they are slow to load.)

```
(FILESLOAD COLOROBJ SKETCH SKETCHCOLOR)
```

```
(TEDIT 'IRIS-EG.TEDIT)
```

### To print a TEdit (or sketch) to the IRIS

First time: Press "2D-Home". In the title bar of the window, use the right button to bring up the default window menu. Pick the roll-off item Hardcopy. Rolloff into "hardcopy to printer", then choose "Iris". Choose "yes" to the menu saying "Make this the default printer".

Later times: The IRIS will now be the default printer, so just press "2D-Home" and choose Hardcopy from the default window.

### To use Sketch

From the background menu, choose "SKetch". Build a closed polygon (with the  in the sketch menu). Choose "change" in the sketch menu. Pick a control point of the polygon you just built. Pick "filling color" from the menu that appears. From the menu that appears, pick a filling color.

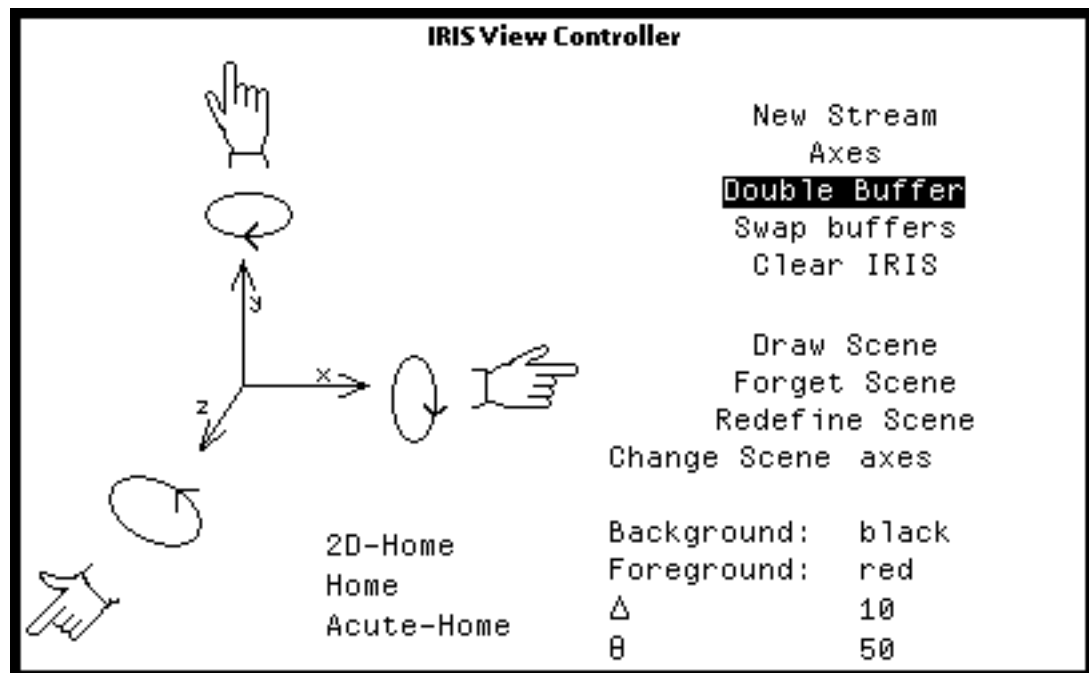
### Clearing the IRIS

In the IRIS icon is an entry "clear IRIS". It will return the IRIS to the original state with the white screen, etc.

### Defining new colors

At any time the color menu is presented, "new color" may be selected. RGB will provide color sliders and CNS will provide an english description of colors. A window will pop up asking for a name for the color. Type in a name, and that name will appear in the color menu from then on.

## Using the IRISVIEW controller



**N.B.** When an IRISview controller is initially opened, the IRIS will be double buffering. This means that any drawing on the iris occurs on the back of two buffers. "Swap Buffers" must be chosen to bring the back buffer to the front. This includes "Clear IRIS" and other drawing commands. Double buffering can be turned off with the "Double Buffer" button, but moving images are much smoother with it turned on.

This window allows you to define objects on the IRIS and control your view of them.

### Changing the scene:

First, select "change scene". You will be prompted with the predefined scenes as examples. Choose "axes". After a moment (during which "defining object for axes" is printed), an XYZ axes picture will appear on the screen. Try choosing each of the 3 home buttons to see the effect.

Press the "2d Home" button. (This actually means to move the view such that the IRIS simulates an 1108 screen in size and view). Now change the scene to "SKULL". Again after a delay, a Grateful Dead logo will be displayed.

### Defining a scene:

We are going to define a scene that calls a function on the file IRISDEMOFNS so type

(FILESLOAD IRISDEMOFNS)

Button "Change Scene". Choose "New Scene" from the Menu. In response to "Scene Name", type "TETRA". In response to "Form to eval" click in the window the message is printed in and then type

(TETRA) [carriage return]

When the object is displayed, press each of the "Home" buttons to see different home positions. (2D-Home is perhaps best).

### Viewing the scene:

Before viewing the scene, choose the "Double Buffer" option. Now hold the left button down on the item marked "Background:" in the view controller. Choose the background color for the object. White is often a good choice. (Note that sliding off the menu at this point will display "none" as the background. This means that each time the scene is drawn will be on top of the previous result. This can give very pleasing, flashy results. Experiment a bit. It's great fun (and you are probably being paid for it!))

### Adjusting the view:

Bug the part of the free menu marked  This will move the image in the positive x axis. Similarly with the other hand icons. Holding down the shift key will reverse the direction in the same axis. Bugging the  entry in the view controller will let you change the "delta" for translation.



Now bug the part of the free menu marked . This will rotate the image in that axis. Holding down the shift key will reverse the direction of rotation.

Bugging the  entry will let you choose a new "theta" for rotation (in degrees).

### **Home:**

Bugging "home" will return the scene to the original view. It is often a good idea to choose "Home" before changing scenes.

### **Axes:**

Bugging the "axes" entry in the menu will superimpose a set of three dimensional axes on the view for reference purposes. It defines the axes of rotation controlled by the view controller and the directions of translation. (The axes look best against a black background, I think)

### **Forgetting a Scene:**

Reclaims the space inside the IRIS and removes scene names from the scene menu. All but one scene may be deleted.

### **Defining a scene containing a sketch (or Tedit):**

Button "Change Scene". Choose "New Scene" from the Menu. In response to "Scene Name", type "IRIS-EG". In response to "Form to eval", click in the window the message is printed in and then type a carriage return.

The view controller will prompt with "Make object then type RETURN". Now, hardcopy the sketch exactly as above. (periodically the view controller will flash the screen to remind you that you are still making the object). When you are finished, type carriage return as before. The current scene name will be printed in the view controller.

## **Troubleshooting:**

### **"Iris Terminall SPP not responding" printing in prompt window**

This may occasionally print when the Dlion is too busy to service the SPP connection to the IRIS. If it repeats, however, this means that a previous IRIS stream has been lost. Bring up a process status window by choosing PSW in the background, then choose the process (Iris Terminal SPP#2 for instance) and bug "kill" in the PSW. Be sure to kill the correctly numbered process. "Iris Terminal SPP" is not the same as "Iris Terminal SPP#3".

### **Magic Variables**

IRISNSHOSTNUMBER

[Variable]

contains the 48 bit etherhostnumber of the IRIS. If you are etherbooting, this will be set automatically. Otherwise, set it to a string like "0#4000.12000.41504#0" as described on page 31.9 of the IRM.

\IRIS.DEBUG

[Variable]

Defaults to NIL. If T, when fonts are created, only the first lowercase letters will be defined. This is much faster than loading the whole font.

IRISBOOTDIRECTORIES

[Variable]

For users still running with the R1B version of the IRIS terminal program, the file concerned is called xiris. This is the file that should be copied to {core}irisbootfile. However, for users running R1C (also referred to as GL2), it is necessary to obtain an updated version of the terminal program from SGI. (It was named simply "iris" on the tape we received.) This variable must contain the list of directories where the boot files are to be found. (For instance, very fast booting may be obtained by copying the boot files to the {core} device and putting {core} at the front of IRIS.BOOTDIRECTORIES.)

IRIS.VERBOSE

[Variable]

Defaults to T. Says whether or not status messages are printed during font creation etc.

\IRISSTREAM

[Variable]

contains the IRIS stream that is current.

IRISCONN

[Variable]

contains the SPP connection that is current.

---

---

**KOTOLOGO**

---

---

By: Masinter (Masinter.PA@Xerox.COM)

Uses: none

This document last edited on August 17, 1988.

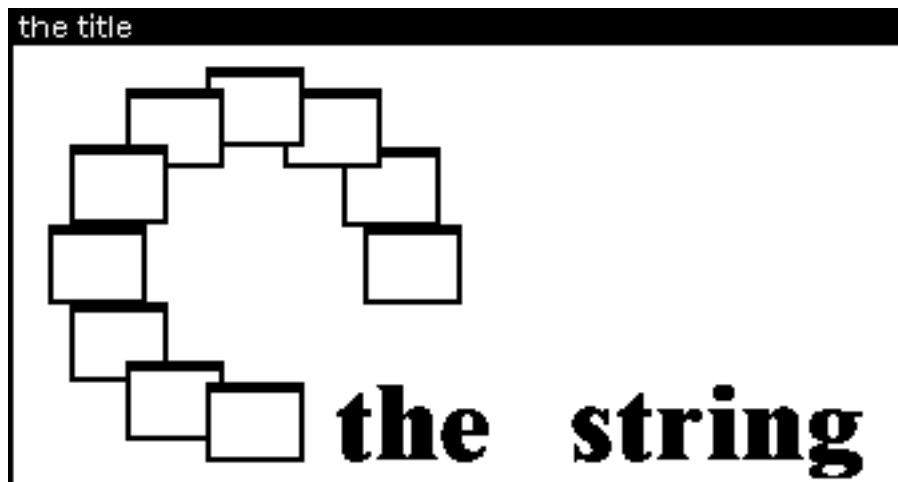
**INTRODUCTION**

Makes a Koto-style logo window.

(KOTOLOGOW *string where title angledelta*)

[Function]

Works like LOGOW did in Koto. Put *string* as the main logo name, with *title* in the window title. *angledelta* is the angle at which the little windows go. *where* is either a position or an old window. For example (KOTOLOGOW "the string" NIL "the title" 30) produces:



---

---

**LispNerd**

---

---

By: Maxwell (Maxwell.pa)

DICTCLIENT, DINFO

INTERNAL

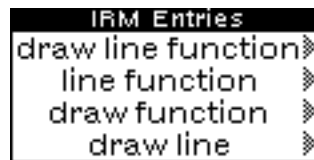
## INTRODUCTION

The LispNerd provides a menu-based interface to the Interlisp Reference Manual. The data for the LispNerd is stored on the Dictionary Server, and is accessed via DICTCLIENT. Because of certain licensing agreements we have with Houghton-Mifflin, the Dictionary Server should only be used by people within PARC. Hence LispNerd has not been released as a LispUsers package.

## HOW TO USE THE LISPNERD

When you load the LISPNERD, it adds a new menu item called "Search IRM" to the background menu. Bugging "Search IRM" causes the LispNerd to prompt the user for keywords in the prompt window. It then produces a menu of items that have at least two of the keywords in their definitions (perhaps with submenus, if there is more than one class of items). Bugging one of these items will cause the LispNerd to fetch the documentation for that item using DINfo.

For example, if you type the input "draw line function", you will get the following menu:



where each of the entries has a sub-menu of the items that have the keywords listed in the entry.

If you type the input "date", you will get a menu all of the items in the Interlisp Reference Manual that have the word "date" in them. Putting parenthesis around items means that the items should all be treated as one keyword for the purposes of sorting them into groups. For instance, the input "(draw draws)(line lines) function" will look for all of the items that have "draw" OR "draws" in their definition.

LispNerd only fetches 50 entries at a time, so sometimes you will see an entry in the menu that says something like ". . . + 103 more". Clicking this item will cause the LispNerd to fetch the next 50 entries. Also, when there is more than one class of entries, sometimes a sub-menu will only list the number of

entries it has, and not the entries themselves. To get the entries, click the menu item with the keywords in it, and LispNerd will recompute the menu with just those keywords.

Please send all bug reports to [Maxwell.pa](mailto:Maxwell.pa).

---

---

**MATHTONS**

---

---

By: Tad Hogg (Hogg.pa@Xerox.com)

**INTRODUCTION**

This file defines the translation array needed to convert from the Press MATH font to corresponding NS characters. This allows documents containing the MATH font to be printed on Interpress printers.

The array \MATHTONSARRAY contains the translations for most of the characters. Some may not be available on particular printers, causing them to appear as black boxes.

---

---

**MICROTEK**

---

---

By: Ron Clarke (RClarke.pa@Xerox.COM)

This document last edited October 31, 1988

**INTRODUCTION**

MICROTEK is an image processing software package that enables you to operate Microtek Models-300 and 300A Intelligent Image Scanners with the Xerox 1108 and 1186 workstations. The Microtek MS-300, 300A, and MSF-300C are high-resolution optical scanners that can convert text, artwork, photographs, etc, into digital form for processing by computer. The digitized images that are output to the computer contain up to 300 black and white dots for every linear inch of the original document. Page size can be as large as 8.5 by 14 inches. Sophisticated firmware in this scanner enables the user to set the scanning area and control brightness, contrast, scaling, shading and other characteristics of the scanned images through simple commands transmitted from the 1108 or 1186. Two basic scanning modes are supported: Line Art mode for accurate capture of completely black-and-white material, and Halftone mode for faithful reproduction of material with varied shading. Mixed-mode scanning is also available.

With the MICROTEK software package you will be able to: Set the scanner to capture images of all kinds, with desired visual effects, and transmit them to the 1108/1186, save scanned images to disk, floppy or file server for later reloading to recreate images and print scanned images to a Xerox 4045 or 8044 laser printer.

**SOFTWARE REQUIRED**

MICROTEK.DFASL

MICROTEKPRINT.DFASL (if you have a Xerox 4045 or 8044 laser printer)

DLRS232C.LCOM

EDITBITMAP.LCOM

READNUMBER.LCOM

4045XLPSTREAM.DFASL (if you have a Xerox 4045 laser printer)

**FONTS USED**

MODERN 10, 12 BOLD

Other useful software for manipulating the scanned image:

Lispuser's Packages:

ACTIVEREGIONS, ACTIVEREGIONS2, AIREGIONS, FILLREGION

**HARDWARE REQUIRED**

Xerox 1108 with RS232C port (E-30 upgrade kit). It is also recommended that the 1108 have 3.5 meg of memory and a floating point processor (CPE board) to enable faster scanning and creation of bitmaps.

Xerox 1186. It is also recommended that the 1186 have 3.7 meg of memory.

Microtek MS-300, MS-300A, or MSF-300C Intelligent Image Scanner with optional serial port.

### CABLE CONFIGURATION

Note that the cable configuration is DIFFERENT for the MSF-300C scanner. Plugging a standard RS232C cable into the MSF-300C DB25 connector may result in damage to the equipment.

#### RS232C Port (DTE)      MICROTEK MS-300, MS-300A - DB25 Connector

| Signal  | Pin | Pin | Signal  |
|---------|-----|-----|---------|
| FGround | 1   | 1   | FGround |
| TD      | 2   | 3   | RD      |
| RD      | 3   | 2   | TD      |
| Sground | 7   | 7   | Sground |

Pins 5, 6, 8 and 20 are jumpered together on the RS232C port end of the cable.

#### RS232C Port (DTE)      MICROTEK MSF-300C- DB25 Connector

| Signal | Pin | Pin | Signal |
|--------|-----|-----|--------|
| TD     | 21  | 3   | RD     |
| RD     | 9   | 2   | TD     |
| Ground | 5,7 | 7   | Ground |

### DOCUMENTATION REQUIRED

Microtek MS-300, MS-300A, or MSF-300C Intelligent Image Scanner Operation Manual

### LOADING MICROTEK

Make sure that DIRECTORIES contains the directory where the required software is located. When the file MICROTEK.DFASL is loaded, the item "MicrotekScanner" will be added to the Background menu. If you have a Xerox 4045 or 8044 laser printer load MICROTEKPRINT.DFASL. If you have a Xerox 4045 laser printer load 4045XLPSTREAM.DFASL. Your 4045 laser printer should be connected to the TTY/DCE port.

### RUNNING MICROTEK

The process of running the Microtek scanner software consists of three phases: Scanner initialization, scanning, and creating a bitmap of the scanned image that can eventually be printed. Each of these are controlled by different menus within the Microtek Scanner Control Window.

### SCANNER INITIALIZATION

Set the Microtek scanner so that it is operating at 19200 baud by setting its internal DIP switches (See Microtek Operating Manual for details). Turn on the Microtek scanner. Select "MicrotekScanner" from the background menu and the Microtek Scanner Control window (figure 1) and Microtek Scanner Pagemap window (figure 2) will be created. (Note you may have to do a control-E and retry if cursor flashes while trying to create the control window). The scanner pagemap window is used to select the area of the image to be scanned and to select the page length. The scanner control window is used to set all other scanner parameters, start and stop scanning as well as to initiate creation and printing of scanned image bitmaps. After these windows have been created, the RS232 port will be initialized to



19200 baud and an attention command will be sent to the scanner. If all cables are connected properly and the scanner is on, the message "MICROSCAN 300(A) V# is ready" will be displayed in the Microtek Status Window. If the cable is configured incorrectly or the scanner is not on or ready the message "Microtek Not Responding ... Check scanner and cable" will appear instead.

**Microtek Status Window**  
! MICROSCAN 300A V1.3 READY

**Microtek Command Menu**

**Output Filename!:** {DSK}<LISPPFILES>IMAGE

**Microtek Configuration Menu**

**Reduction!** 0% = 300 DPI **Grain Size Levels**  
**Gray Level!** 0 = 8X8 33

**Contrast:**  **Brightness:**

**Background!** HALFTONE **Window Mode:** LINEART **Page Length:** 4

**Frame!** X1: 0.0 Y1: 0.0 X2: 5.0 Y2: 3.0

|                  |     |     |     |       |     |      |     |     |     |     |
|------------------|-----|-----|-----|-------|-----|------|-----|-----|-----|-----|
| <b>Window 1!</b> | ON? | YES | X1: | 2.125 | Y1: | 1.25 | X2: | 4.5 | Y2: | 2.5 |
| <b>Window 2!</b> | ON? | NO  | X1: | 0.0   | Y1: | 0.0  | X2: | 0.0 | Y2: | 0.0 |
| <b>Window 3!</b> | ON? | NO  | X1: | 0.0   | Y1: | 0.0  | X2: | 0.0 | Y2: | 0.0 |
| <b>Window 4!</b> | ON? | NO  | X1: | 0.0   | Y1: | 0.0  | X2: | 0.0 | Y2: | 0.0 |

**Microtek Display Menu**

**Bitmap Name!:** IMAGE **Shrinkfactor!** 1 **Rotation!** NONE

**Source Filename!:** {DSK}<LISPPFILES>IMAGE

**Microtek Print Menu**  
 **Printer!:** 8044 **XPOS!:** 0 **YPOS!:** 0 **SCALE!:** 1:1

FIGURE 1 - MICROTEK SCANNER CONTROL WINDOW

Before scanning can be initiated, a number of parameters have to be set by the user via the Microtek Command Menu and Microtek Configuration Menu as follows:

**Microtek Command Menu:**

**Output FileName** Left buttoning on this item allows you enter the name of the file on disk, floppy or filesaver where the scanned data is to be saved. Be sure to type a carriage return to terminate this entry.

**Microtek Configuration Menu:**

**Reduction** Left button on the number next to the item Reduction and a menu will appear. Reduction can be changed from 0%, which corresponds to scanning at 300 dots per inch (DPI) to 75%, which corresponds to 75 DPI.

**GrayLevel** Left button on the number next to the item GrayLevel and a menu will appear allowing you to choose from a selection of gray levels based on grain size and number of gray levels within that grain size.

**Contrast** Left button on either the the left or right arrow to either decrease or increase the contrast setting.

**Brightness** Left button on either the the left or right arrow to either decrease or increase the brightness setting.

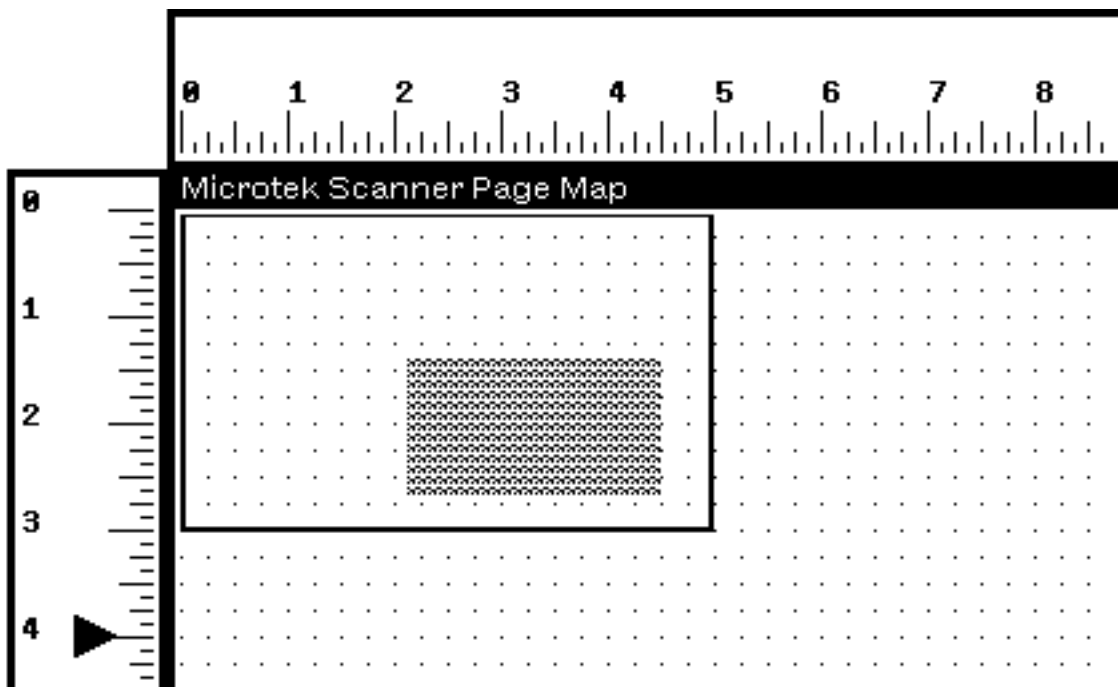


**FIGURE 2 - MICROTEK SCANNER PAGEMAP WINDOW**

**BackGround** Select either HALFTONE or LINEART as the primary scanning mode for the image. Line Art mode is for accurate capture of completely black-and-white material, and Halftone mode for faithful reproduction of material with varied shading.

**Pagelength** Move the cursor to the vertical ruler of the page map ( figure 2 ). The cursor will change to a right pointing triangle. Position this triangle and left-button to select the pagelength. The page length will also show up in the configuration menu. The page length should be set so that it is longer

than the actual page length of the document to be scanned. Otherwise you will get a paper jam message at the completion of scanning. The minimum page length is 3 inches and the maximum page length is 14 inches.

**Frame** The scanning frame is an area within the document that will be scanned. The maximum scanning frame is 8.5" by 14". Left button on the item Frame and you will be prompted to sweep out an area on the scanner page map to select the primary area to be scanned. The horizontal and vertical rulers and the page map grid dots can be used as a guide in determining the dimensions of the scanning frame. When the the scanning frame has been swept out, a box of the scanned area will be drawn on the page map and the actual X and Y coordinates of the top lefthand corner and lower righthand corner will appear next to corresponding items on the configuration menu (See Figure 2).

**Windows 1- 4** Windows are areas within the scanning frame that are scanned in a different mode from the rest of the frame. If LINEART mode is selected as the background, all material in any windows you set will be scanned in Halftone mode, and vice versa.

The method used to set the windows is similar to that used to set the scanning frame except that you first need to specify whether the window is to be selected or not. This is done by left buttoning on the YES/NO indicator next to each window. A menu will pop-up and will allow you select "yes" or "no". After making your selection, left buttoning on the appropriate Window # will cause you to be prompted to sweep out an area within the scanning frame. Each selected window will be displayed and have a unique shade to it (See Figure 2). The only restriction is that the scanning mode must not change more than twice in one 8.5" horizontal scan line. Thus, if two windows lie across the same scan line they must extend to the edges of the page setting area. (Note that material to the left and right of the frame is scanned but not transmitted to the 1108.) You can select different windows for halftone vs lineart mode by switching between backgrounds. The item above WINDOW1 indicates which window mode is selected. An illegal window setting will result in an error message when you attempt to scan. Also note that the windows will be displayed on the scanner pagemap only if there is a "yes" next to the window number.

## SCANNING

After the Microtek scanning parameters have been initialized, the document to be scanned should be placed in the scanner top-first with the image to be scanned facing away from the user. Scanning is initiated by left-buttoning SCAN on the Microtek Command Menu. The software first creates a scratch file in {CORE} for storage of the incoming data. It then sends the scanning parameters to the scanner and if all are valid the scanning process starts as indicated by movement of the rollers. You have up to 5 minutes to insert a document before the scanner automatically stops. After scanning has been completed you will be notified in the status window that it is saving the core file to the file specified in Output Filename. It takes approximately 20 minutes to scan an 8.5" x 11" document at 300 DPI.

You may stop the scanning at any time by selecting STOP. The document will be ejected and the scanner reset. You can also explicitly reset the scanner by selecting RESET. This closes the scanner scratch file if it is open, sends a reset command to the scanner and then sends the attention command. If everything is reset properly, you will get the message "MICROSCAN 300(A) V# is ready" in the status window.

## CREATING BITMAPS OF SCANNED IMAGES

The Microtek Display Menu is used to create bitmaps from a file that contains scanned data. Select SOURCE FILENAME and enter the name of the file that contains the scanned data. Select BITMAP NAME and enter the name of a variable that you would like the bitmap bound to. **Be sure to type a carriage return to terminate the entry of each of these items.** Left button on the number next to

SHRINKFACTOR and choose a factor by which you want the bitmap shrunk. The default value is 1. Left button on the item next to ROTATION and choose how you want the scanned image to be rotated. The default is "none." After these items have been set, you can then select CREATE BITMAP to start the bitmap creation process. The status window will be updated as it proceeds to create the bitmap and finally, you will be prompted to sweep out a scrollable window to display the bitmap. NOTE: Depending on the size of the bitmap, rotation may take a "very" long time and will look like your machine has frozen...be patient, it will come back. If you desire to save the bitmap(s) on a file you can do the following:

```
(SETQ filenameCOMS '((VARS bitmapname1 bitmapname2 etc))).
(MAKEFILE '{device}<directory>filename)
```

### PRINTING BITMAPS OF SCANNED IMAGES TO A XEROX LASER PRINTER

If you have the package MICOTEKPRINT loaded you will have a MicrotekPrint Menu under your display menu (See Figure 1). Select BITMAPNAME on the display menu and enter the name of the bitmap that you would like to print. To select where on the page the bitmap is printed, left button XPOS and YPOS and enter a number. For the 4045 laser printer the values of XPOS can be between 0 - 2550 and YPOS, between 0 - 3300. 1" = 300 print units o 4045 . For an 8044 Interpress laser printer the values of XPOS can be between 0 - 21590 and YPOS, between 0 - 27940. 1" = 2540 Interpress units. The scale that an image is printed at is dependent upon its initial scanned reduction/DPI. You can increase or decrease the scale at which the bitmap is printed by buttoning on the number next to the item SCALE and selecting a scaling factor. On an 8044 Interpress printer a scale of 8:1 will magnify an image by 8 times on printing , 1:1 will print at true size and 1:8 reduce the image by 8 times. Values in between are also available. On a 4045 laser printer only a limited number of scale factor are availble and is dependent upon the original scan reduction as shown in the table below.

| REDUCTION (%) | RESOLUTION (DPI) | SCALES ALLOWED |
|---------------|------------------|----------------|
| 0             | 300              | 1:1, 2:1, 4:1  |
| 5             | 285              | 1:1, 2:1, 4:1  |
| 10            | 270              | 1:1, 2:1, 4:1  |
| 15            | 255              | 1:1, 2:1, 4:1  |
| 20            | 240              | 1:1, 2:1, 4:1  |
| 25            | 225              | 1:1, 2:1, 4:1  |
| 33            | 200              | 1:1, 2:1, 4:1  |
| 35            | 195              | 1:1, 2:1, 4:1  |

| REDUCTION (%) | RESOLUTION (DPI) | SCALES ALLOWED |
|---------------|------------------|----------------|
| 40            | 180              | 2:1, 1:1, 1:2  |
| 45            | 165              | 2:1, 1:1, 1:2  |
| 50            | 150              | 2:1, 1:1, 1:2  |
| 55            | 135              | 2:1, 1:1, 1:2  |
| 60            | 120              | 2:1, 1:1, 1:2  |
| 67            | 100              | 1:1, 1:2, 1:4  |
| 70            | 90               | 1:1, 1:2, 1:4  |
| 75            | 75               | 1:1, 1:2, 1:4  |

Select **PRINT** to initiate the printing process. **NOTE:** The amount of reduction that you will be able to do is dependent upon the number bits that were originally scanned in. If you make the scale too small nothing will be printed out.

#### OTHER ITEMS AND GENERAL COMMENTS

On the Microtek Command Menu, left buttoning the item **PAGEMAP** will alternately open and close the scanner pagemap window. Left buttoning on the item **QUIT** will close the input and output streams to the scanner, shutdown the RS232C port and close the scanner pagemap and control windows. The following icon will be displayed if you shrink the Microtek Scanner Control window.



The Microtek Pagemap window will close when you shrink the Microtek Scanner Control window and has to be explicitly opened when the Microtek Scanner Control window is expanded again. This is done by buttoning on **PAGEMAP** in the Microtek Command Menu window.

Within Interlisp you normally cannot create bitmaps larger than approximately 2.1 million pixels (about 1400 x 1400). The Microtek scanner software allows you to create bitmaps much larger than this but at the cost of using a correspondingly large amount of virtual memory. If you are near your maximum vmemsize, as determined by comparing (VMEMSIZE) to (VOLUMESIZE 'volumename), there is a good chance you could crash your system if you create a very large bitmap...caveat emptor. In addition you will not be able to call the function **EDITBM** to edit bitmaps larger than 2.1 million pixels.

The reduction % used to scan the original image is stored on the property list of the atom that the bitmap is bound to. It is saved as the property "Resolution" and is in %. This is used to determine the appropriate values that will make an image 1:1 when printed. If you attempt to print a bitmap to an Interpress printer that was not created by use of the Microtek scanner software you will be prompted to enter a scale explicitly. The following table indicates the 8044 laser printer scale used for scanned images and can be used as a guide when attempting to print bitmaps not created by the Microtek software.

| <u>REDUCTION (%)</u> | <u>RESOLUTION (DPI)</u> | <u>SCALE</u> |
|----------------------|-------------------------|--------------|
| 0                    | 300                     | .240         |
| 5                    | 285                     | .252         |
| 10                   | 270                     | .266         |
| 15                   | 255                     | .282         |
| 20                   | 240                     | .300         |
| 25                   | 225                     | .320         |
| 33                   | 200                     | .360         |
| 35                   | 195                     | .369         |
| 40                   | 180                     | .400         |
| 45                   | 165                     | .439         |
| 50                   | 150                     | .480         |
| 55                   | 135                     | .533         |
| 60                   | 120                     | .600         |
| 67                   | 100                     | .720         |
| 70                   | 90                      | .800         |
| 75                   | 75                      | .960         |

Further information about the Microtek scanner can be obtained from:

Microtek Lab Inc  
 16901 South Western Avenue  
 Gardena, California 90247  
 Tel: 213-321-2121, 800-654-4160

---

---

## NSCOPYFILE

---

---

By: Bill van Melle (vanMelle@Xerox.com)

The module NSCOPYFILE modifies COPYFILE so that if both the source and destination files are on NS file servers, the copying is done by an NSFiling-specific copy routine. This routine copies *all* attributes of the source, including non-standard ones, such as those used by Viewpoint. Thus, you can safely copy Viewpoint files from inside Lisp without losing information. In addition, if the copy is from an NS file server to itself, the copy is performed by the server itself, which is considerably faster than shipping the file over the Ethernet.

In addition, you can also copy entire directories, by specifying directory names as the source and destination, e.g.,

```
(COPYFILE "{FS:}<Carstairs>Lisp" "{FS:}<Calvin>Lisp>Current")
```

The destination directory must not already exist, since this operation creates an entirely new directory, whose contents are a copy of all the source directory's offspring, to all levels. If the destination directory happens to exist but has no children, it is considered vestigial and is quietly deleted first (Lisp usually suppresses such directories when performing a directory enumeration).

You can also use RENAMEFILE in the same manner to either rename a directory, or to move an entire directory and its descendents to a new node in the file server's hierarchy, or to a new server altogether. You must, of course, have access rights to delete the source directory *and* all its children, and the destination must be on an NS file server.

A word about protection: when a file is copied or moved, the new file is given "defaulted" access rights, i.e., its protection is set as specified by its new parent (sub)directory, just as if you had created the file afresh by any other means. Thus, if the original file happened to have its own explicit protection, that protection is ignored. When copying or moving an entire directory, only the top-level directory receives default protection, so if any individual descendent file had non-default protection, that protection is copied verbatim. This can lead to confusion—you may want to use the NSPROTECTION module to change the new directory's descendents to default protection. See the documentation of NSPROTECTION for more discussion about protection issues. Note that if a file/directory is renamed within the same parent directory, the operation is considered merely "changing the name", and its protection is left unchanged.

Note: If you are using the FILEWATCH module, be aware that files being copied between NS servers do not appear (because the files are not opened by the normal Lisp file system).

---

---

**POSTSCRIPT**

---

---

By: Matt Heffron (BEC.HEFFRON@ECLA.USC.EDU)

**INTRODUCTION**

The PostScript package defines a set of imageops for printers which understand the PostScript page description language by Adobe. At Beckman we have successfully used TEdit, Sketch, LISTFILES, and HARDCOPYW to an Apple LaserWriter and an AST TurboLaser PS. The PostScript imagestream driver installs itself when it is loaded. All symbols in the PostScript driver are located in the INTERLISP: package.

**VARIABLES**

POSTSCRIPT.FONT.ALIST [InitVariable]

POSTSCRIPT.FONT.ALIST is an ALIST mapping Xerox Lisp font names into the root names of PostScript font files. It is also used for font family coercions. The default value should be acceptable for any of the fonts which are built into the Apple Laserwriter.

POSTSCRIPTFONTDIRECTORIES [InitVariable]

POSTSCRIPTFONTDIRECTORIES is the list of directories where the PostScript .PSCFONT font files can be found. The default value is: ("{DSK}<LISPFILES>FONTS>PSC>").

\POSTSCRIPT.SHORTEDGE.SHIFT [InitVariable]

\POSTSCRIPT.SHORTEDGE.SHIFT is the distance (in points) to shift the image perpendicular to the short edge of the paper. A positive value gives a shift upward in portrait mode, and to the right in landscape mode. The default value is: 0.

\POSTSCRIPT.LONGEDGE.SHIFT [InitVariable]

\POSTSCRIPT.LONGEDGE.SHIFT is the corresponding variable for shifts perpendicular to the long edge of the paper. A positive value here gives a shift to the right in portrait mode and downward in landscape mode. The default value is: 0.

\POSTSCRIPT.SHORTEDGE.PTS [InitVariable]

\POSTSCRIPT.SHORTEDGE.PTS indicates the printable region of the page (in points) along the short edge of the paper. It should be adjusted to allow for any shifts of the image (see above). The default value is: 576 (= 8 inches).

\POSTSCRIPT.LONGEDGE.PTS [InitVariable]



\POSTSCRIPT.LONGEDGE.PTS indicates the printable region of the page (in points) along the long edge of the paper. It should be adjusted to allow for any shifts of the image (see above). The default value is: 786.24 (= 10.92 inches).

#### HINT

The AST TurboLaser PS has an imageable area on the page which is a different size than that of the Apple LaserWriter. The values of \POSTSCRIPT.SHORTEEDGE.PTS and \POSTSCRIPT.LONGEDGE.PTS for the AST are 575.76 and 767.76, respectively.

\POSTSCRIPT.MAX.WILD.FONTSIZE [InitVariable]

\POSTSCRIPT.MAX.WILD.FONTSIZE indicates the maximum point size that should be returned from FONTSAVAILABLE when the SIZE argument is wild (i.e. \*). All integer point sizes from 1 to \POSTSCRIPT.MAX.WILD.FONTSIZE will be indicated as available. The default value is: 72.

POSTSCRIPT.PREFER.LANDSCAPE [InitVariable]

POSTSCRIPT.PREFER.LANDSCAPE indicates if the OPENIMAGESTREAM method should default the orientation of output files to LANDSCAPE. It can have one of three values: NIL, T, or ASK. NIL means prefer portrait orientation output, T means prefer landscape, and ASK says to bring up a menu to ask the preferred orientation if it wasn't explicitly indicated in the OPENIMAGESTREAM call (with the ROTATION option). The default value is: NIL.

POSTSCRIPT.TEXTFILE.LANDSCAPE [InitVariable]

POSTSCRIPT.TEXTFILE.LANDSCAPE indicates if the printing of TEXT files (e.g. LISTFILES, ...) should force the orientation of output files to LANDSCAPE. The default value is: NIL.

POSTSCRIPT.BITMAP.SCALE [InitVariable]

POSTSCRIPT.BITMAP.SCALE specifies an independent scale factor for display of bitmap images (e.g. window hardcopies). Values less than 1 will reduce the image size. (I.e. a value of 0.5 will give a half size bitmap image.) The position of the scaled bitmap will still have the SAME lower-left corner (i.e. the scaled bitmap is not centered in the region of the full size bitmap image). The default value is: 1.

#### HINT

Setting POSTSCRIPT.BITMAP.SCALE to 0.96, instead of 1, will give cleaner BITMAP images on a 300 dpi printer. (This corrects for the 72 ppi imagestream vs. the 75 dpi printer, using 4x4 device dots per bitmap pixel.) Also, values of 0.24, 0.48 and 0.72, instead of 0.25, 0.5 and 0.75, will also give cleaner images for reduced size output. In general, use integer multiples of 0.24 for a 300 dpi printer.

POSTSCRIPT.TEXTURE.SCALE [InitVariable]

POSTSCRIPT.TEXTURE.SCALE specifies an independent scale for the display of bitmap textures. The value represents the number of device space units per texture unit (bitmap bit). The default value is 4, which represents each bit of the texture as a 4x4 block, so that textures are approximately the same resolution as on the screen (for 300 dpi output devices, such as the Apple Laserwriter).

The PostScript package extends the allowed representations of a texture, beyond 16-bit FIXP and 16x16 bitmap, to ANY square bitmap. (If the bitmap is not square, its longer edge is truncated from the top or right to make it square.) Use this feature with caution, as large bitmap textures, or sizes other than multiples of 16 bits square, require large amounts of storage in the PostScript interpreter (in the printer controller), and can cause limitcheck errors when actually printing.

Anywhere that a texture or color can be used on an imagestream or in the specification of a BRUSH, you can instead give a FLOATP between 0.0 and 1.0 (inclusive) to represent a PostScript halftone gray shade. (0.0 is black and 1.0 is white. Specifically, the value sets the brightness of the shade.) The value you specify will not be range checked, and will be passed directly through to the PostScript setgray operator. (E.g. you can pass 0.33 as the color to DRAWLINE to get a dark gray line with approximately 67% of the pixels in the line black.)

POSTSCRIPT.IMAGESIZEFACTOR

[InitVariable]

POSTSCRIPT.IMAGESIZEFACTOR specifies an independent factor to change the overall size of the printed image. This re-sizing affects the entire printed output (specifically, it superimposes its effects upon those of POSTSCRIPT.BITMAP.SCALE and POSTSCRIPT.TEXTURE.SCALE). Values greater than 1 enlarge the printed image, and values less than 1 reduce it. An invalid POSTSCRIPT.IMAGESIZEFACTOR (i.e. not a positive, non-zero number) will use a value of 1. The BITMAPSCALE function for the POSTSCRIPT printer type does NOT consider the POSTSCRIPT.IMAGESIZEFACTOR when determining the scale factor for a bitmap.

## MISCELLANEOUS

The SCALE of a PostScript imagestream is 100. This is to allow enough resolution in the width information for fonts to enable TEdit to correctly fill and justify text.

The first time any PostScript imagestream is created (even if only to hardcopy a bitmap or window) the DEFAULTFONT is instantiated (unless a FONTS option was given to the OPENIMAGESTREAM, in which case the initial font for the imagestream will be set to that font, or to the CAR if a list).

The PostScript imagestream method for FILLPOLYGON uses the global variable FILL.WRULE as the default value for the WINDINGNUMBER argument. (This is the same variable which is used by the DISPLAY imagestream method for FILLPOLYGON.)

The PostScript imagestream method for OPENIMAGESTREAM (and, therefore, SEND.FILE.TO.PRINTER), supports an IMAGESIZEFACTOR option to change the size of the printed image. The IMAGESIZEFACTOR re-sizing is combined with the POSTSCRIPT.IMAGESIZEFACTOR to produce an overall re-sizing of the printed image. A HEADING option is also supported to give a running header on each page of output. The value of the HEADING option is printed at the top left of the page, followed by "Page " and the appropriate page number. They are printed in the

DEFAULTFONT (unless a FONTS option was given to the OPENIMAGESTREAM, in which case it will be that font, or to the CAR if a list).

The PostScript package is contained in the files: POSTSCRIPT.LCOM & PS-SEND.LCOM, with the source in the files: POSTSCRIPT & PS-SEND. The module PS-SEND.LCOM is required and will be loaded automatically when POSTSCRIPT.LCOM is loaded. It contains the function which is called by SEND.FILE.TO.PRINTER to actually transmit the file to the printer. It is, by its nature, quite site specific, so it is in a separate file to make modifying it for any site relatively simple. System record declarations required to compile POSTSCRIPT can be found in EXPORTS.ALL.

I'm pretty sure that the output generated by the PostScript imageops fully conforms to the Adobe Systems Document Structuring Conventions, Version 2.0, January 31, 1987.

### Including Other PostScript Operations

If you wish to insert your own specific PostScript operations into a PostScript imagestream, you can do so with the following functions:

(POSTSCRIPT.OUTSTR *STREAM STRING*) [Function]

POSTSCRIPT.OUTSTR outputs a string or value to the imagestream. *STREAM* must be an open PostScript imagestream. *STRING* is the value to output (STRINGP and LITATOM are most efficient, but any value can be output (its PRIN1 pname is used)).

(POSTSCRIPT.PUTCOMMAND *STREAM STRING ... STRING*) [NoSpread Function]

POSTSCRIPT.PUTCOMMAND is more general for sequences of commands and values. It calls POSTSCRIPT.OUTSTR repeatedly to output each of the *STRING* arguments to *STREAM*.

(\POSTSCRIPT.OUTCHARFN *STREAM CHAR*) [Function]

\POSTSCRIPT.OUTCHARFN is used to output the characters forming the text of a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream to output to, and *CHAR* is the CHARCODE of the character to output. The / (slash), ( and ) (parenthesis) characters will be quoted with /, and characters with ASCII values less than 32 (space) or greater than 126 (tilde) will be output as *lnnn* (in octal). \POSTSCRIPT.OUTCHARFN will output the ( character to open the string, if necessary. Use POSTSCRIPT.CLOSESTRING (below) to close the string.

(POSTSCRIPT.CLOSESTRING *STREAM*) [Function]

POSTSCRIPT.CLOSESTRING closes a PostScript string (e.g. the argument to a show or charpath operator). *STREAM* is the open PostScript imagestream. It is important to use POSTSCRIPT.CLOSESTRING to output the ) character to close the string, because it also clears the stream state flag that indicates that a string is in progress (otherwise, the next POSTSCRIPT.PUTCOMMAND would output the commands to close the string and show it).

### Warning

Do not attempt to create a PostScript font larger than about 600 points, as much of Interlisp's font information is stored in SMALLP integers, and too large a font would overflow the font's height, or the width for any of the wider characters. (I know that 600 points is a ridiculously large limit (about 8.3 inches), but I thought I'd better mention it, or someone might try it!)

**Changes from the Lyric Release**

The Medley release of this PostScript imagestream driver changed the default value of POSTSCRIPT.TEXTFILE.LANDSCAPE from T to NIL. It also added the support for the HEADING option.

**Known Problems/Limitations**

The output generated for a PostScript imagestream is rather brute force. It isn't particularly careful to generate the smallest output file for a given sequence of operations. Specifically, it often generates extra end-of-lines between PostScript operator sequences (this has no effect on the printed output, only on the file size).

Using BITMAPs or Functions as BRUSH arguments to the curve drawing functions is not supported, nor is using a non-ROUND BRUSH with DRAWCIRCLE or DRAWELLIPSE.

There is no support for NS character sets other than 0, and there is no translation of the character code values from NS encoding to PostScript encoding.

There is no support for color.

\POSTSCRIPT.OUTCHARFN is pretty wimpy in its handling of TAB characters. It just outputs 8 SPACES for the TAB.

I haven't yet documented how to build the .PSCFONT files for any new fonts that become available, I'll do that eventually.

---

---

**PS-PATCH**

---

---

By: Will Snow (snow.envos@xerox.com)

Requires: POSTSCRIPT

This module fixes some bugs in software other than postscript. If you are going to load sketch, load it BEFORE you load this patch. If you load sketch after loading this patch, evaluate the following form in an Interlisp Executive:

```
(MOVD (QUOTE NEW-SK-PICK-FONT) (QUOTE SK.PICK.FONT) NIL T)
```

This will make the printing of sketches with text in them reasonable.

---

**PS-RS232**

---

By: Matt Heffron (mheffron@orion.cf.uci.edu)

Requires: POSTSCRIPTSTREAM, PS-SEND, DLRS232C

The module PS-RS232 defines a printing host named PS-RS232 which sends PostScript output to a printer over the {RS232} port of the 11xx. It also puts a function onto AROUNDXITFNS which reinitializes the {RS232} port after returning from LOGOUT. The BaudRate and other parameters of the {RS232} port are controlled by the following variables.

**VARIABLES**

PS-RS232-BAUD [InitVariable]

This is the BaudRate for the {RS232} port output stream. Defaults to: 9600.

PS-RS232-DATABITS [InitVariable]

This is the BitsPerSerialChar for the {RS232} port output stream. Defaults to: 8.

PS-RS232-PARITY [InitVariable]

This is the Parity for the {RS232} port output stream. Defaults to: NONE.

PS-RS232-STOPBITS [InitVariable]

This is the NoOfStopBits for the {RS232} port output stream. Defaults to: 1.

PS-RS232-FLOWCONTROL [InitVariable]

This is the FlowControl for the {RS232} port output stream. Defaults to: XOnXOff.



STA2

| \*



---



---

## XORcursorPatch

---



---

By: Christopher Lane (Lane@sumex-aim.stanford.edu)

This document last edited on July 22, 1987

### INTRODUCTION

This module allows the 1186/Daybreak (only) users to twiddle the hardware bits so that they can have an inverting cursor (white on black and black on white instead of black on everything) and provides a patch to keep the system from undoing the effect when calling VIDEOCOLOR to reset the screen.

### USE

(DOVE.XOR.CURSOR *FLG*)

[Function]

The argument *FLG*, if T, will switch to the inverting cursor mode. If *FLG* is NIL it will switch back to normal mode. If *FLG* is a number between 0 and 15 then it is used as the 'mix-in rule' and has an effect according to the table below.

### Mix-in rules

|    | Screen    | Source   | Cursor Mode |                                                                                                                                                                                                                                                                                                                                                  |
|----|-----------|----------|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0  | All Black | None     |             | This table is relative to the normal mode of the display (1), normal screen, normal cursor in paint mode. The inverted display, (VIDEOCOLOR T), would be 13, inverted screen and inverted cursor in paint mode. There is probably a more precise or logical way to notate these modes, but this should give you a rough idea of what's available |
| 1  | Normal    | Normal   | Paint       |                                                                                                                                                                                                                                                                                                                                                  |
| 2  | Normal    | Inverted | Paint       |                                                                                                                                                                                                                                                                                                                                                  |
| 3  | Normal    | None     | mode.       |                                                                                                                                                                                                                                                                                                                                                  |
| 4  | Inverted  | Normal   | Erase       |                                                                                                                                                                                                                                                                                                                                                  |
| 5  | All Black | Inverted | Paint       |                                                                                                                                                                                                                                                                                                                                                  |
| 6  | Inverted  | Normal   | Invert      |                                                                                                                                                                                                                                                                                                                                                  |
| 7  | Normal    | Inverted | Erase       |                                                                                                                                                                                                                                                                                                                                                  |
| 8  | Inverted  | Inverted | Erase       |                                                                                                                                                                                                                                                                                                                                                  |
| 9  | Normal    | Normal   | Invert      |                                                                                                                                                                                                                                                                                                                                                  |
| 10 | All White | Inverted | Paint       |                                                                                                                                                                                                                                                                                                                                                  |
| 11 | Normal    | Normal   | Erase       |                                                                                                                                                                                                                                                                                                                                                  |
| 12 | Inverted  | None     |             |                                                                                                                                                                                                                                                                                                                                                  |
| 13 | Inverted  | Inverted | Paint       |                                                                                                                                                                                                                                                                                                                                                  |
| 14 | Inverted  | Normal   | Paint       |                                                                                                                                                                                                                                                                                                                                                  |
| 15 | All White | None     |             |                                                                                                                                                                                                                                                                                                                                                  |

### Note

The function is set up such that when trying different modes, you must do a (DOVE.XOR.CURSOR) (no argument) between calls.

No warranty expressed or implied, but we have been using it locally without problem (at least as far as I know). Enjoy.

The following notes explain my (woz) idea of what the edit-interface should be responsible for, and what constitutes and "editor".

An editor is a symbol whose function takes the args (structure props options) and starts an interactive editor on the structure. PROPS is a property list, and OPTIONS is a list of keywords, both affecting the behavior of the editor. The property :completion-fn specifies the function to be called when the user completes the edit. The completion-fn will be called with the arguments (structure props options changed?), where structure is the edited structure (even on abort, so that the edit interface could implement "undo abort"), props and options are as specified on the call to the editor and will be used to figure out what to do with this completion, and changed? is NIL (no changes made), T (changes made), or :ABORT (user wants to abort changes). It is then up to the completion function to do the right thing with the result of the edit.

The goal is that the editor doesn't know anything about who started it, where the structure came from, or what to do with it when it's done. And the edit-interface doesn't know anything about the editor's data structures.

In the case of open edit sessions (open or shrunk):

If the editor is told to start an edit, the editor must look for one already existing that matches (this can't be the responsibility of the edit interface, because it doesn't know about existence of open edits). the editor should restart the existing edit, processing any new props or options appropriately.

The markaschanged issue:

Since the editor knows it may have open edits, it needs to provide a hook for when the world gets changed underneath the editor. in this world this means markaschanged. in this case the editor should try to restart itself with the new structure. in other words, sedit::markaschangedfn should call edit-definition to start a new edit. the editor will then notice it has a matching edit open and restart itself with the new info.

This model is complicated by the fact that markaschanged gets called as a result of completion.

Presently \*ignore-changes-on-completion\* controls the behavior in this case. Ideally, the editor would just say "i know it got changed, i just changed it!", and it would ignore the call.

The new version of SEdit (1/25/91) is very close to this definition, with the following exceptions, which can be fixed upon implementation of this edit interface design:

- the completion-fn is called with (context structure changed?) since all of SEDITE's completion-fns expect these args. this should be fixed in handle-completion.
- in the abort case, undo is run until there are no more changes to undo, since sedit is sometimes handed structures "in place", destructive edits need to be undone, and thus the completion-fn never gets to see the edits. this can be fixed in the function complete.

-

To make this editor active, call (il:editmode editor-name), where editor-name is the symbol defined above. The function xcl::edit will then start the active editor.

xcl::edit-expression provides an example of starting the editor on an unnamed structure, where eqness is expected upon completion.

xcl::edit-definition provides a replacement for il:editdef, if il:getdef and il:putdef work correctly on all types.

:dontwait

the editor (sedit) should not wait. don't wait is part of the editinterface features. if its going to wait, the editinterface should create the event and bind it in the completionfn to the notified by the completionfn.

need published edit-expression command. same as ed, but takes an expression, waits for once-only completion, and returns the expression. eg for FIX

sedit should provide a published interface to creating gaps, for use by make-prototype-defn.

edit interface may need a way to ask editor if it has an open edit for name/type

deal with editdates

an editor used to install itself by replacing EDITL and EDITE, then editmode was created to look for Definition-for-EDITE property on the symbol returned by editmode. now an editor is defined as a function which takes (expr props options) and implements at least the props :completion-fn

and :root-changed-fn. EDITMODE then returns the name of this function so the current editor can be applied.



```

prompt% mkdir $YOURWORKDIR/init.$ARCH
-- $ARCH sun4 sparc , sun3 mc68020
prompt% cd $YOURWORKDIR/bin
rompt% makeinitlde -e

-- "-DINIT"

INITLDE '$YOURWORKDIR/init.$ARCH/ lde'

: INITLDE
cg3,cg6
cg3,cg6 INITLDE,
makeinitlde OPTFLAGS -DDISPLAYBUFFER
$YOURWORKDIR/init.$ARCH/
makeinitlde -e

```

## 2. MAKEINIT/LOADUP

```

MAKEINIT/LOADUP Venue
Venue

```

### FILE: runloadup

```

set LDE = HOME/maiko/sunos4.sparc/lde
set LDEPATH = HOME/maiko/sunos4.sparc
set INITLDE = HOME/maiko/init.sparc/lde
set INITLDEPATH = HOME/maiko/init.sparc
set FULL_SYSOUT = /usr/local/sysouts/FULL.SYSOUT
set FIRST_REM_CM = HOME/SUNLOADUP/XREM.CM
set SECOND_REM_CM = HOME/SUNLOADUP/LOADUP-REM.CM

```

### FILE: INIT.MAKEINIT

```

DIRECTORIES
FILESETS LOAD
FASTINIT.DFASL LOAD
MAKEINIT4
DLFIXINIT3

```

### FILE: LOADUP-REM.CM

```

LOADUP.LISP LOAD

```

## MAKEINIT/LOADUP

1. MAKEINIT/LOADUP
  - ~/I-NEW, ~/I-NEW.LCOM
  - ~/INIT.SYSOUT

- ~/INIT.DLINIT
- ~/lisp.virtualmem
- ~/REM.CM
- ~/SUNLOADUP/LOADUP.LOG

2. [REDACTED]

[REDACTED] SUNLOADUP ■ runloadup [REDACTED].

prompt% cd ~/SUNLOADUP  
prompt% runloadup

[REDACTED] SUNLOADUP ■ LISP.SYSOUT [REDACTED].

---

SYSOUT [REDACTED] [REDACTED]

---

LOADUP.LISP [REDACTED] FILESETS [REDACTED]  
SYSOUT [REDACTED].

---

Ethernet [REDACTED] SYSOUT

---

LISP.SYSOUT ■ Ethernet [REDACTED] SYSOUT [REDACTED]  
[REDACTED].

[REDACTED]:

[1] FILESETS [REDACTED]  
1LISPSET ■ LLEther [REDACTED]

[2] LOADUP.LISP [REDACTED]  
DPUPFTP LLNS TRSERVER SPP COURIER NSPRINT  
CLEARINGHOUSE NSFILING INTERPRESS [REDACTED]

[3] XREM.CM [REDACTED]  
[REDACTED] 2 [REDACTED]  
(MOVD (QUOTE \ETHEREVENTFN) (QUOTE \ETHEREVENTFN-  
)  
(MOVD (QUOTE NILL) (QUOTE \ETHEREVENTFN))

[4] runloadup [REDACTED]

[REDACTED]: SYSOUT [REDACTED].

---

D-Machine [REDACTED] SYSOUT

---

LIS.SYSOUT ■ D-Machine [REDACTED] (D-Machine [REDACTED]  
[REDACTED]) [REDACTED] SYSOUT [REDACTED].

[REDACTED]:

[1] LOADUP.LISP [REDACTED]  
DPUPFT DISKDLION DOVEINPUTOUTPUT DOVEDISK  
DOVEDISPLAY DOVEMISC DOVEETHER DOVEFLOPPY  
DSKDISPLAY FLOPPY [REDACTED]

[2] runloadup █

█: SYSOUT █.

---

## █SYSOUT

---

LISP.SYSOUT █Interlisp BYTE Compiler, XCL compiler  
█SYSOUT █.

█:

[1] LOADUP.LISP█  
DLAP BYTECOMPILER COMPILE FASDUMP XCL-COMPILER  
DPUPFT DISKDLION DOVEINPUTOUTPUT DOVEDISK  
DOVEDISPLAY DOVEMISC DOVEETHER DOVEFLOPPY  
DSKDISPLAY FLOPPY █

[2] runloadup █

█: █

Undefined function SPECVARS █ Break █.  
SPECVARS █ █COMPILE █.



# TCP-IP

---

The Transport Control Protocol - Internet Protocol (TCP-IP) family of networking protocols was developed under the auspices of the Department of Defense to standardize communication mechanisms within Department of Defense networks such as the ARPANET.

The protocols are documented in a collection of working papers known as Requests for Comments (RFCs). Appropriate RFC numbers appear throughout this document as new protocols are introduced.

## Requirements

---

TCP-IP has both hardware and software requirements.

### Hardware

- Ethernet
- Cooperating host (yours or theirs)
- 110X/118X with an Ethernet controller (usually co-resident on an otherwise inhabited module)
- XCVR interface cable
- XCVR installed on an Ethernet with a logical (direct or internet) connection to the cooperating host.

### Software

You need the files enumerated in the section titled "Interlisp Files." Files loaded by the high-level modules TCPFTP, TCPFTPSRV, TCPCHAT, and TCPTFTP automatically load their dependencies. If you load files from floppy, you must load their dependencies first:

| File       | Dependencies                |
|------------|-----------------------------|
| TCP        | TCPLLLIP                    |
| TCPCHAT    | TCP, CHAT                   |
| TCPCONFIG  | None                        |
| TCPDEBUG   | TCP                         |
| TCPDOMAIN  | TCPUDP                      |
| TCPFTP     | TCPNAMES, TCP               |
| TCPFTPSRV  | TCPFTP                      |
| TCPHTE     | None                        |
| TCPLLAR    | None                        |
| TCPLLLICMP | None                        |
| TCPLLLIP   | TCPHTE, TCPLLLICMP, TCPLLAR |
| TCPNAMES   | None                        |
| TCPTFTP    | TCPUDP                      |
| TCPUDP     | TCPLLLIP                    |

## User Interface

---

TCP does not have a user interface module of its own. Its functions and variables are accessible via an Interlisp Executive, and you can direct some of its debugging information to a window.

As a network protocol module, it extends capability to other programs which may have their own window interfaces, for example, Chat and FileBrowser.

## Installation

---

The first step in installing TCP-IP is to add your workstation to a network supporting TCP-IP and communications with others on the net. The rest of this section contains a step-by-step set of directions for this installation.

After you are on the network, load the required .LCOM modules for the type of service you want. For a full description of these modules, see the Interlisp Files section.

| Module    | Implementation                      |
|-----------|-------------------------------------|
| TCPFTP    | TCP-based file transfer protocol    |
| TCPFTPSRV | TCP-based FTP server                |
| TCPCHAT   | TELNET protocol for the Chat system |
| TCPTFTP   | TFTP protocol                       |

### Obtaining Network Addresses

The first thing you need to do is to get a TCP-IP address assigned to each of your workstations from your network administrator. If your site supports Domains, get the name of your local domain and the addresses of your domain server(s) from your network administrator. You will also need to know the network addresses and operating system of the hosts you want to communicate with and the addresses of any network gateways you have.

Note: The maximum length of the domain and organization fields is 20 characters each.

Be sure to find out whether your net is a true Class A, B or C network and is not broken up into subnets. If it is broken up into subnets, be sure to read the discussion on SUBNETMASKs in the Primer on IP Networks section.

### Differences between TCP and the Medley File Systems

When running TCP-IP to a Sun from an 11xx, directory enumeration on an unmatched directory path returns a listing for the top-level directory of the logged-in user. The TCPFTP protocol does not support directory creation.

Warning: When running TCP-IP to a Sun, a file which is still open on the Sun can be deleted.

### Creating HOST.TXT File

Create a HOSTS.TXT file containing entries for the TCP-IP hosts needed by the user community and place a copy of the file on either a directory contained in the DIRECTORIES search path of each workstation on the net or the local disk of each Interlisp workstation.

The following is a sample HOSTS.TXT file:

```
; Hosts.txt,
; Internet Hosts Table for Networks 192.20.10.0 and 174.23.0.0
; 12-Dec-86
;
; The format of this file is documented in RFC 810, "DoD Internet
; Host Table Specification", which is available online at SRI-NIC
; as the file
; [SRI-NIC] <RFC>RFC952.TXT
;
; It may be retrieved via FTP using username ANONYMOUS with
; any password.
;
; or as the file
;
; [INDIGO] <RFC>RFC952.TXT
; Read access to GV World. Valid GV credentials required.
;
; The format for entries is:
;
; GATEWAY: ADDR, ADDR : NAME : CPUTYPE : OPSYS : PROTOCOLS :
; HOST: ADDR, ALTERNATE-ADDR (if any): HOSTNAME,NICKNAME : CPUTYPE :
; OPSYS : PROTOCOLS :
;
; Where:
;; ADDR = internet address in decimal, e.g., 26.0.0.73
;; CPUTYPE = machine type (Xerox-11xx, VAX-11/780, SUN, etc.)
;; OPSYS = operating system (UNIX, TOPS20, TENEX, VMS, Interlisp, etc.)
;; PROTOCOLS = transport/service (TCP/TELNET, TCP/FTP, etc.)
;; : (colon) = field delimiter
;; :: (2 colons, NO space between) = null field
;
HOST : 192.20.10.1 : Bach : Xerox-1108 : Interlisp : TCP/TELNET, TCP/FTP
:
HOST : 192.20.10.3 : PARC-VAXC : VAX-11/780 : UNIX : TCP/TELNET, TCP/FTP
:
HOST : 192.20.10.15 : Oberon : VAX-11/780 : VMS : TCP/TELNET, TCP/FTP :
HOST : 192.20.10.71 : Explorer : TI-EXPLORER : TOPS-20 : TCP/TELNET,
TCP/FTP :
HOST : 174.23.77.22 : Sunrise : SUN : UNIX : TCP/TELNET, TCP/FTP :
HOST : 174.23.30.21 : Rutgers : VAX-11/780 : TOPS-20 : TCP/TELNET,
TCP/FTP :
HOST : 174.23.76.21 : Simba : SYMBOLICS : SYMBOLICS-3600 : TCP/TELNET,
TFP/FTP :
GATEWAY : 192.20.10.240, 174.23.77.250 : Hellsgate : VMS : IP/GW :
```

This example shows a host table that indicates that there are four hosts (Bach, PARC-VAXC, Oberon, and Explorer) on net 192.20.10.0, three hosts (Sunrise, Rutgers, and Simba) on net 174.23.0.0 and a gateway (Hellsgate) that connects the two.

In regard to the OPSYS field in the HOSTS.TXT file, it is preferable to use values recognized by the Lisp variable NETWORKOSTYPES. Interlisp is the default value if a host's OSType is not declared.

Note that if any host is accessible via another network protocol (for example, PUP or NS), you may desire to call the host by an unambiguous name when it is accessed via TCP. You can do this by giving it an unambiguous name in the HOSTS.TXT file.

If you ever modify the HOSTS.TXT table after TCP.LCOM has been loaded, use the function (`\HTE.READ.FILE 'HOSTTABLE`) to reread the file.

For example,

```
(\HTE.READ.FILE '{DSK}<LISPFILES>HOSTS.TXT)
```

TCP.ALWAYS.READ.HOSTS.FILE

[Variable]

Initially set to T. Setting it to NIL causes the system to parse the HOSTS.TXT file only when the filename (stored in the configuration file) is different from the previously read filename, or the write date of the file has changed. The HOSTS.TXT file will always be read at least once when loading the software into a clean sysout.

## Creating the Local IP.INIT File

TCP.CONFIGURE brings up a menu that you complete.

```
Exec 3 (XCL)
3/36> (tcp.configure)
#<window @ 47,55554>
3/37>
```

```
TCP Configuration
Apply! Reset! Quit!

Host Name: Panther
Host Address: 13.2.77.8
Network Address: 13.2.77.0
Subnet mask: 13.252.205.0
Default Gateway: 192.20.10.240
Local Domain:
Domain servers:
Hosts.txt file: {Dsk}<Lispfiles>HOSTS.TXT
```

```
Writing {dsk}ip.init... done.

TCP Configuration
Apply! Reset! Quit!

Host Name: Panther
Host Address: 13.2.77.8
Network Address: 13.2.77.0
Subnet mask: 13.252.205.0
Default Gateway: 192.20.10.240
Local Domain:
Domain servers:
Hosts.txt file: {Dsk}<Lispfiles>HOSTS.TXT
```

If any field does not apply to your site, leave it blank.

Selecting **Apply!** writes the file {DSK}<LISPFILES>IP.INIT to the local disk.

Note: The file {DSK}<LISPFILES>IP.INIT must exist on each Interlisp machine before TCP.LCOM is loaded. And this file *must* remain on the workstation and must not be copied to other workstations. Also, the font GACHA 12 MRR must be available.

Selecting **Reset!** resets the menu to the original state.

Selecting **Quit!** closes the window.

You must perform the TCP.CONFIGURE step individually on each workstation, but you need to perform it only once. As long as there is an IP.INIT file on the workstation, the TCP-IP module will be configured automatically whenever it is loaded or initialized.

If you change your IP.INIT file while TCP-IP is running, you will be prompted to confirm **Restarting TCP**. In most cases, you should confirm the restart.

## Adding Host and Operating System Names to NETWORKOSTYPES

The variable NETWORKOSTYPES is used during Chat to determine the sequence of characters to send when performing auto-login. There should be an entry in NETWORKOSTYPES for each TCP host that you want to communicate with in the form (TCPHOSTNAME . OSTYPE).

For example:

```
((SUNRISE . UNIX) (RUTGERS . TOPS-20) etc)
```

## End-of-Line Conventions

To ensure correct line spacing, set the TCPFTP.EOL.CONVENTION switch. The associated function takes one argument, TYPE, which sets it correctly. TYPE can be one of the following:

|       |                                     |
|-------|-------------------------------------|
| CR    | Set EOL to CR                       |
| LF    | Set EOL to LF                       |
| CRLF  | Set EOL to CRLF                     |
| OS    | Set it to something based on the OS |
| Other | Set it to the default (CRLF)        |

## Loading TCP

Make sure the variables `DIRECTORIES` and `LISPUSERSDIRECTORIES` point to the location of the `.LCOM` files of TCP-IP, or that they are in the connected directory.

You can then load `TCP.LCOM` which in turn loads its dependent files.

If you plan to do TCP file transfers, load `TCPFTP`.

If you plan to use the 11xx Lisp workstation as a `TCPFTP` Server host, load `TCPFTP.SRV`. To start the server, evaluate `(TCPFTP.SERVER)`. An Interlisp machine running the `TCPFTP` server should be identified as a TOPS-20 machine in the other Interlisp machines' `HOSTS.TXT` table. It will thus masquerade as a TOPS-20 server.

The rest is automatic. You can treat an Interlisp host running the server just like any other `TCPFTP` server. The default path for resolving filenames is `{DSK}<LISPFILES>`, but you can change or override it.

For example, assume `{ERIC}` is a machine running the FTP server. From another machine which has `TCPFTP` loaded, you can do `SEE {ERIC}{FLOPPY}FOO`, which will type out the file `FOO` located on the floppy drive of `{ERIC}`.

If you plan to Chat to a TCP host, load `CHAT`, `CHATTERMINAL`, `DMCHAT` and then `TCPCHAT.LCOM`. Be sure that hosts with which you wish to chat have their `NETWORKOSTYPES` set.

If you plan to use the TCP Trivial File Transfer Protocol, load `TCPTFTP`.

Interlisp's `TCPTFTP` also provides a `TCPTFTP` server. Load `TCPTFTP.LCOM` and evaluate `(TFTP.SERVER)`. You can then use the appropriate TFTP commands to copy files from the Interlisp machine; for example `TFTP.PUT` and `TFTP.GET`.

## Verifying TCP Connections

Load `TCPDEBUG`. Execute `(TCPTRACE T)` and you will be prompted to open a window to show TCP packets. Select `INCOMING`, `OUTGOING` and `CONTENTS` from the window's menu. If the host that you are communicating with has a TCP echoserver process you can then try `(TCP.ECHOTEST 'HOSTNAME 3)`. For example, using the above `HOSTS.TXT` file this would be `(TCP.ECHOTEST 'SIMBA 3)`.

You will be prompted to open a window for the echo test and should see text, for example:

```
This is byte number 21
This is byte number 45
This is byte number 69
```

You should also see packets being sent and received in the `TCPTRACE` window.

Note: If a remote host is not running a TCP echo server process you will not get this response.

---

## Connecting, Transferring Files, and Chatting to a Host

First log in to the host by typing `(LOGIN 'HOST)`; for example, `(LOGIN 'SUNRISE)`. You will then be prompted for a user name and password. This will be sent to the host when you attempt to `CONNECT` or `Chat`.

Use the command `CONN {HOST}<DIRECTORY>SUBDIR>` to connect to a particular host. The local directory delimiters `<` and `>` can be used when connecting or file transferring. When communicating with a remote host you can specify the directory path as `<DIRECTORY> SUBDIRECTORY> SUBDIRECTORY. . .`, and the appropriate delimiters are presented to the remote host. Determination of what delimiter is presented depends upon the value of the `OSTYPE` field in the `HOSTS.TXT` file. If the field is empty, `OSTYPE = 'Interlisp'` is the default.

Using the above `HOST.TXT` file as an example you can do the following:

```

 UNIX CONN {SUNRISE}<DIR>SUBDIR>SUBDIR>
 VMS CONN {OBERON}<DIR>SUBDIR>SUBDIR>
 TOPS-20 CONN {RUTGERS}<DIR>SUBDIR>
 SYMBOLICS-3600 CONN {SIMBA}<DIR>SUBDIR>

```

You can then do a `DIR` of the remote host, `COPYFILE` files to and from the host, assuming `TCPFTP` is loaded, `MAKEFILE`, etc.

Since the `TCPFTP` specification does not specify file type conventions, the variable `TCP.DEFAULT.FILETYPES` is used to associate a file's extension with the type of file it is. It is a list in the form (extension . type); for example,

```
((LCOM . BINARY) (TXT . TEXT) etc)
```

Since UNIX systems are case-sensitive, you should also have the lower case version of the file extensions on this list. If a file extension is not found on this list, the variable `TCP.DEFAULTFILETYPE` is used as the default file type during file transfers.

To Chat to a remote host, select Chat from the background menu and enter the host name when prompted. You will be prompted for a Chat window and should then be able to chat to the host. If you have problems opening the Chat connection, try `(CHAT 'HOST 'NONE)`. This will suppress the automatic login.

Warning: If no username or password has been provided for the TCP host, the default ID and password will be sent. This may create a security hazard.

---

## Making a Sysout that Contains TCP-IP

---

1. Load Medley sysout.
2. Create TCP host table.
3. Load `TCPCONFIG.LCOM` and run `TCP.CONFIGURE` if there is no `IP.INIT` file on local disk.
4. Load `TCP.LCOM`, `TCPFTP.LCOM`, `TCPCHAT.LCOM`.
5. Load `TCPDEBUG.LCOM` if you always want to have trace and echo facilities available.
6. Evaluate `(TCP.STOP)`
7. Evaluate `(STOPIP)`
8. Evaluate
 

```
(SETQ RESTARTETHERFNS (LIST ' (LAMBDA NIL (AND \IPFLG
(\IPINIT)))))
```
9. Load any other files that you want in this sysout.

10. Evaluate SYSOUT to the device of your choice. Evaluate (`\TCP.INIT`) to re-enable TCP.
11. Load TCP sysout on other machine.
12. Create TCP host table.
13. Evaluate (`TCP.CONFIGURE`) and identify the new machine.
14. Evaluate (`\TCP.INIT`) to re-enable TCP.
15. Evaluate (`\IPINIT`) to restart the IPLISTENER process.

## TCP-IP Protocol Layers

---

The TCP-IP family consists of four principal protocol layers: the link layer, the internet layer, the transport layer, and the application layer.

### Link Layer

The physical link layer, the medium for transferring packets between hosts, is assumed to be any medium capable of transporting packets of data between hosts. Common link layers in this family include the Ethernet and the ARPANET.

The Address Resolution (AR) Protocol enables hosts to map between internet addresses and link layer addresses.

For example, the internet layer protocol IP (see below) uses a 32-bit combined unique host and network address; the host address field is of variable size and depends on the pattern encoded in the high-order bits of the address. On the other hand, the 10 MB Ethernet uses a fixed-size 48-bit unique host address. The Address Resolution protocol, documented in RFC826, allows hosts to discover dynamically the link layer address equivalents of other internet hosts.

### Internet Layer

The internet layer is responsible for routing packets between hosts. Unlike the link layer, the internet layer is capable of moving packets between hosts that are not connected to the same network. The term IP in TCP-IP refers to the Internet Protocol, the protocol that performs this task in the TCP-IP family. IP is documented in RFC791. IP is not assumed to be error-free; packets may be lost or duplicated while moving through the internet. It is the responsibility of the transport layer (see below) to guarantee perfect delivery, should the client require it.

IP also depends on an associated protocol called the Internet Control Message Protocol (ICMP). ICMP is responsible for handling exception conditions that arise between hosts using IP. Such conditions include the inability to deliver packets, errors in packet formats, etc. ICMP is documented in RFC792.

### Transport Layer

The transport layer is responsible for assuring error-free, duplicate-free, sequenced delivery of packets between communicating processes. The most common transport layer is TCP, the Transport Control Protocol. TCP maintains the appearance of a perfect byte stream between processes. TCP is documented in RFC793.



An unreliable transport layer called the User Datagram Protocol (UDP) allows for packet exchange between communicating processes, but makes no attempt to guarantee delivery, suppress duplication, etc. Clients of UDP must provide their own error-recovery mechanisms if necessary. UDP is documented in RFC768.

## Application Layer

Many applications exist in the TCP-IP family. The most common applications are file transfer, virtual terminal interaction, and mail delivery.

### File Transfer

Two principal file transfer applications are in use: FTP, based on TCP and documented in RFC765; and TFTP (the Trivial File Transfer Protocol), based on UDP and documented in RFC783. Both are implemented in Interlisp, and are discussed at greater length below.

### Virtual Terminal Interaction

The TELNET protocol, documented in RFC854, specifies the protocol for virtual terminal interaction between a user and a remote system. The Chat module will use the TELNET protocol to connect to TCP-only hosts.

### Mail Delivery

The Simple Mail Transfer Protocol (SMTP) enables the delivery of mail between system elements using TCP. It is not currently implemented in Interlisp. SMTP is documented in RFC821. The format of messages is described in RFC822.

---

## Primer on IP Networks

The Internet Protocol internetwork is a collection of IP networks, a subset of which may communicate with each other. Each network is assigned an IP address, which is composed of a network number and a host number. No two hosts in the internetwork have the same network and host number combination; the composition of the network and host number for a particular host unambiguously identifies that host within the internetwork.

### Network Addresses

The address space of the internetwork is formed of the concatenated network and host numbers of its constituent hosts, and is 32 bits long. This 32-bit address space is currently partitioned into three classes of network addresses, known as class-A, class-B, and class-C:

Class-A addresses consist of 7 bits of network number and 24 bits of host number.

Class-B addresses consist of 14 bits of network number and 16 bits of host number.

Class-C addresses consist of 21 bits of network number and 8 bits of host number.

Thus, there may be 128 class-A networks, 16,384 class-B networks, and over two million class-C networks. In addition, a single class-A network has the capacity to address over 16 million hosts, while a class-C network can address only 255 hosts. The class to which a particular IP network belongs may be determined by examining the most significant bits of its address.

Network number assignments are strictly controlled by a central authority. Institutions requesting network assignments are given class-A, -B, or -C networks

depending on their estimated eventual size (numbers of hosts). Sites without assigned network numbers may request an assigned number by contacting:

Joyce Reynolds  
 USC Information Sciences Institute  
 4676 Admiralty Way  
 Marina del Rey, California 90292-6695  
 Phone: (213) 822-1511  
 ARPANET: JKREYNOLDS@USC-ISIF.ARPA

IP addresses are normally stored or exchanged as single 32-bit numbers. The printed representation of an IP address takes the form W.X.Y.Z, where W through Z are the decimal equivalents of each of the 8-bit bytes that constitute the address. Class-A addresses are of the form N.H.H.H; class-B addresses are of the form N.N.H.H; and class-C addresses are of the form N.N.N.H, where N indicates a byte of the network number, and H indicates a byte of the host number.

Class-A: N.H.H.H The first number is between 0-127 (for example, 122.0.2.1)

Class-B: N.N.H.H The first number is between 128-191 (for example, 153.4.23.5)

Class-C: N.N.N.H The first number is between 192-255 (for example, 194.5.67.3)

For example, 36.47.0.12 is an address on network 36, a class-A network; and 192.10.200.1 is an address on network 192.10.200, a class-C address.

## Broadcast Address

The Internet Protocol defines an address in which the host field contains all ones to be a broadcast address for its network. Thus, the address 36.255.255.255 is the broadcast address on network 36, and 192.10.200.255 is the broadcast address on network 192.10.200.

## Subnets

It is quite common for class-B networks to be partitioned into a set of smaller subnetworks, which are really class-C networks, but have the wrong network number to be recognized as class-C networks. This is just as common is partitioning a class-A network into many class-B subnetworks. An implementation of TCP-IP that is not prepared to handle this violation of the IP standard will not be able to communicate with hosts on the same network but different subnetworks. Fortunately, extending an IP implementation to support subnetworks is straightforward.

SUBNETMASK is a 32-bit parameter that resembles an IP address. The purpose of the mask is to enable a host to determine when a destination IP address is or is not on the same subnet as the sending host itself.

The SUBNETMASK has the following properties:

- The bitwise-AND of a source host's address (for example, this machine) and the SUBNETMASK must be equal to the bitwise-AND of a destination host's address and the SUBNETMASK if and only if the two hosts are on the same subnetwork.
- The bitwise-AND of a source host's address and the SUBNETMASK must not be equal to the bitwise-AND of a destination host's address and the SUBNETMASK if and only if the two hosts are on different subnetworks.

As an example, consider network 39.0.0.0. This is a class-A network. Suppose this network consists of a number of subnetworks; for example, subnetworks with numbers like 39.47.\*.\* and 39.9.\*.\*. According to the IP specification, these subnetworks should

really be one monolithic network, such that a host desiring to communicate with any other host whose address begins with 39... should have to take no special action with regard to routing packets to that host. Let us assume that this is not the case. The only way a machine has of telling which hosts are on different networks is to compare the masked version of the address with the masked version of its own address.

To continue the example further, assume the following:

Host A has address 39.9.0.6. Host A's SUBNETMASK is 39.255.0.0.

Host B has address 39.9.0.7. Host B's SUBNETMASK is also 39.255.0.0.

Host C has address 39.47.0.6. Host C's SUBNETMASK is also 39.255.0.0.

When host A sends to host B, it compares its masked address with host B's masked address, and finds them equal:

$39.9.0.6 \text{ AND } 39.255.0.0 = 39.9.0.0$ ;  $39.9.0.7 \text{ AND } 39.255.0.0 = 39.9.0.0$

However, when host A sends to host C, it finds the masked comparison does not match:

$39.9.0.6 \text{ AND } 39.255.0.0 = 39.9.0.0$ ;  $39.47.0.6 \text{ AND } 39.255.0.0 = 36.47.0.0$

Class-A networks that are subdivided into class-B subnetworks have SUBNETMASKs that look like X.255.0.0, where X is the class-A network number. Likewise, class-B networks subdivided into class-C subnetworks have SUBNETMASKs that look like X.Y.255.0, where X.Y is the class-B network number. Finally, networks in which subnet routing is not in use have SUBNETMASKs identical to their network addresses. For example, if network 36 did not use subnet routing, its SUBNETMASK would be 36.0.0.0.

The definitive document on this approach to subnetwork routing is RFC940.

---

## Interlisp Files

The files that implement the TCP-IP protocol suite are divided into two classes: those that implement low-level functionality, normally not of interest to general users, and those that implement higher-level functionality for user programs (either application or transport layer protocols).

The higher-level functions reside in the files TCP, TCPDEBUG, TCPFTP, TCPCHAT, TCPNAMES, TCPUDP, and TCPFTP.

|           |                                                                                                                                                                                         |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| TCP       | The TCP layer. Implements TCP streams, based on the buffered TCP device (for example , BIN runs in microcode).                                                                          |
| TCPDEBUG  | Contains routines to help debug TCP and TCP-based applications.                                                                                                                         |
| TCPFTP    | Contains the TCP-based file transfer protocol. Creates a new virtual I/O device, allowing transparent filing operations with TCP-only hosts.                                            |
| TCPFTPSRV | Contains the TCP-based FTP server program. When the server program is running on a Xerox 1100-series workstation, other TCP-based hosts may transfer files to and from the workstation. |
| TCPNAMES  | Implements translation of file name formats between operating system types.                                                                                                             |
| TCPCHAT   | Implements the TELNET protocol for the Chat system.                                                                                                                                     |
| TCPUDP    | Contains the UDP layer.                                                                                                                                                                 |

TCPTFTP      Implements the TFTP protocol. Creates a buffered TFTP device to allow efficient bulk transfer between hosts.

The low-level functions reside in the files TCPLLLIP, TCPLLLICMP, TCPLLLAR, TCPHTE, and TCPCONFIG.

TCPLLLIP      Implements the IP layer.

TCPLLLICMP    Implements ICMP for IP.

TCPLLLAR      Implements ARP for the 3- and 10-megabyte Ethernets.

TCPHTE        Implements the functionality necessary to parse RFC810-style HOSTS.TXT files. This allows name-to-address translation within the Interlisp host.

TCPCONFIG     Provides a function to carry on a configuration dialog when TCP-IP is first installed on a machine. This file needs to be loaded only once, to produce the file {DSK}IP.INIT. Thereafter, TCPCONFIG is needed only to reestablish or modify IP parameters.

TCPDOMAIN     Implements a domain host address lookup client.

## TCP

TCP implements the transport control protocol for Interlisp. After TCP is loaded, Interlisp supports a TCP stream capable of bidirectional I/O to a remote system element. The following functions are intended for use by applications programs.

(TCP.OPEN *DST.HOST DST.PORT SRC.PORT MODE ACCESS NOERRORFLG*  
*OPTIONS*) [Function]

Opens a TCP stream to *DST.PORT* on *DST.HOST* from *SRC.PORT*.

*DST.HOST* can be a host name, an IP host address in text format (such as 192.10.200.1), or the 32-bit integer representation of an IP host address as returned by the function DODIP.HOSTP (which is documented under TCPLLLIP).

*DST.PORT* is a 16-bit number representing a TCP port open in LISTENING mode on the remote system.

*SRC.PORT* is also a 16-bit number, but may be supplied as NIL to obtain a defaulted unique local port number.

*MODE* is either ACTIVE, meaning to act as initiator of the connection, or PASSIVE, meaning to wait for a remote system element to initiate the connection.

*ACCESS* is either INPUT, OUTPUT, or APPEND (OUTPUT and APPEND are treated in the same manner).

If *NOERRORFLG* is non-NIL, TCP.OPEN will return NIL if the connection fails; otherwise, TCP.OPEN will call ERROR to signal failure.

*OPTIONS* is an optional parameter which allows the application program to control some of the characteristics of the TCP connection. *OPTIONS* is supplied in property-list format. Currently, the only recognized option is MAXSEG, whose value should be the number of data bytes the remote TCP sender is allowed to place into a single TCP segment (Ethernet packet). The maximum value of MAXSEG is 536.

If `TCP.OPEN` succeeds, it returns a `STREAM` open as specified by `ACCESS`. The generic operations `BIN`, `BOUT`, `PEEKBIN`, `BINS`, `BOUTS`, `READP`, `EOFP`, `OPENP`, `GETFILEPTR`, `FORCEOUTPUT`, and `CLOSEF` may be performed on streams opened for suitable access.

(`TCP.OTHER.STREAM STREAM`) [Function]

Returns the `STREAM` open in the other direction with respect to `STREAM` (for example, if `STREAM` is open for `INPUT`, `TCP.OTHER.STREAM` returns a `STREAM` open for `OUTPUT`, and vice versa).

(`TCP.URGENT.EVENT STREAM`) [Function]

Returns an event upon which a user process may wait for `URGENT` data to arrive on `STREAM`.

(`TCP.URGENTP STREAM`) [Function]

Returns `T` if `STREAM` is currently reading `URGENT` data.

(`TCP.URGENT.MARK STREAM`) [Function]

Marks the current point in `STREAM` as the end of `URGENT` data. `STREAM` must be open for `OUTPUT`.

(`TCP.CLOSE.SENDER STREAM`) [Function]

Closes the output side of `STREAM`, which may be either the `INPUT` or `OUTPUT` stream for the connection. This function differs from `CLOSEF` in that the `INPUT` side of the connection is not closed (although the remote system element may close the connection once the local output side of the connection is closed).

(`TCP.STOP`) [Function]

Disables the TCP protocol, closing all open TCP streams.

(`\TCP.INIT`) [Function]

(Re)initializes the TCP module.

`\TCP.DEFAULT.RECEIVE.WINDOW` [Variable]

Is the default number of bytes allowed outstanding from the remote system. It is initially 4,096.

`\TCP.DEFAULT.USER.TIMEOUT` [Variable]

Is the default number of milliseconds a remote system element is allowed to remain silent before the TCP connection is declared broken. It is initially 60,000.

## TCPDEBUG

TCPDEBUG implements tracing and test functions used to debug TCP and TCP-based applications.

(TCPTRACE)

[Function]

Opens a trace window and attaches a menu to the window's top.

| TCP Trace Window                                                                                                                    |             |           |
|-------------------------------------------------------------------------------------------------------------------------------------|-------------|-----------|
| Incoming                                                                                                                            | Transitions | Contents  |
| Time                                                                                                                                | Outgoing    | Checksums |
| RECV: from 192.9.200.1:23 to 192.9.200.43:57511<br>69478473..69478482/57550 [ACK,PSH] window = 2048<br>checksum = 16401 length = 10 |             |           |

The menu entries represent state changes or data elements to be traced; each entry is a toggle. Clicking on the toggle once will activate the trace of the particular element and will gray-over the entry; clicking a second time will deactivate the tracing and ungray the menu item. The following data elements/transitions may be displayed:

|             |                                                                                         |
|-------------|-----------------------------------------------------------------------------------------|
| Contents    | Displays a line's worth of packet contents. The Incoming or Outgoing switch must be on. |
| Incoming    | Displays incoming data.                                                                 |
| Outgoing    | Displays outgoing data.                                                                 |
| Checksums   | Displays checksums for each TCP segment.                                                |
| Time        | Displays the time interval since the last action on the connection.                     |
| Transitions | Displays state transitions on the TCP state machine.                                    |

(PPTCB *TCP FILE*)

[Function]

Prints the state of a TCP connection. PPTCB is normally the INFO function for the process that monitors a connection; thus, selecting INFO in the process status window will cause a window to pop up containing a report on the status of the associated connection.

(TCP.ECHOTEST *HOST NLINES*)

[Function]

Opens a TCP connection to the TCP echo port on *HOST* and sends *NLINES* of random text. The echo responses are displayed in a window. If *NLINES* is NIL, the echo test will run forever.

(TCP.ECHO.SERVER *PORT*)

[Function]

Starts a TCP echo server on *PORT* (defaults to the TCP echo port). It is usually more useful to start the echo server as a process by doing (ADD.PROCESS ' (TCP.ECHO.SERVER *PORT*) ).

(TCP.SINK.SERVER *PORT*)

[Function]

Starts a TCP sink server on *PORT* (defaults to the TCP sink port). Any data sent to this port will be acknowledged and discarded. As with the TCP echo server, it is usually more useful to start this server as an independent process.

(TCP.FAUCET *HOST PORT NLINES*)

[Function]

If *HOST* is non-NIL, this function opens a connection to *PORT* on *HOST* and sends *NLINES* of text (the default is to send lines of text forever). *PORT* defaults to the TCP sink port. If *HOST* is NIL, this function waits for a remote system to connect to the TCP faucet port and then sends out *NLINES* of random text.

## TCPFTP

TCPFTP implements a virtual I/O device that performs Lisp filing operations transparently using the RFC765 FTP protocol. The standard filing operations of reading, writing, renaming, deleting, and directory enumeration are supported by the TCPFTP device. However, neither random access filing nor GETFILEINFO are supported, as there is no protocol specification for performing these operations on files. Interlisp operations such as RECOMPILE will not work when files are stored on TCPFTP file servers.

Once TCPFTP is loaded, filing operations should be transparent to users; no additional initialization need be performed. There are, however, three important global variables:

TCPFTP.EOL.CONVENTION

[Variable]

This variable controls the end-of-line convention used with files accessed via TCP. Generally, you should set it to match the convention on the system where the files reside. The value can be one of those shown below.

(TCPFTP.EOL.CONVENTION *TYPE*)

[Function]

Sets the variable TCP.EOL.CONVENTION to *TYPE*. *TYPE* can be one of the following:

|       |                                     |
|-------|-------------------------------------|
| CR    | Set EOL to CR                       |
| LF    | Set EOL to LF                       |
| CRLF  | Set EOL to CRLF                     |
| OS    | Set it to something based on the OS |
| Other | Set it to the default (CRLF)        |

TCPFTP.DEFAULT.FILETYPES

[Variable]

This variable is an association list, keyed by common extensions of file names, and contains appropriate file types (for example, TEXT or BINARY) for such files. The TCPFTP protocol provides no mechanism for determining the type of a file about to be retrieved. The file type is usually known in the case of output operations (for example, COPYFILE or MAKEFILE to a file server). However, in the case of COPYFILE from a file server, the TCPFTP module has to infer the file type from other knowledge. The module tries to match the extension of the file name with an entry on the list TCPFTP.DEFAULT.FILETYPES. If it finds a match, it uses the value of the entry in the list as the file type of the file; if it doesn't find a match, it uses the value of TCP.DEFAULTFILETYPE for the file type of the file.

TCP.DEFAULTFILETYPE

[Variable]

If no matching extension is found for the file being opened, the TCPFTP module uses the value of TCP.DEFAULTFILETYPE as the file type of the remote file. The initial value of TCP.DEFAULTFILETYPE is BINARY.



The following functions are available for debugging broken file server connections.

(FTPDEBUG *FLG*) [Function]

If *FLG* is T, this function opens a scrolling trace window that displays FTP commands as they are issued. PUPFTP commands will also be displayed in this window (the window is the value of FTPDEBUGLOG).

(\TCP.BYE *HOST*) [Function]

Breaks an FTP connection to *HOST*.

(\TCPFTP.INIT) [Function]

(Re)initializes the TCPFTP module.

## TCPFTPSRV

The TCPFTPSRV module contains a program which implements an FTP service for Interlisp. When this program is running on a workstation, other hosts are able to store and retrieve files from the workstation.

(TCPFTP.SERVER PORT DEFAULT.FILE.PATH) [Function]

To start the server program, evaluate the form (TCPFTP.SERVER). If *PORT* is supplied, the FTP server program will listen for connections on the TCP port specified by *PORT*; otherwise, the server will listen on the default FTP server port, port 21.

If *DEFAULT.FILE.PATH* is supplied, the initial path for resolving file names will be relative to *DEFAULT.FILE.PATH*; the default value of this variable is {DSK}<LISPFILES>.

TCPFTP.SERVER.USE.TOPS20.SYNTAX [Variable]

This variable controls whether file names sent back to FTP client programs are formatted in Tops-20 or Interlisp syntax. If the variable is true (the default), all file names will be formatted in Tops-20 syntax. This permits an Interlisp workstation to masquerade as a Tops-20 mainframe for the purposes of file transfer to and from other vendors' machines.

## TCPNAMES

The TCPNAMES module provides a set of functions for translating among the file-naming conventions of different operating systems. This is needed by the TCPFTP module in order for it to convert between Interlisp format file names and the file name formats of other operating systems.

(REPACKFILENAME.STRING NAME FOROSTYPE) [Function]

*NAME* is a file name in some operating system's format. *FOROSTYPE* is the name of an operating system. REPACKFILENAME.STRING attempts to translate *NAME* into a format acceptable to the operating system named by *FOROSTYPE*. *NAME* may be a string or atom; the function always returns a string.

Currently acceptable operating system types are:

IFS  
INTERLISP  
MS-DOS  
SYMBOLICS-3600  
TENEX  
TOPS-20 (also TOPS20)  
UNIX  
VMS

(TI-Explorers should use TOPS-20 as their operating system.)

The correspondence between the target operating system type and the file name translation function is maintained in an extensible hash table.

(\REPACKFILENAME.NEW.TRANSLATION *OSTYPE FUNCTION*) [Function]

This function adds a new file name translation function for a new operating system type. The function must be a LAMBDA-NOSPREAD function, and must be prepared to receive either a single property-list format argument, such as would be returned by UNPACKFILENAME, or an arbitrary number of arguments in property-list format.

File names in the above format will be passed to the translation function adhering to the conventions of many operating systems; the function must recognize the operating system type and produce the desired output format, which must be a string.

\REPACKFILENAME.OSTYPE.TABLE [Variable]

This variable is the hash table that stores the correspondence between operating system types and translation functions.

## TCPCHAT

TCPCHAT implements the TELNET protocol for virtual terminal I/O between Interlisp and a remote system. Once loaded into Interlisp, the standard Chat system will use TCP TELNET to communicate with hosts that are believed to support the protocol.

No user-callable functions reside in this module, although the following variables may be of interest.

TCPCHAT.TELNET.TTY.TYPES [Variable]

This variable is an association list that maps internal names of Chat terminal emulators to official terminal names as specified in RFC884, the TELNET Terminal Type Option. This allows TCPCHAT to set the user's terminal type automatically when a connection is established.

TCPCHAT.TRACEFLG [Variable]

If this variable is non-NIL, TELNET negotiations will be printed to TCPCHAT.TRACEFILE (see below). This is sometimes useful in debugging negotiation problems.

TCPCHAT.TRACEFILE [Variable]

TELNET negotiations are printed to this file if TCPCHAT.TRACEFLG is non-NIL.

## TCPUDP

UDP implements the user datagram protocol. The following functions are meant to be called by client applications.

(UDP.INIT) [Function]

Initializes the UDP module. This function is normally called when UDP is loaded and should not need to be called again under normal circumstances.

(UDP.STOP) [Function]

Disables the UDP module, closing any open UDP sockets.

(UDP.OPEN.SOCKET *SKT# IFCLASH*) [Function]

Opens a socket for UDP operations.

*SKT#*, if supplied, is a 16-bit number and will default to a number between 1,000 and 65,535.

*IFCLASH* specifies what to do if the requested socket is already open and is handled as in *OPENPUPSOCKET* and *OPENNSOCKET* (see the *IRM*).

It returns an instance of an *IPSOCKET*.

(UDP.CLOSE.SOCKET *IPSOCKET NOERRORFLG*) [Function]

Closes an open *IPSOCKET*. If *IPSOCKET* is not an open socket and *NOERRORFLG* is *NIL*, an error will occur; otherwise, *NIL* is returned if the socket is not active, and *T* is returned if the socket is active.

Any remaining packets on the socket's input queue are discarded when this function is called.

(UDP.SOCKET.EVENT *IPSOCKET*) [Function]

Returns an event that a process may use to wait for packet arrival on *IPSOCKET*.

(UDP.SOCKET.NUMBER *IPSOCKET*) [Function]

Returns the socket number of *IPSOCKET*.

(UDP.GET *IPSOCKET WAIT*) [Function]

Returns the next packet waiting on *IPSOCKET*. If no packets are waiting, does one of the following based on the value of *WAIT*.

|            |                                                                                                                                  |
|------------|----------------------------------------------------------------------------------------------------------------------------------|
| <i>NIL</i> | Returns immediately.                                                                                                             |
| <i>T</i>   | Waits forever for a packet to arrive.                                                                                            |
| a number   | <i>FIXP</i> waits up to <i>WAIT</i> milliseconds for a packet to arrive and returns <i>NIL</i> if none arrived during that time. |

Thus, this function is like *GETPUP* and *GETXIP*.

(UDP.SETUP UDP *DESTHOST DESTSOCKET ID IPSOCKET REQUEUE*) [Function]

Initializes a fresh packet (as returned from *\ALLOCATE.ETHERPACKET*). The packet will be sent to *DESTSOCKET* on *DESTHOST*.

*ID* is a number to be placed in the IP header ID field (zero is fine).

*REQUEUE* specifies what to do with the packet after it is sent; *NIL* (the default) means no special treatment; *FREE* means to release the packet and return it to the free packet queue. Any instance of a *SYSQUEUE* will cause the packet to be queued on the tail of the specified queue.

*UDP.SETUP* initializes all IP and UDP fields and sets the packet up as a minimum-length UDP packet.

(*UDP.SEND IPSOCKET UDP*) [Function]

Sends *UDP*, a UDP-formatted packet, out from *IPSOCKET*.

(*UDP.EXCHANGE IPSOCKET OUTUDP TIMEOUT*) [Function]

Sends *OUTUDP* out from *IPSOCKET* and waits *TIMEOUT* milliseconds for a response; returns *NIL* if no response came in during the specified interval, or the packet that did come in during that time.

Clears the socket's input packet queue before waiting for a packet to arrive.

(*UDP.APPEND.BYTE UDP BYTE*) [Function]

Appends *BYTE* to the UDP data portion of *UDP* and increments the UDP and IP length fields by one.

(*UDP.APPEND.WORD UDP WORD*) [Function]

Appends *WORD* to the UDP data portion of *UDP* and increments the UDP and IP length fields by two.

(*UDP.APPEND.CELL UDP CELL*) [Function]

Appends *CELL* to the UDP data portion of *UDP* and increments the UDP and IP length fields by four.

(*UDP.APPEND.STRING UDP STRING*) [Function]

Appends *STRING* to the UDP data portion of *UDP* and increments the UDP and IP length fields by the length *STRING*.

## TCPTFTP

TFTP implements the trivial file transfer protocol. This protocol is useful for transferring unimportant files rapidly (for example, between workstations and printers). The following user-callable functions exist.

(*TFTP.PUT FROM TO PARAMETERS*) [Function]

Sends a file to a TFTP host.

*FROM* may refer to any accessible file; *TO* must refer to a file accessible via TFTP.

No attempt is currently made to translate between Interlisp file name syntax and remote system file name syntax for *TO*.

For example, if *TO* resides on a UNIX host, it would take a syntax like *{HOST}/DIRECTORY/SUBDIRECTORY/FILENAME*.

*PARAMETERS* is currently a list of parameters in the same format used by *OPENFILE* in *.PARAMETERS*; for example `((EOLCONVENTION 1) (TYPE TEXT))`.

Note: TFTP transfers between Xerox Lisp and UNIX hosts initiated from Xerox Lisp should have the *PARAMETERS* argument be `'((EOLCONVENTION 10))`.

`(TFTP.GET FROM TO PARAMETERS)` [Function]

Gets a file from a TFTP host. *FROM* must be a file accessible by TFTP; *TO* may be any file.

The file name syntax caveats for *FROM* are the same as for *TO* in *TFTP.PUT*. *PARAMETERS* is also as in *TFTP.PUT*.

`(TFTP.SERVER LOGSTREAM)` [Function]

Starts a TFTP server process.

*LOGSTREAM* may be left *NIL*, causing a new window to appear when the TFTP server is first invoked. Remote systems that support TFTP clients may store or retrieve files through any Interlisp workstation running the TFTP server.

The full Interlisp syntax for file names is supported; thus, requests to store files whose names include hosts will result in the Interlisp workstation's transparently storing the files on the designated hosts.

`(\TFTP.OPENFILE FILENAME ACCESS RECOG PARAMETERS)` [Function]

Returns a *STREAM* to open for *ACCESS* on *FILENAME*.

*PARAMETERS* is the usual format; *TYPE* is the only recognized parameter (*BINARY* opens a stream in *octet* format; *TEXT*, the default, opens a stream in *NETASCII* format; see RFC783).

*BIN*, *BOUT*, *READP*, *EOFP*, etc., may be used on this stream.

The stream is not *RANDACCESSP*.

`(\TFTP.CLOSEFILE STREAM)` [Function]

Closes the open stream. This is normally useful for streams open for *OUTPUT*; for *INPUT* streams, end-of-file will occur eventually.

## TCPLLIB

For users planning implementations on top of IP, the following low-level TCP functions are available.

### IP Socket Access

`(\IPINIT)` [Function]

Reinitializes the IP world; for example, after some catastrophe.

`(\STOPIP)` [Function]

Disables IP.

(DODIP.HOSTP *NAME*)

[Function]

If *NAME* is an integer, *NAME* is returned unaltered. If *NAME* is a text format IP host address (such as 192.10.200.1), DODIP.HOSTP returns its integer representation.

If *NAME* is a string or atom name, DODIP.HOSTP attempts to convert *NAME* to its IP host address integer value, using information supplied in the HOSTS.TXT file (see TCPHTE, below), followed by doing a domain query if TCPDOMAIN is loaded.

If *NAME* is unknown, DODIP.HOSTP returns NIL.

If *NAME* is known, it is cached with its corresponding address so that the function IPHOSTNAME may be used later to convert the address back to a name.

(IPHOSTNAME *IPADDRESS*)

[Function]

Tries to convert *IPADDRESS* to a host name.

If *IPADDRESS* has no known name, it is converted to the text representation of an IP address (for example, 192.10.200.1).

(IPTRACE *MODE*)

[Function]

Turns on tracing of IP activity. This function is like PUPTRACE and XIPTRACE, which are documented in the IRM.

If *MODE* is NIL, IP tracing is disabled.

If *MODE* is T, verbose IP tracing is enabled.

If *MODE* is PEEK, concise IP tracing is enabled. If *MODE* is either T or PEEK, the user is prompted for a window into which trace output will be printed.

(\IP.ADD.PROTOCOL *PROTOCOL SOCKETCOMPAREFN NOSOCKETFN INPUTFN ICMPFN*)

[Function]

Defines a new IP-based protocol. The lowest-level IP functions maintain a list of active protocols and perform packet delivery based on the existence of open sockets for protocols of received packet types.

*PROTOCOL* is a protocol number, a number between 1 and 255. The following protocols are defined and should not be disturbed:

|      |    |
|------|----|
| TCP  | 6  |
| ICMP | 1  |
| UDP  | 17 |

*SOCKETCOMPAREFN* is a function with two arguments, an IP packet that has just been received and an open IPSOCKET. This function should return NIL if the packet does not belong to the supplied socket, or T if it does. The function will typically be interested in the IPSOCKET field of the IPSOCKET.

*NOSOCKETFN* is a function with one argument, an IP packet that has just been received. Its purpose is to handle received packets for which no socket can be found. If *NOSOCKETFN* is NIL, the default function, \IP.DEFAULT.NOSOCKETFN, will be used; this function simply returns an ICMP message indicating the socket is unreachable.

*INPUTFN* is a function with two arguments, a received IP packet and an open IPSOCKET. The *INPUTFN* is supposed to handle reception of packets when

their destination socket has been found. If *INPUTFN* is *NIL*, the default function, `\IP.DEFAULT.INPUTFN`, will be supplied.

*INPUTFN* enqueues the received packet on the *IPSQUEUE* field of the *IPSOCKET* if the current queue length (stored in the *IPSQUEUELENGTH* field) is less than the allocated length (stored in the *IPSQUEUEALLOC* field).

*INPUTFN* also increments the *IPSQUEUELENGTH* field, and notifies the event stored in the *IPSEVENT* field.

*ICMPFN* is a function with two arguments and is called when an ICMP packet referring to the protocol is received. The first argument is a pointer to the received ICMP packet. The second argument is a pointer that may be used as if pointed to the original outgoing packet included in the ICMP data. This allows the protocol functions to parse the data in the ICMP packet to determine which socket sent the offending packet. The *ICMPFN* must never attempt to deallocate the packet identified by the second argument; however, it is quite permissible (and expected) that the *ICMPFN* will release the packet identified by the first argument. The default *ICMPFN* simply releases the packet identified by the first argument.

`\IP.ADD.PROTOCOL` returns an *IPSOCKET* datum, which represents the active protocol; it is not in fact a useful *IPSOCKET* and may be safely ignored.

`(\IP.DELETE.PROTOCOL PROTOCOL)` [Function]

Deactivates a protocol with protocol number *PROTOCOL*. Any open sockets are closed.

`(\IP.OPEN.SOCKET PROTOCOL SOCKET NOERRORFLG SOCKETCOMPAREFN NOSOCKETFN INPUTFN)` [Function]

Attempts to open an *IPSOCKET* for protocol *PROTOCOL*.

*SOCKET* is the identifying information for this socket; this quantity will be *EQUAL*-compared with other sockets open on *PROTOCOL*. Should a match be found, an error will occur unless *NOERRORFLG* is *T*, in which case the existing socket will be returned.

*SOCKETCOMPAREFN*, *NOSOCKETFN*, and *INPUTFN* may be supplied to override the functions specified when the protocol was defined; they are not normally useful, however.

`(\IP.CLOSE.SOCKET SOCKET PROTOCOL NOERRORFLG)` [Function]

Closes a socket open on *PROTOCOL*. *SOCKET* is the same quantity passed to `\IP.OPEN.SOCKET`; it is currently not an instance of an *IPSOCKET*. If *NOERRORFLG* is *T*, an error will not occur if the socket is not found.

## IP Packet Building

The following functions are useful for placing bytes into IP packets (as allocated by `\ALLOCATE.ETHERPACKET`).

Most applications will probably want to define a block record to overlay the data portion of an IP packet. Here is an example of such a block record.

Note: Users who are developing new IP-based protocols will need to load `EXPORTS.ALL` from the library.

```
(ACCESSFNS UDP
 ((UDPBASE (\IPDATABASE DATUM)))
 (BLOCKRECORD UDPBASE
 ((UDPSOURCEPORT WORD)
 (UDPDESTPORT WORD)
 (UDPLENGTH WORD)
 (UDPCHECKSUM WORD)))
 (ACCESSFNS UDP
 ((UDPCONTENTS
 (\ADDBASE
 (\IPDATABASE DATUM)
 (FOLDHI \UDPOVLEN BYTESPERWORD)))))))
```

(\IP.APPEND.BYTE *IP BYTE INHEADER*) [Function]

Appends *BYTE* to the IP data portion of *IP* and increments the IP length field by one. If *INHEADER* is *T*, the *IPHEADERLENGTH* field is appropriately incremented so that the bytes appear to have been appended to the options portion of the IP header. There must not be any data bytes in the data portion of the packet if this function is to work correctly.

(\IP.APPEND.WORD *IP WORD INHEADER*) [Function]

Appends *WORD* to the IP data portion of *IP* and increments the IP length field by two. *INHEADER* is as in \IP.APPEND.BYTE.

(\IP.APPEND.CELL *IP CELL INHEADER*) [Function]

Appends *CELL* to the IP data portion of *IP* and increments the IP length field by four. *INHEADER* is as in \IP.APPEND.BYTE.

(\IP.APPEND.STRING *IP STRING*) [Function]

Appends *STRING* to the IP data portion of *IP* and increments the IP length field by the length *STRING*.

## IP Packet Sending

(\IP.SETUPIP *IP DESTHOST ID SOCKET REQUEUE*) [Function]

Initializes *IP*. This function should be called just after *IP* is obtained from \ALLOCATE.ETHERPACKET; if this is not done, the append functions above will fail.

*DESTHOST* is the 32-bit IP address to which this packet will be sent.

*ID* is an arbitrary 16-bit quantity that will become the IPID field of the packet.

*SOCKET* is the open IPSOCKET from which the packet will be sent.

*REQUEUE* defaults to *FREE* and controls the disposition of the packet after transmission (see the *IRM* for the documentation of *SETUPPUP* or *FILLINXIP*).

(\IP.TRANSMIT *IP*) [Function]

Tries to send *IP*. Performs IP checksum algorithm prior to sending. Returns *NIL* if successful, otherwise it returns a status indication, such as *NoRouting* or



`AlreadyQueued`. This function is like `SENDPUP` and `SENDXIP`, except that no socket argument is required.

## TCPHTE

HTE provides functions for parsing `HOSTS.TXT` files as documented by RFC810. This file is loaded automatically by LLIP and is used by `\IPINIT` to read in the initial file, `HOSTS.TXT`. The following variable and function may be of interest.

`HOSTS.TEXT.DIRECTORIES` [Variable]

Is the search path for the file `HOSTS.TXT`. This variable is initialized to `NIL`; thus the search path to be used is by default `DIRECTORIES`.

`(\HTE.READ.FILE FILE WANTEDTYPES)` [Function]

Reads a `HOSTS.TXT` file.

`WANTEDTYPES` is a list of types drawn from the set `{HOST, NET, GATEWAY}`, to be read from the file; types not specified in `WANTEDTYPES` are ignored. `WANTEDTYPES` defaults to `(HOST)`.

## TCPDOMAIN

TCPDOMAIN enhances the host address lookup to do queries to a domain name server as specified by RFC883. This allows the system to only keep in memory the addresses of hosts that are actually communicated with, resulting in considerable savings of space on large networks. There is a slight delay in establishing communications with a host the first time. Since it is faster to initiate communications with hosts defined in the `HOSTS.TEXT` file, we recommend that frequently-communicated-with hosts be defined therein.

`(DOMAIN.INIT)` [Function]

(Re)initializes the TCPDOMAIN package.

`(DOMAIN.TRACE)` [Function]

Opens up a window for tracing domain queries. (The window is the value of `DOMAIN.TRACE.FILE`).

`(DOMAIN.LOOKUP NAME TYPE SERVER)` [Function]

Looks up a host named `NAME` and type `TYPE` from server `SERVER`. If `SERVER` is unspecified, uses the default server.

`(DOMAIN.LOOKUP ADDRESS NAME SERVER DONT.GET.OSTYPE)` [Function]

Looks up the address of host `NAME` from server `SERVER`. If `SERVER` is unspecified, uses the default server. If `DONT.GET.OSTYPE` is `T`, will not get the `OSTYPE` of the host.

`(DOMAIN.LOOKUP.NAMESERVER NAME SERVER)` [Function]

Looks up the server serving the domain of `NAME`. If `SERVER` is unspecified, chooses the "best" server based on its knowledge of the namespace.

`(DOMAIN.GRAPH WINDOW)` [Function]

Draws a graph of the namespace in window `WINDOW`. Opens a new one if `WINDOW` is unspecified.

## TCP Debugging Aids

---

With TCPDEBUG loaded use (TCPTRACE T) to open up a trace window of TCP traffic. The appropriate items need to be selected from the windows menu in order for data to be seen.

(SETQ TCPCHAT.TRACEFLG T) will print TELNET negotiations to a file, which is what the variable TCPCHAT.TRACEFILE points to.

(FTPDEBUG T) opens a scrolling trace window that displays FTP commands as they are issued. You will see unencrypted passwords if they are issued.

## Limitations

---

You must use the 1186 microcode in order for TCP-IP to work on an 1186 (microcode for the 1185 will not do).

TCP-IP will not work with UNIX systems that have trailer encapsulation enabled. Connections will hang and then eventually break.

Directory enumeration on a VMS system results in NIL.

## Known Problems in TCPFTPSRV

It does not handle error conditions in the middle of file transfers.

Doing a DIR gives you only filename and version: no author, creation date, etc. This is because the TCPFTP protocol specification doesn't support author, creation date, etc.

If there are multiple files on the system, deleting a file without specifying a specific version deletes the most recent version. The workaround is to give the specific version to delete.

The subdirectory structure is not presented back to the client host. If you have a file on both the <lispfiles> directory and a subdirectory, when you do a DIR \*.\* you do not see the subdirectory listed, but you do see that there are two files on the host with the same version number.

## References

---

Users with access to the ARPANET may retrieve any RFC from host SRI-NIC.ARPA with the file transfer protocol (FTP) anonymous log-in option. RFCs are stored under <RFC>RFCnnn.TXT, where nnn is replaced by the number of the particular RFC.

From points on the Xerox internet, the RFC files can be retrieved from {Indigo}<RFC>. {Indigo} is an IFS host. From Lisp, you can simply (LOGIN) and supply your GV credentials if you haven't already, open a FileBrowser on that directory, and retrieve the file to the local workstation environment.

The following RFCs are mentioned in this manual:

RFC765 (superseded by RFC 959)  
RFC768  
RFC783  
RFC791  
RFC792  
RFC793  
RFC810 (superseded by RFC 952)  
RFC814  
RFC821  
RFC822  
RFC826  
RFC854  
RFC894  
RFC895  
RFC903  
RFC904  
RFC940

[This page intentionally left blank]

## FREE MENU

---

Free Menus are powerful and flexible menus that are useful for applications that need menus with different types of items, including command items, state items, and editable items. A Free Menu lives in a window, which can be opened and closed as desired, or attached as a control menu to the application window.

---

### Making a Free Menu

---

A Free Menu is built from a description of the contents and layout of the menu. As a Free Menu is simply a group of items, a Free Menu Description is simply a specification of a group of items. Each group has properties associated with it, as does each Free Menu Item. These properties specify the format of the items in the group, and the behavior of each item. The function FREEMENU takes a Free Menu Description, and returns a closed window with the Free Menu in it.

Probably the easiest way to make a Free Menu is to define your own function which calls FREEMENU with the Free Menu Description right there in your function. This function can then also set up the Free Menu window as required by the application. The Free Menu Description is then saved as part of your function when you save your application.

Alternatively, you can save the Free Menu Description as a variable in your file, and then just call FREEMENU with the name of the variable. This may be a more difficult alternative if you want to use the backquote facility to build your Free Menu Description. See the section **Free Menu Item Descriptions**.

---

### Free Menu Formatting

---

A Free Menu can be formatted in one of four ways. The items in any group can be automatically layed out in rows, in columns, or in a table, or else the application can specify the exact location of each item in the group. Additionally, Free Menu keeps track of the region that a group of items occupies, and items can be justified within that region. This way an item can be automatically positioned at one of the nine justification locations, top-left, top-center, top-right, middle-left, etc.

---

### Free Menu Description

---

A Free Menu Description, specifying a group of items, is a list structure. The first thing in the list is an optional list of the properties of this group of items, in the form:

(PROPS <PROP> <VALUE> <PROP> <VALUE> ...)

The key word PROPS determines whether or not the optional group props list is specified. The section **Free Menu Group Properties** describes each group property. For now, the important property is FORMAT. The type of formatting determines the syntax of the rest of the Free Menu Description, in a very simple way.

When using EXPLICIT formatting, the rest of the description is any number of Item Descriptions, which have LEFT and BOTTOM properties specifying the position of the item in the menu. The syntax is:

```
((PROPS FORMAT EXPLICIT ...)
 <ITEM DESCRIPTION>
 <ITEM DESCRIPTION> ...)
```

When using ROW or TABLE formatting, the rest of the description is any number of item groups, each group corresponding to a row in the menu. These groups are identical in *syntax* to an EXPLICIT group description, with an optional PROPS list and then any number of Item Descriptions, except that the items need not have LEFT and BOTTOM properties, as the location of each item is figured out by the formatter. But the order of the rows and items is important. The menu is layed out top to bottom by row, and left to right within each row. The syntax is (the comments are not part of the description):

```
((PROPS FORMAT ROW ...) ; props of this group
 (<ITEM DESCRIPTION> ; items in first row
 <ITEM DESCRIPTION> ...)
 ((PROPS ...) ; props of second row
 <ITEM DESCRIPTION> ; items in second row
 <ITEM DESCRIPTION> ...))
```

When using COLUMN formatting, the syntax is identical to that of ROW formatting. However each group of items corresponds to a column in the menu, rather than a row. The menu is layed out left to right by column, top to bottom within each column.

Finally, a Free Menu Description can have recursively nested groups. Anywhere the description can take an Item Description, it can take a group, marked by the key word GROUP. A nested group inherits all of the properties of its mother group, by default. However, any of these properties can be overridden in the nested groups PROPS list, including the FORMAT. The syntax is:

```
(; no PROPS list, default row format
 (<ITEM DESCRIPTION> ; first in row
 (GROUP ; nested group, second in row
 (PROPS FORMAT COLUMN...) ; optional props
 (<ITEM DESCRIPTION> ...) ; first column
 (<ITEM DESCRIPTION> ...))
 <ITEM DESCRIPTION>)) ; third in row
```

Here is an example of a simple Free Menu Description, for a menu which might provide access to a simple data base:

```
(((LABEL LOOKUP SELECTEDFN MYLOOKUPFN) (LABEL EXIT SELECTEDFN MYEXITFN))
 ((LABEL Name: TYPE DISPLAY) (LABEL "" TYPE EDIT ID NAME))
 ((LABEL Address: TYPE DISPLAY) (LABEL "" TYPE EDIT ID ADDRESS))
 ((LABEL Phone: TYPE DISPLAY)
 (LABEL "" TYPE EDIT LIMITCHARS MYPHONEP ID PHONE)))
```

This menu has two command buttons, LOOKUP and EXIT, and three edit fields, with ID's NAME, PHONE, and ADDRESS. The Edit items are initialized to the empty string, as in this example they need no other initial value. The user could click after the Name: prompt, type a person's name, and then press the LOOKUP button. This would cause the function MYLOOKUPFN to be called, which could look at the NAME Edit item, lookup that name in the data base, and then fill in the rest of the fields appropriately. Note that the PHONE item has MYPHONEP as a LIMITCHARS function. This function would be called when editing the phone number, in order to restrict input to a valid phone number. After looking up Perry, the Free Menu might look like:

```
LOOKUP EXIT
Name: Herbert Q Perry
Address: 13 Middleperry Dr
Phone: (411) 767-1234
```

Here is a more complicated example:

```
(((PROPS FONT (MODERN 10))
 ((LABEL Example FONT (MODERN 10 BOLD) HJUSTIFY CENTER))
 ((LABEL NORTH) (LABEL SOUTH) (LABEL EAST) (LABEL WEST))
 ((PROPS ID ROW3 BOX 1)
 (LABEL ONE) (LABEL TWO) (LABEL THREE))
 ((PROPS ID ROW4)
 (LABEL ONE ID ALPHA)
 (GROUP (PROPS FORMAT COLUMN BACKGROUND 23130 BOX 2 BOXSPACE 4)
 ((TYPE NWAY LABEL A BOX 1 COLLECTION COL1 NWAYPROPS (DESELECT T))
 (TYPE NWAY LABEL B BOX 1 COLLECTION COL1)
 (TYPE NWAY LABEL C BOX 1 COLLECTION COL1))
 ((TYPE STATE LABEL "Choose Me" BOX 1 MENUITEMS (BRAVO DELTA)
 INITSTATE DELTA LINKS (DISPLAY (GROUP ALPHA)))
 (TYPE DISPLAY ID ALPHA LABEL "" BOX 1 MAXWIDTH 35)))
 (LABEL THREE)))
```

which will produce the following Free Menu:

```
Example
NORTH SOUTH EAST WEST
ONE TWO THREE
A Choose Me
B DELTA
C
ONE THREE
```



And if the Free Menu were formatted as a Table, instead of in Rows, it would look like:

| Example |                                                                                           |       |      |
|---------|-------------------------------------------------------------------------------------------|-------|------|
| NORTH   | SOUTH                                                                                     | EAST  | WEST |
| ONE     | TWO                                                                                       | THREE |      |
|         | <div> <div>A</div> <div>Choose Me</div> <div>B</div> <div>DELTA</div> <div>C</div> </div> |       |      |
| ONE     |                                                                                           | THREE |      |

## Free Menu Group Properties

Each group has properties. Most group properties are relevant, and should be set, in the group's PROPS list in the Free Menu Description. User properties can freely be included in the PROPS list. A few other properties are setup by the formatter. After the Free Menu is created, group properties can be accessed by the macro FM.GROUPPROP or FM.MENUPROP.

|             |                                                                                                                                                                                                                                                                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ID          | The identifier of this group. Setting the group ID is desirable, for example, if the application needs to get handles on items in particular groups, or access group properties.                                                                                                                                                                      |
| FORMAT      | One of ROW, COLUMN, TABLE, or EXPLICIT. The default is ROW.                                                                                                                                                                                                                                                                                           |
| FONT        | A font description of the form (FAMILY SIZE FACE), or a FONTDESCRIPTOR data type. This will be the default font for each item in this group. The default font of the top group is the value of the variable DEFAULTFONT.                                                                                                                              |
| COORDINATES | One of GROUP, or MENU. This property applies only to Explicit formatting. If GROUP, then the items in the explicit group are positioned in coordinates relative to the lower left corner of the group, as determined by the mother group. If MENU, which is the default, then the items are positioned relative to the lower left corner of the menu. |
| LEFT        | Specifies a left offset for this group, pushing the group to the right.                                                                                                                                                                                                                                                                               |
| BOTTOM      | Specifies a bottom offset for this group, pushing the group up.                                                                                                                                                                                                                                                                                       |
| ROWSPACE    | The number of bits between rows in this group.                                                                                                                                                                                                                                                                                                        |
| COLUMNSPACE | The number of bits between columns in this group.                                                                                                                                                                                                                                                                                                     |
| BOX         | The number of bits in the box around this group of items.                                                                                                                                                                                                                                                                                             |
| BOXSHADE    | The shade of the box.                                                                                                                                                                                                                                                                                                                                 |
| BOXSPACE    | The number of bits between the box and the items.                                                                                                                                                                                                                                                                                                     |
| BACKGROUND  | The background shade of this group. Nested groups will inherit this background shade, but items in this group and nested groups will not. This is because in general it is difficult to read text on a                                                                                                                                                |

background, so items appear on white background by default. This can be overridden by the BACKGROUND Item Property.

---

## Other Group Properties

---

|           |                                                                                                                                           |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------|
|           | The following group properties are setup and maintained by Free Menu. The application should probably not change any of these properties. |
| ITEMS     | A list of the items in the group.                                                                                                         |
| REGION    | The region that is the extent of the items in the group.                                                                                  |
| MOTHER    | The ID of the group that is the mother of this group.                                                                                     |
| DAUGHTERS | A list of ID of groups which are daughters to this group.                                                                                 |

---

## Free Menu Items

---

Each Free Menu Item is stored as an instance of the Data Type FREEMENUITEM. Free Menu Items can be thought of as objects, each item having its own particular properties, such as its type, label, and mouse event functions. A number of useful item types, described in the section **Free Menu Item Types**, are predefined by Free Menu. New types of items can be defined by the application, using Display items as a base.

Each Free Menu Item is created from a Free Menu Item Description when the Free Menu is created.

---

## Free Menu Item Descriptions

---

A Free Menu Item Description is a list in property list format, specifying the properties of the item. For example:

```
(LABEL Refetch SELECTEDFN MY.REFETCHFN)
```

describes a command (Momentary) item labelled 'Refetch', with the function MY.REFETCHFN to be called when the item is selected.

None of the property values in an item description are evaluated. When constructing Free Menu descriptions that incorporate evaluated expressions, for example labels that are bitmaps, it is helpful to use the backquote facility. For example, if the value of the variable MYBITMAP is a bitmap, then

```
(FREEMENU `(((LABEL A) (LABEL ,MYBITMAP))))
```

would create a Free Menu of one row, with two items in that row, the second of which has the value of MYBITMAP as its label.

---

## Free Menu Item Properties

---

|      |                                                                                                                                                                                                                           |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      | The following Free Menu Item Properties can be set in the Item Description. Any other properties given in an Item Description will be treated as user properties, and will be saved on the USERDATA property of the item. |
| TYPE | The type of the item. Choose from one of the Free Menu Item type keywords MOMENTARY, TOGGLE, 3STATE, STATE, NWAY, EDITSTART, EDIT, NUMBER, or DISPLAY. The default is MOMENTARY.                                          |

|                 |                                                                                                                                                                                                                                                                                                                                      |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LABEL           | An atom, string, or bit map. Bit maps are always copied, so that the original won't be changed. This property must be specified for every item.                                                                                                                                                                                      |
| FONT            | The font that the item will appear in. The default is the font specified for the group that this item is in. Can be a font description of the form (FAMILY SIZE FACE), or a FONTDESCRIPTOR data type.                                                                                                                                |
| ID              | May be used to specify a unique identifier for this item, but is not necessary.                                                                                                                                                                                                                                                      |
| LEFT and BOTTOM | When Row, Column, or Table formatting, these specify offsets, pushing the item right and up, respectively, from where the formatter would have put the item. In Explicit formatting, these are the actual coordinates of the item, in the coordinate system given by the group's COORDINATES property.                               |
| HJUSTIFY        | One of LEFT, CENTER, or RIGHT. Specifies that this item is to be horizontally justified within the extent of its group. Note that the main group, as opposed to the smaller row or column group, is used.                                                                                                                            |
| VJUSTIFY        | One of TOP, MIDDLE, or BOTTOM. Specifies that this item is to be vertically justified.                                                                                                                                                                                                                                               |
| HIGHLIGHT       | Specifies the highlighted looks of the item, that is, how the item changes when a mouse event occurs on it. See the section <b>Free Menu Item Highlighting</b> , below.                                                                                                                                                              |
| MESSAGE         | A string that will be printed in the prompt window after a mouse button is held down over this item for MENUHELDWAIT milliseconds. Or, if an atom, treat as a function to get the message. The function is applied to ITEM WINDOW BUTTONS, and should return a string. The default is a message appropriate to the type of the item. |
| INITSTATE       | The initial state of the item. This is only appropriate to TOGGLE, 3STATE, and STATE items.                                                                                                                                                                                                                                          |
| MAXWIDTH        | The width allowed for this item. The formatter will leave enough space after the item for the item to grow to this width without collisions.                                                                                                                                                                                         |
| MAXHEIGHT       | Similar to MAXWIDTH, but in the vertical dimension.                                                                                                                                                                                                                                                                                  |
| BOX             | The number of bits in the box around this item. Boxes are made around MAXWIDTH and MAXHEIGHT dimensions. If unspecified, no box is drawn.                                                                                                                                                                                            |
| BOXSHADE        | The shade that the box is drawn in. The default is BLACKSHADE.                                                                                                                                                                                                                                                                       |
| BOXSPACE        | The number of bits between the box and the label. The default is one bit.                                                                                                                                                                                                                                                            |
| BACKGROUND      | The background shade on which the item appears. The default is WHITESHADE, regardless of the group's background.                                                                                                                                                                                                                     |
| LINKS           | Can be used to link this item to other items in the Free Menu. See the section <b>Free Menu Item Links</b> .                                                                                                                                                                                                                         |

### Mouse Properties

The following properties provide a way for application functions to be called under certain mouse events. These functions are called with the ITEM, the WINDOW, and the BUTTONS depressed as

arguments. These application functions do not interfere with any Free Menu system functions that take care of handling the different item types. In each case, though, the application function is called *after* the system function. The default for all of these functions is NIL. The value of each of the following properties can be the name of a function, or a lambda expression.

|            |                                                                                                                                           |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| SELECTEDFN | The function to be called when this item is selected. Note that Edit and EditStart items cannot have a selectedfn. See Edit items, below. |
| DOWNFN     | The function to be called when a mouse button goes down over this item, or when the mouse moves over the item with buttons depressed.     |
| HELDFN     | The function to be called repeatedly while the mouse is held down over this item.                                                         |
| MOVEDFN    | The function to be called when the mouse moves off this item with buttons still depressed.                                                |

## System Properties

---

The following Free Menu Item properties are set and maintained by Free Menu. The application should probably not change these properties directly.

|                                          |                                                                                                                                                                                                                                           |
|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GROUPID                                  | The ID of the smallest group that the item is in. For example, in a row formatted group, the item's GROUPID will be set to the ID of the row that the item is in, not the ID of the whole group.                                          |
| STATE                                    | The current state of TOGGLE, 3STATE, or STATE items. The state of an NWAY item behaves like that of a toggle item.                                                                                                                        |
| BITMAP                                   | The bitmap from which the item is displayed.                                                                                                                                                                                              |
| REGION                                   | The region of the item, in window coordinates. This is used for locating the display position, as well as determining the mouse sensitive region of the item.                                                                             |
| MAXREGION                                | The maximum region the item may occupy, determined by the MAXWIDTH and MAXHEIGHT properties. This is used by the formatter and the display routines.                                                                                      |
| SYSDOWNFN<br>SYSMOVEDFN<br>SYSSELECTEDFN | These are the system mouse event functions, setup by Free Menu according to the type of the item. These functions are called before the users mouse event functions, and are used to implement highlighting, state changes, editing, etc. |
| USERDATA                                 | Any other properties are stored on this list in property list format. This list should probably not need to be manipulated directly.                                                                                                      |

## Predefined Item Types

---

### Momentary

---

Momentary items are like command buttons. When the button is selected, its associated function is called.

### Toggle

---

Toggle items are simple two-state buttons. When depressed the button is highlighted, and it stays that way until pressed again. The states of a toggle button are T and NIL, initially NIL.

---

### 3State

---

3State items rotate through NIL, T, and OFF, states each time they are pressed. The default looks of the OFF state are with a diagonal line through the button, while T is highlighted, and NIL is normal. The default initial state is NIL.

The following Item Property applies to 3State items:

|     |                                                                                                                                              |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------|
| OFF | Specifies the looks of a 3STATE item in its OFF state. Similar to HIGHLIGHT. The default is that the label gets a diagonal slash through it. |
|-----|----------------------------------------------------------------------------------------------------------------------------------------------|

### State

---

State items are general multiple state items. The following Item Property determines how the item changes state:

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| CHANGESTATE | This Item Property can be changed at any time to change the effect of the item. If a MENU datatype, then this menu is popped up when the item is selected, and the user can select the new state. Otherwise, if this property is given, it is treated as a function name, which is applied to ITEM WINDOW BUTTONS. This function can do whatever it wants, and is expected to return the new state (an atom, string, or bitmap), or NIL, meaning don't change state. |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

The state of the item can automatically be indicated in the Free Menu, by setting up a DISPLAY link to a Display item in the menu (see **Free Menu Item Links** below). If such a link exists, the label of the DISPLAY item will be changed to the new state. Note that the possible states are not restricted at all, except that if a popup menu is used, of course the possible selections are restricted. The state can be changed to any atom, string, or bitmap, manually via FM.CHANGESTATE.

The following Item Properties are relevant to State items when building a Free Menu:

|           |                                                                                                                                                                |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MENUITEMS | If specified, should be a list of item to go in a popup menu for this item. Free Menu will build the menu and save it as the CHANGESTATE property of the item. |
| MENUFONT  | The font of the items in the popup menu.                                                                                                                       |
| MENUTITLE | The title of the popup menu. The default title is the label of the State item.                                                                                 |

### Nway

---

NWay items provide a way to collect any number of items together, in any format within the Free Menu. Only one item from each Collection can be selected at a time, and that item is highlighted to indicate so.

The following Item Properties are particular to NWay items:

|            |                                                                                                                                                                     |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| COLLECTION | An identifier that specifies which NWay Collection this item belongs to.                                                                                            |
| NWAYPROPS  | A property list of information to be associated with this collection. This property is only noticed in the Free Menu Description on the first item in a Collection. |

NWay Collections are formed by creating a number of NWay items with the same COLLECTION property. Each NWay item acts individually as a Toggle item, and can have its own mouse event functions.

Each NWay Collection itself has properties, its state for instance. After the Free Menu is created, these Collection properties can be

accessed by the macro FM.NWAYPROPS. Note that NWay Collections are different from Free Menu Groups.

There are three NWay Collection properties that Free Menu looks at:

|           |                                                                                                                                                                                                                                                           |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DESELECT  | If given, specifies that the Collection can be deselected, yielding a state in which no item in the Collection is selected. When this property is set, the Collection can be deselected by pressing the Right mouse button on any item in the Collection. |
| STATE     | The current state of the Collection, which is the actual item selected.                                                                                                                                                                                   |
| INITSTATE | Specifies the initial state of the Collection. The value of this property is an item Link Description (see the section <b>Free Menu Item Links</b> .)                                                                                                     |

---

## Edit

Edit items are textual items that can be edited. The label for an Edit item cannot be a bitmap. When the item is selected an edit caret appears at that cursor position within the item, allowing inserting and deleting characters at that point. If selected with the Right mouse button, the item is cleared before editing starts. While editing, the Left mouse button moves the caret to a new position within the item. The Right mouse button deletes from the caret to the cursor. Control-W deletes the previous word.

Editing is stopped when another item is selected, when the user clicks in another tty window, or by the Free Menu function FM.ENDEDIT, which is called when the Free Menu is reset, or the window is closed. Additionally, the Free Menu editor will time out after about a minute, returning automatically. Because of the many ways in which editing can terminate, Edit items are not allowed to have a Selectedfn, as it is not clear when this function should be called.

Each Edit item should have an ID specified, which is used when getting the state of the Free Menu, since the string being edited is defined as the state of the item, and thus cannot distinguish edit items. The following Item Properties are particular to Edit items:

|               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MAXWIDTH      | Specifies the maximum string width of the item, in bits, after which input will be ignored. If MAXWIDTH is not specified, the items becomes "infinitely wide" and input is never restricted.                                                                                                                                                                                                                                                                                                                                                                                 |
| INFINITEWIDTH | This property is set automatically when MAXWIDTH is not specified. This tells Free Menu that the item has no right end, so that the item becomes mouse sensitive from its left edge to the right edge of the window, within the vertical space of the item.                                                                                                                                                                                                                                                                                                                  |
| LIMITCHARS    | The input characters allowed can be restricted in two ways: If this item property is a list, it is treated as a list of legal characters; any character not in the list will be ignored. If it is an atom, it is treated as the name of a test predicate, which is applied to ITEM WINDOW CHARACTER when each character is typed. This predicate should return T if the character is legal, NIL otherwise. The LIMITCHARS function can also call FM.ENDEDIT to force the editor to terminate, or FM.SKIPNEXT, to cause the editor to jump to the next edit item in the menu. |
| ECHOCHAR      | This item property can be set to any character. This character will be echoed in the window, regardless of what character is typed. However the item's label contains the actual string typed. This is useful for operations like password prompting. If ECHOCHAR is used, the font of the item must be fixed pitch.                                                                                                                                                                                                                                                         |

Unrestricted Edit items should not have other items to their right in the menu, as they will be edited over. If the item is boxed, input is restricted to what will fit in the box. Typing off the edge of the window will cause the window to scroll appropriately. Control characters can be edited, including CR and LF, and they are echoed as a black box. While editing, the Skip/Next key ends editing the current item, and starts editing the next Edit item in the Free Menu.

---

## Number

Number items are Edit items that are restricted to numerals. The state of the item is coerced to the the number itself, not a string of numerals.

There is one Number specific Item Property:

|            |                                                                                               |
|------------|-----------------------------------------------------------------------------------------------|
| NUMBERTYPE | If FLOATP (or FLOAT), then decimals are accepted. Otherwise only whole numbers can be edited. |
|------------|-----------------------------------------------------------------------------------------------|

---

## EditStart

EditStart items serve the purpose of starting editing on another item when they are selected. The associated Edit item is linked to the EditStart item by an EDIT link (see **Free Menu Item Links** below). If the EditStart item is selected with the Right mouse button, the Edit item is cleared before editing is started. Similar to Edit items, EditStart items cannot have a Selectedfn, as it is not clear when the associated editing will terminate.

---

## Display

Display items serve two purposes. First, they simply provide a way of putting dummy text in a Free Menu, which does nothing when selected. The item's label can be changed, though. Secondly, Display items can be used as the base for new item types. The application can create new item types by specifying DOWNFN, HELDFN, MOVEDFN, and SELECTEDFN for a Display item, making it behave as desired.

---

## Free Menu Item Highlighting

Each Free Menu Item can specify how it wants to be highlighted. First of all, if the item doesn't specify a HIGHLIGHT property, there are two default highlights. If the item is not boxed, the label is simply inverted, as in normal menus. If the item is boxed, it is highlighted in the shade of the box.

Alternatively, the value of the HIGHLIGHT property can be a SHADE, which will be painted on top of the item when a mouse event occurs on it. Or the HIGHLIGHT property can be an alternate label, which can be an atom, string, or bitmap. If the highlight label is a different size than the item label, the formatter will leave enough space for the larger of the two.

In all of these cases, the looks of the highlighted item are determined when the Free Menu is built, and a bitmap of the item with these looks is created. This bitmap is stored on the item's HIGHLIGHT property, and simply displayed when a mouse event occurs. The value of the highlight property in the Item Description is copied to the userdata list, in case it is needed later for a label change.

---

## Free Menu Item Links

---

Links between items are useful for grouping items in abstract ways. In particular, links are used for associating Editstart items with their item to edit, and State items with their state display. The Free Menu Item property LINKS is a property list, where the value of each Link Name property is a pointer to another item.

In the Item Description, the value of the LINK property should be a property list as above. The value of each Link Name property is a Link Description.

A Link Descriptions can be one of the following forms:

- |                                           |                                                                                                                                                                                                                                                |
|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>&lt;ID&gt;</code>                   | Simply an ID of an item in the Free Menu. This is okay if items can be distinguished by ID alone.                                                                                                                                              |
| <code>(&lt;GROUPID&gt; &lt;ID&gt;)</code> | A list whose first element is a GROUPID, and whose second element is the ID of an item in that group. This way items with similar purposes, and thus similar ID's, can be distinguished across groups.                                         |
| <code>(GROUP &lt;ID&gt;)</code>           | A list whose first element is the keyword GROUP, and whose second element is an item ID. This form describes an item with ID, in the same group that this item is in. This way you don't need to know the GROUPID, just which group you're in. |

Then after the entire menu is built, the links are setup, turning the Link Descriptions into actual pointers to Free Menu Items. There is no reason why circular Item Links cannot be created, although such a link would probably not be very useful. If circular links are created, the Free Menu will not be garbage collected after it is not longer being used. The application is responsible for breaking any such links that it creates.

---

## Free Menu Window Properties

---

- |                 |                                                                                                                                                                                                                                                                                         |
|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FM.PROMPTWINDOW | Specifies the window that Free Menu should use for displaying the item's messages. If not specified, PROMPTWINDOW is used.                                                                                                                                                              |
| FM.BACKGROUND   | The background shade of the entire Free Menu. This property can be set automatically by specifying a BACKGROUND argument to the function FREEMENU. The window border must be 4 or greater when a Free Menu background is used, due to the way the Window System handles window borders. |
| FM.DONTRESHAPE  | Normally Free Menu will attempt to use empty space in a window by pushing items around to fill the space. When a Free Menu window is reshaped, the items are repositioned in the new shape. This can be disabled by setting the FM.DONTRESHAPE window property.                         |

---

## Free Menu Interface Functions

---

|                                                             |                         |
|-------------------------------------------------------------|-------------------------|
| <code>(FREEMENU DESCRIPTION TITLE BACKGROUND BORDER)</code> | <code>[Function]</code> |
|-------------------------------------------------------------|-------------------------|

---



Creates a Free Menu from a Free Menu Description, returning the window. This function will return quickly unless new display fonts have to be created. See the example above.

## Accessing Macros

These Accessing Macros are provided to allow the application to get and set information in the Free Menu data structures. They are implemented as macros so that the operation will compile into the actual access form, rather than figuring that out at run time.

(FM.ITEMPROP *ITEM PROP {VALUE}*)

[Macro]

Similar to WINDOWPROP, this macro provides an easy access to the fields of a Free Menu Item. A handle on the item can be gotten from the Free Menu by the function FM.GETITEM, described below. *VALUE* is optional, and if not given, the current value of the *PROP* property will be returned. If *VALUE* is given, it will be used as the new value for that *PROP*, and the old value will be returned.

When a call to FM.ITEMPROP is compiled, if the *PROP* is known (quoted in the calling form), the macro figures out what field to access, and the appropriate Data Type access form is compiled. However, if the *PROP* is not known at compile time, the *function* FM.ITEMPROP, which goes through the necessary property selection at run time, is compiled.

The TYPE and USERDATA properties of a Free Menu Item are Read Only, and an error will result from trying to change the value of one of these properties.

(FM.GROUPPROP *WINDOW GROUP PROP {VALUE}*)

[Macro]

Provides access to the Group Properties set up in the PROPS list for each group in the Free Menu Description. *GROUP* specifies the ID of the desired group, and *PROP* the name of the desired property. If *VALUE* is specified, it will become the new value of the property, and the old value will be returned. Otherwise, the current value is returned.

(FM.MENUPROP *WINDOW PROP {VALUE}*)

[Macro]

Provides access to the group properties of the top-most group in the Free Menu, that is to say, the entire menu. This provides an easy way for the application to attach properties to the menu as a whole, as well as access the Group Properties for the entire menu.

(FM.NWAYPROP *WINDOW COLLECTION PROP {VALUE}*)

[Macro]

This macro works just like FM.GROUPPROP, except it provides access to the NWay Collections.

## Accessing Functions

(FM.GETITEM *ID GROUP WINDOW*)

[Function]

Get a handle on item *ID* in *GROUP* of the Free Menu in *WINDOW*. This function will search the Free Menu for an item whose ID property matches, or secondly whose LABEL property matches *ID*. If *GROUP* is NIL, then the entire Free Menu is searched. If no matching item is found, NIL is returned.

(FM.GETSTATE *WINDOW*)

[Function]

Return in property list format the ID and current STATE of every NWay Collection and item in the Free Menu. If an item's or Collection's state is NIL, then it is not included in the list. This provides an easy way of getting the state of the menu all at once. If the state of only one item or Collection is needed, the application

can directly access the STATE property of that object using the Accessing Macros above. Note that this function can be called when editing is in progress, in which case it will provide the label of the item being edited at that point.

## Changing Free Menus

---

Many of the following functions operate on Free Menu Items, and thus take the item as an argument. The *ITEM* argument to these functions can be the Free Menu Item itself, or just a reference to the item. In the second case, FM.GETITEM will be used to find the item in the Free Menu.

The reference can be in one of the following forms:

<ID> Specifies the first item in the Free Menu whose ID or LABEL property matches <ID>.

(<GROUPID> <ID>) Specifies the item whose ID or LABEL property matches <ID> within the group specified by <GROUPID>.

---

(FM.CHANGELABEL *ITEM NEWLABEL WINDOW UPDATEFLG*) [Function]

This function changes an item's label after the Free Menu has been created. It works for any type of item, and state items will remain in their current state. If the window is open, the item will be redisplayed with its new appearance. *NEWLABEL* can be an atom, a string, or a bit map (except for Edit items), and will be restricted in size by the MAXWIDTH and MAXHEIGHT Item Properties. If these properties are unspecified, the item will be able to grow to any size. *UPDATEFLG* specifies whether or not the regions of the groups in the menu are recalculated to take into account the change of size of this item. The application should not change the label of an Edit item while it is being edited.

The following Item Property is relevant to changing labels:

CHANGELABELUPDATE Exactly like *UPDATEFLG* except specified on the item, rather than as a function parameter.

---

(FM.CHANGESTATE *X NEWSTATE WINDOW*) [Function]

Programmatically changes the state of items and NWay Collections. *X* is either an item or a Collection name. For items *NEWSTATE* is a state appropriate to the type of the item. For NWay Collections, *NEWSTATE* should be the desired item in the Collection, or NIL to deselect. For Edit and Number items, this function just does a label change. If the window is open, the item will be redisplayed.

---

(FM.RESETSTATE *ITEM WINDOW*) [Function]

Set an item back to its initial state.

---

(FM.RESETMENU *WINDOW*) [Function]

Reset every item in the menu back to its initial state.

---

(FM.RESETSHAPE *WINDOW ALWAYSFLG*) [Function]

Reshapes the window to its full extent, leaving the lower-left corner unmoved. Unless *ALWAYSFLG* is T, the window will only be increased in size as a result of resetting the shape.

---

(FM.RESETGROUPS *WINDOW*) [Function]

Recalculate the extent of each group in the menu, updating group boxes and backgrounds appropriately.

(FM.HIGHLIGHTITEM *ITEM WINDOW*)

[Function]

This function provides a way of programmatically forcing an item to be highlighted. This might be useful for items which have a direct effect on other items in the menu. The item will be highlighted according to its HIGHLIGHT property, as described in the section **Free Menu Item Highlighting**. Note that this highlight is temporary, and will be lost if the item is redisplayed, by scrolling for example.

## Editor functions

(FM.EDITITEM *ITEM WINDOW CLEARFLG*)

[Function]

Start editing an Edit or Number item at the beginning of the item, as long as the window is open. This function will most likely be useful for starting editing of an item that is currently the null string. If *CLEARFLG* is set, the item is cleared first.

(FM.SKIPNEXT *WINDOW CLEARFLG*)

[Function]

This function causes the editor to jump to the beginning of the next Edit item in the Free Menu. If *CLEARFLG* is set, then the next item will be cleared first. If there is not another Edit item in the menu, this function will simply cause editing to stop. If this function is called when editing is not in progress, editing will begin on the first Edit item in the menu. This function can be called from any process, and can also be called from inside the editor, in a LIMITCHARS function.

(FM.ENEDIT *WINDOW WAITFLG*)

[Function]

Stop any editing going on in *WINDOW*. If *WAITFLG*, then block until the editor has completely finished. This function can be called from another process, or from a LIMITCHARS function.

(FM.EDITP *WINDOW*)

[Function]

If an item is in the process of being edited in the Free Menu *WINDOW*, that item is returned. Otherwise, NIL is returned.

## Miscellaneous

(FM.REDISPLAYMENU *WINDOW*)

[Function]

Redisplays the entire Free Menu in its window, if the window is open.

(FM.REDISPLAYITEM *ITEM WINDOW*)

[Function]

Redisplays a particular Free Menu Item in its window, if the window is open.

(FM.SHADE *X SHADE WINDOW*)

[Function]

*X* can be an item, or a group ID. *SHADE* is painted on top of the item or group. Note that this is a temporary operation, and will be undone by redisplaying. For more permanent shading, the application may be able to add a REDEDISPLAYFN and SCROLLFN for the window as necessary to update the shading.

(FM.WHICHITEM *WINDOW POSorX Y*)

[Function]

Gets a handle on an item from its known location within the window. If *WINDOW* is NIL, (*WHICHW*) is used, and if *POSorX* is NIL, the current cursor location is used.

(FM.TOPGROUPID *WINDOW*)

[Function]

Return the ID of the top group of this Free Menu.

## **Free Menu Changes:**

This document describes the incompatible changes from the old version to the new version of Free Menu. This document does not describe any of the new features of Free Menu. Some of the terminology used in these notes is introduced in the Free Menu documentation. You should read the Free Menu documentation first.

The function FREEMENU is used to create a Free Menu, replacing and combining the functions FM.MAKEMENU and FM.FORMATMENU.

## **In the Description of the Free Menu:**

There is no longer a WINDOWPROPS list in the Free Menu Description. Instead, the window properties TITLE and BORDER that used to be set in the WINDOWPROPS list can now be passed to the function FREEMENU. Other window properties (like FM.PROMPTWINDOW) can be set directly after FREEMENU returns the window using the system function WINDOWPROP. See the section in the documentation entitled **Free Menu Window Properties**.

Setting the initial state of an item is now done with the item property INITSTATE in the item description, rather than the STATE property.

## **Free Menu Items:**

3STATE items now have states OFF, NIL, and T (instead of a NEUTRAL state). They appear by default in the NIL state.

STATE items are general purpose items which maintain state, and replace the functionality of NCHOOSE items. To get the functionality of NCHOOSE items, specify the property MENUITEMS (a list of items to go in a popup menu), which instructs the STATE item to popup the menu when it is selected. STATE items do not display their current state by default, like NCHOOSE items used to. Instead, if you want the state displayed in the Free Menu, you have to link the STATE item to a DISPLAY item using a Free Menu Item Link named "DISPLAY". The current state of the STATE item will then automatically be displayed in the specified DISPLAY item. The item properties MENUFONT and MENUTITLE also apply to the popup menu.

NWAY items are declared slightly differently. There is now the notion of an NWay Collection, which is a collection of items acting as a single nway item. The Collection is declared by specifying any number of NWay items, each with the same COLLECTION property. NWay Collections have properties themselves, accessible by the macro FM.NWAYPROPS. These

properties can be specified in property list format as the value of the NWayPROPS Item Property of the first NWay item declared for each Collection. NWay Collections by default cannot be deselected (a state in which no item selected). Setting the Collection property DESELECT to any non-nil value changes this behavior. The state of the NWay Collection is maintained in its STATE property.

EDIT items no longer will stop at the edge of the window. Editing is either restricted by the MAXWIDTH property, or else it is not restricted at all. The EDITSTOP property is obsolete. Starting editing with the Right mouse button causes the item to be cleared first.

EDITSTART items now specify their associated edit item (there can only be one, now) by a Free Menu Item Link named "EDIT" from the EDITSTART item to the EDIT item.

TITLE items are replaced by DISPLAY items, which work the same way.

### **Free Menu Interface functions:**

(FREEMENU DESCRIPTION TITLE BACKGROUND BORDER) replaces FM.MAKEMENU and FM.FORMATMENU.

The desired format is not specified as the value of the FORMAT property in the group's PROPS list.

(FM.GETITEM ID GROUP WINDOW) replaces FM.ITEMFROMID.

Searches within GROUP for an item whose ID property is ID. ID is matched against the item ID and then the item LABEL. If GROUP is NIL, the entire menu is searched.

(FM.GETSTATE WINDOW) replaces FM.READSTATE.

Returns a property list of the selected item in the menu. This list now also includes the NWay Collections and their selected item.

(FM.CHANGELABEL ITEM NEWLABEL WINDOW UPDATEFLG) **new argument order.**

Now works by rebuilding the item label from scratch, taking the original specification of MAXWIDTH and MAXHEIGHT into account. NEWLABEL can be an atom, string, or bitmap. If UPDATEFLG is set, then the Free Menu Group's regions are recalculated, so that boxed groups will be redisplayed properly.

(FM.CHANGESTATE X NEWSTATE WINDOW) **new argument order.**

X is either an item or an NWay Collection ID. NEWSTATE is an appropriate state to the type of item. If an NWay collection, NEWSTATE is the actual item to be selected, or NIL to deselect. Toggle items take either T or NIL as NEWSTATE, and 3STATE items take OFF, NIL, or T, and STATE items take any atom, string, or

bitmap as their new state. For EDIT items, *NEWSTATE* is the new label, and FM.CHANGELABEL is called to change the label of the EDIT item.

(FM.RESETSHAPE WINDOW ALWAYSFLG) replaces FM.FIXSHAPE

(FM.HIGHLIGHTITEM ITEM WINDOW) replaces FM.SHADEITEM and FM.SHADEITEMBM.

FM.HIGHLIGHTITEM will programmatically highlight an item, as specified by its HIGHLIGHT property. The highlighting is temporary, and will be undone by a redisplay or scroll. To programmatically shade an item an arbitrary shade, use the new function FM.SHADE.

## ICONW

---

This is a package of functions for building small windows of arbitrary shape, principally for use as icons for shrinking windows; i.e., these functions are likely to be invoked from within the ICONFN of a window.

An icon is specified by supplying its image (a bit map) and a mask that specifies its shape. The mask is a bit map of the same dimensions as the image whose bits are on (black) in those positions considered to be in the image, off (white) in those positions where the background should “show through.” By using the mask and appropriate window functions, the icon package maintains the illusion that the icon window is nonrectangular, even though the actual window itself is rectangular. The illusion is not complete, of course. For example, if you try to select what looks like the background (or an occluded window) around the icon but still within its rectangular perimeter, the icon window itself is selected. Also, if you move a window occluded by an icon, the icon never notices that the background changed behind it.

Icons created with this package can also have “titles”; some part of the image can be filled with text computed at the time the icon is created, or even changed after creation.

---

## Creating Icons

---

(ICONW *IMAGE MASK POSITION NOOPENFLG*) [Function]

Creates a window at *POSITION*, or prompts for a position if *POSITION* is NIL. The window is borderless, and filled with *IMAGE*, as cookie-cut by *MASK*. If *MASK* is NIL, the image is considered rectangular (i.e., *MASK* defaults to a black bit map of the same dimensions as *IMAGE*). If *NOOPENFLG* is T, the window is returned unopened.

(TITLEDICONW *ICON TITLE FONT POSITION NOOPENFLG JUST BREAKCHARS OPERATION*) [Function]

Creates a titled icon at *POSITION*, or prompts for a position if *POSITION* is NIL. If *NOOPENFLG* is T, the window is returned unopened. The argument *ICON* is an instance of the record TITLEDICON, which specifies the icon image and mask, as with ICONW, and a region within the image to be used for displaying the title. Thus, the *ICON* argument is usually of the form

(create TITLEDICON *ICON\_someIconImage*  
*MASK\_iconMask TITLEREG\_someRegionWithinIcon*).

The title region is specified in icon-relative coordinates, i.e., the lower-left corner of the image bit map is (0, 0). The mask can be NIL if the icon is rectangular. The image should be white where it is covered by the title region (in any event, TITLEDICONW clears the region before printing on it).



The title is printed into the specified region in the image, using *FONT*, which if *NIL* defaults to the value of *DEFAULTICONFONT*, initially Helvetica 10. The title is broken into multiple lines if necessary; *TITLEDICONW* attempts to place the breaks at characters that are in the list of character codes *BREAKCHARS*. *BREAKCHARS* defaults to (CHARCODE (SPACE *Ÿ*)). In addition, line breaks are forced by any carriage returns in *TITLE*, independent of *BREAKCHARS*. *BREAKCHARS* is ignored as needed if a long title would not otherwise fit in the specified region. For convenience, *BREAKCHARS* = *FILE* means the title is a file name, so break at file name field delimiters.

The argument *JUST* indicates how the text should be justified relative to the region—it is an atom or list of atoms chosen from *TOP*, *BOTTOM*, *LEFT*, or *RIGHT*, which indicate the vertical positioning (flush to top or bottom) and/or horizontal positioning (flush to left edge or right). Where not indicated, the text is centered.

The argument *OPERATION* is a display stream operation indicating how the title should be printed. If *OPERATION* is *INVERT*, then the title is printed white-on-black. The default *OPERATION* is *REPLACE*, meaning black-on-white. *ERASE* is the same as *INVERT*; *PAINT* is the same as *REPLACE*.

For convenience, *TITLEDICONW* can also be used to create icons that consist solely of a title, with no special image. If the argument *ICON* is *NIL*, *TITLEDICONW* creates a rectangular icon large enough to contain *TITLE*, with a border the same width as a regular window. The remaining arguments are as described above, except that a *JUST* of *TOP* or *BOTTOM* is not meaningful.

---

## Modifying Icons

---

(*ICONW*.*TITLE* *ICON* *TITLE*)

[Function]

Returns the current title of the window *ICON*, which must be a window returned by *TITLEDICONW*. Additionally, if *TITLE* is non-*NIL*, makes *TITLE* be the new title of the window and repaints it accordingly. To erase the current title, make *TITLE* be a null string.

(ICONW.SHADE *WINDOW SHADE*)

[Function]

Returns the current shading of the window *ICON*, which must be a window returned by *ICONW* or *TITLEDICONW*. Additionally, if *SHADE* is non-NIL, paints the texture *SHADE* on *WINDOW*. A typical use for this function is to communicate a change of state in a window that is shrunk, without reopening the window. To remove any shading, make *SHADE* be *WHITESHADE*.

---

## Default Icons

---

When you shrink a window that has no *ICONFN*, the system currently creates an icon that looks like the window's title bar. You can make the system instead create titled icons by setting the global variable *DEFAULTICONFN* to the value *TEXTICON*.

(TEXTICON *WINDOW TEXT*)

[Function]

Creates a titled icon window for the main window *WINDOW* containing the text *TEXT*, or the window's title if *TEXT* is NIL.

DEFAULTTEXTICON

[Variable]

The value that *TEXTICON* passes to *TITLEDICONW* as its *ICON* argument. Initially NIL, which creates an unadorned rectangular window, but you can set it to a *TITLEDICON* record of your choosing if you would like default icons to have a different appearance.

---

## Sample Icons

---

The file <LispUsers>StockIcons contains a collection of icons and their masks usable with *ICONW*, including:

FOLDER, FOLDERMASK—A file folder

PAPERICON, PAPERICONMASK—A sheet of paper with the top right corner turned

FILEDRAWER, FILEDRAWERMASK—The front of a file drawer

ENVELOPEICON, ENVELOPEMASK—An envelope

TITLED.FILEDRAWER—A TitledIcon of the filedrawer front (Capacity, about three lines of 10-pt. text)

TITLED.FILEFOLDER—A TitledIcon of the file folder (Capacity, about three lines of 10-pt. text)

TITLED.ENVELOPE—A TitledIcon of the envelope (Capacity, one short line of 10-pt. text)

## APPENDIX B. SEDIT—THE LISP EDITOR

---

SEdit is the Xerox Lisp structure editor. It allows you to edit Xerox Lisp code directly in memory. This editor replaces DEdit in Chapter 16, Structure Editor, of the *Interlisp-D Reference Manual*. First introduced in Lyric, the SEdit structure editor has been greatly enhanced in the Medley release. Medley additions are indicated with revision bars in the right margin.

All symbols referenced in this appendix are external in the package named "SEdit", unless otherwise qualified.

---

### 16.1 SEdit - The Structure Editor

---

As a structure editor, SEdit alters Lisp code directly in memory. The effect this has on the running system depends on what is being edited.

For Common Lisp definitions, SEdit always edits a copy of the object. For example, with functions, it edits the definition of the function. What the system actually runs is the installed function, either compiled or interpreted. The primary difference between the definition and the installed function is that comment forms are removed from the definition to produce the installed function. The changes made while editing a function will not be installed until the edit session is complete.

For Interlisp functions and macros, SEdit edits the actual structure that will be run. An exception to this is an edit of an EXPR definition of a compiled function. In this case, changes are included and the function is unsaved when the edit session is completed.

SEdit edits all other structures, such as variables and property lists, directly. SEdit installs all changes as they are made.

If an error is made during an SEdit session, abort the edit with an Abort command (see Section 16.1.7, Command Keys). This command undoes all changes from the beginning of the edit session and exits from SEdit without changing your environment.

If the definition being edited is redefined while the edit window is open, SEdit redisplay the new definition. Any edits on the old definition will be lost. If SEdit was busy when the redefinition occurred, the SEdit window will be gray. When SEdit is no longer busy, position the cursor in the SEdit window and press the left mouse button; SEdit will get the new definition and display it.

### 16.1.1 An Edit Session

---

The List Structure Editor discussion in Chapter 3, Language Integration, explains how to start an editor in Xerox Lisp.

Whenever you call SEdit, a new SEdit window is created. This SEdit window has its own process, and thus does not rely on an Exec to run in. You can make edits in the window, shrink it while you do something else, expand it and edit some more, and finally close the window when you are done.

Throughout an edit session, SEdit remembers everything that you do through a change history. All edits can be undone and redone sequentially. When an edit session ends, SEdit forgets this information and installs the changes in the system.

The session ends with an event signalling to the editor that changes are complete. Three events signal completion:

- Closing the window.

Do this to terminate the edit session when you are finished.

- Shrinking the window.

Shrink the window when you have made some edits and may want to continue the editing session at a later time.

- Typing one of the Completion Commands, listed below.

Each of these commands has the effect of installing your changes, completing the edit, and returning the TTY process to the Exec. They vary in what is done in addition to completing. Using these commands the definition that you were editing can be automatically compiled, the edit window can be closed, or both.

A new edit session begins when you come back to an SEdit after completing. The change history is discarded at this point.

If the Exec is waiting for SEdit to return before going on, complete the edit session using any of the methods above to alert the Exec that SEdit is done. The TTY process passes back to the Exec.

### 16.1.2 SEdit Carets

---

There are two carets in SEdit, the edit caret and the structure caret. The edit caret appears when characters are edited within a single structure, such as an atom, string, or comment. Anything typed in will appear at the edit caret as part of the structure that the caret is within. The edit caret looks like this:

(a  b)

The structure caret appears when the edit point is between structures, so that anything inserted will go into a new structure. It looks like this:

(a  b)

SEdit changes the caret frequently, depending on where you are in the structure you are editing, and how the caret is positioned. The left mouse button allows an edit caret position to be set. The middle mouse button allows the structure caret position to be set.

### 16.1.3 The Mouse

In SEdit, the mouse buttons are used as follows. The left mouse button positions the mouse cursor to point to parts of Lisp structures. The middle mouse button positions the mouse cursor to point to whole Lisp structures. Thus, selecting the Q in LEQ using the left mouse button selects that character, and sets the edit caret after the Q:

```
(LEQ▲ n 1)
```

Any characters typed in at this point would be appended to the atom LEQ.

Selecting the same letter using the middle mouse button selects the whole atom (this convention matches TEdit's character/word selection convention), and sets a structure caret between the LEQ and the n:

```
(LEQ▲ n 1)
```

At this point, any characters typed in would form a new atom between the LEQ and the n.

Larger structures can be selected in two ways. Use the middle mouse button to position the mouse cursor on the parenthesis of the desired list to select that list. Press the mouse button multiple times, without moving the mouse, extends the selection. Using the previous example, if the middle button were pressed twice, the list (LEQ ...) would be selected:

```
(LEQ n 1)
```

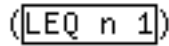
Pressing the button a third time would cause the list containing the (LEQ n 1) to be selected.

The right mouse button positions the mouse cursor for selecting sequences of structures or substructures. Extended selections are indicated by a box enclosing the structures selected. The selection is extended in the same mode as the original selection. That is, if the original selection were a character selection, the right button could be used to select more characters in the same atom. Extended selections also have the property of being marked for pending deletion. That is, the selection takes the place of the caret, and anything typed in is inserted in place of the selection.

For example, selecting the E by pressing the left mouse button and selecting the Q by pressing the right mouse button would produce:

```
(LEQ□ n 1)
```

Similarly, pressing the middle mouse button and then selecting with the right mouse button extends the selection by whole structures. Thus, in our example, pressing the middle mouse button to select LEQ and pressing the right mouse button to select the 1 would produce:



This is not the same as selecting the entire list, as above. Instead, the elements in the list are collectively selected, but the list itself is not.

#### 16.1.4 Gaps

---

The SEdit structure editor requires that everything edited must have an underlying Lisp structure, even if the structure is not directly displayed. For example, with quoted forms the actual structure might be (**QUOTE** GREEN), although this would be displayed as 'GREEN. Even when the user is in the midst of typing in a form, the underlying Lisp structure must exist.

Because of this necessity, SEdit provides gaps to serve as dummy Lisp objects during typing. SEdit does not need a gap for every form typed in, but gaps are necessary for quoted objects. When something is typed that requires SEdit to build a Lisp structure and thus create a gap, as the quote character does, the gap will appear marked for pending deletion. This means it is ready to be replaced by the structure to be typed in. In this way it is possible to type special structures, like quotes, directly, while SEdit maintains the structure.

A gap looks like: -x-

A gap displayed after a quote has been typed in would look like this:



with the gap marked for pending deletion, ready for typein of the object to be quoted.

#### 16.1.5 Special Characters

---

A few characters have special meaning in Lisp, and are treated specially by SEdit. SEdit must always have a complete structure to work on at any level of the edit. This means that SEdit needs a special way to type in structures such as lists, strings, and quoted objects. In most instances these structures can be typed in just as they would be to a regular Exec, but in a few cases this is not possible.

##### Lists- ( and )

Lists begin with an open parenthesis character (. Typing an open parenthesis gives a balanced list, that is, SEdit inserts both an open and a close parenthesis. The structure caret is between the two parentheses. List elements can be typed in at the structure caret. When a close parenthesis, ) is typed, the caret will be moved outside the list (and the close parenthesis), effectively finishing the

list. Square bracket characters, [ and ], have no special meaning in SEdit, as they have no special meaning in Common Lisp.

## Quoted Structures:

SEdit handles the quote keys so that it is possible to type in all quote forms directly. When typing one of the following quote keys at a structure caret, the quote character typed will appear, followed by a gap to be replaced by the object to be quoted.

### Single Quote – ’

Use to enter quoted structures.

### Backquote – `

Use to enter backquoted structures.

### Comma – ,

Use to enter comma forms, as used with a Backquote form.

### At Sign – @

Use after a comma to create a comma-at-sign gap. This allows type-in of comma-at forms, e.g. ,@list, as used within a Backquote form.

### Dot – .

Use the dot (period) after a comma to create a comma-dot gap. This allows type-in of comma-dot forms, e.g. ,.list, as used within a Backquote form.

### Hash Quote – #'

Use this two character sequence to enter the **CL:FUNCTION** abbreviation hash-quote (#').

## Dotted Lists:

The dot, or period, character (.) is used to type dotted lists in SEdit. After typing a dot, SEdit inserts a dot and a gap to fill in for the tail of the list. To dot an existing list, point the cursor between the last and second to the last element in the list, and type a dot. To undot a list, select the tail of the list before the dot while holding down the SHIFT key.

## Escape- \ or %

Use to escape from a specific typed in character. Use the escape key to enter characters, like parentheses, which otherwise have special meaning to the SEdit reader. Press the escape key then type in the character to escape. SEdit uses the escape key appropriate to the environment it is editing in; it depends on the readtable that was current when the editor was started. The backslash key (\) is used when editing Common Lisp, and the percent key (%) is used when editing Interlisp.

## Multiple Escape- |

Use the multiple escape key, the vertical bar character (|), to escape a sequence of typed in characters. SEdit always balances multiple escape characters. When one multiple escape character is typed, SEdit produces a balanced pair, with the caret between them, ready for typing in the characters to be escaped. If you type a second vertical bar, the caret moves after the second vertical bar, and is still within the same atom, so that you can add more unescaped characters to the atom.

## Comments- ;

The comment key, a semicolon (;), starts a comment. When a semicolon is typed, an empty comment is inserted with the caret in position for typing in the comment. Comments can be edited like strings. There are three levels of comments supported by SEdit: single, double, and triple. Single semicolon comments are formatted at the comment column, about three-quarters of the way across the SEdit window, towards the right margin. Double semicolon comments are formatted at the current indentation of the code that they are in. Triple semicolon comments are formatted against the left margin of the SEdit window. The level of a

comment can be increased or decreased by pointing after the semicolon, and either typing another semicolon, or backspacing over the preceding semicolon. Comments can be placed anywhere in your Common Lisp code. However, in Interlisp code, they must follow the placement rules for Interlisp comments.

**Strings- "**

Enter strings in SEdit by typing a double quote ("). SEdit balances the double quotes. When one is typed, SEdit produces a second, with the caret between the two, ready for typing the characters of the string. If a double quote character is typed in the middle of a string, SEdit breaks the string into two smaller strings, leaving the caret between them.

---

**16.1.6 Commands**

---

SEdit commands are most easily entered through the keyboard. When possible, SEdit uses a named key on the keyboard, for example, the DELETE key. The other commands are either Meta, Control, or Meta-Control key combinations. For the alphabetic command keys, either uppercase or lowercase will work.

There are two menus available, as an alternative means of invoking commands. They are the middle button popup menu, and the attached command menu. These menus are described in more detail below.

---

**16.1.6 Editing Commands**

---

|                             |                  |
|-----------------------------|------------------|
| <b>Redisplay: Control-L</b> | [Editor Command] |
|-----------------------------|------------------|

Redisplays the structure being edited.

|                                 |                  |
|---------------------------------|------------------|
| <b>Delete Selection: DELETE</b> | [Editor Command] |
|---------------------------------|------------------|

Deletes the current selection.

|                               |                  |
|-------------------------------|------------------|
| <b>Delete Word: Control-W</b> | [Editor Command] |
|-------------------------------|------------------|

Deletes the previous atom or whole structure. If the caret is in the middle of an atom, deletes backward to the beginning of the atom only.

---

**16.1.7 Completion Commands**

---

|                      |                  |
|----------------------|------------------|
| <b>Abort: Meta-A</b> | [Editor Command] |
|----------------------|------------------|

Aborts. This command must be confirmed. All changes since the beginning of the edit session are undone, and the edit is closed.

The following commands signal completion of an edit session and install the structure you were editing.

|                  |                  |
|------------------|------------------|
| <b>Control-X</b> | [Editor Command] |
|------------------|------------------|

Signals the system that this edit is complete. The window remains open, though, so the user can see the edit and start editing again directly.

|                  |                  |
|------------------|------------------|
| <b>Control-C</b> | [Editor Command] |
|------------------|------------------|

Signals the system that this edit is complete and compiles the definition being edited. The variable \*compile-fn\* determines the



function to be called to do the compilation. See the Options section below.

---

**Meta-Control-X** [Editor Command]

---

Signals the system that this edit is complete and closes the window.

---

**Meta-Control-C** [Editor Command]

---

Signals the system that this edit is complete, compiles the definition being editing, and closes the window.

### 16.1.8 Undo Commands

---



---

**Undo: Meta-U or UNDO** [Editor Command]

---

Undoes the last edit. All changes since the beginning of the edit session are remembered, and can be undone sequentially.

---

**Redo: Meta-R or AGAIN** [Editor Command]

---

Redoes the edit change that was just undone. Redo only works directly following an Undo. Any number of Undo commands can be sequentially redone.

### 16.1.9 Find Commands

---



---

**Find: Meta-F or FIND** [Editor Command]

---

Finds a specified structure, or sequence of structures. If there is a current selection, SEdit looks for the next occurrence of the selected structure. If there is no selection, SEdit prompts for the structure to find, and searches forward from the position of the caret. The found structure will be selected, so the Find command can be used to easily find the same structure again.

If a sequence of structures is selected, SEdit will look for the next occurrence of the same sequence. Similarly, when SEdit prompts for the structure to find, you can type a sequence of structures to look for.

The variable *\*wrap-search\** controls whether or not SEdit wraps around from the end of the structure being edited and continues searching from the beginning.

---

**Reverse Find: Control-Meta-F** [Editor Command]

---

Finds a specified structure, searching in reverse from the position of the caret.

The variable *\*wrap-search\** controls whether or not SEdit wraps around from the beginning of the structure being edited and continues searching from the end.

---

**Find Gap: Meta-N or SKIP-NEXT**[Editor Command]

---

Skips to the next gap in the structure, leaving it selected for pending deletion.

---

**Substitute: Meta-S or SHIFT-FIND**[Editor Command]

---

Substitutes one structure, or sequence of structures, for another structure, or sequence, within the current selection. SEdit prompts you in the SEdit prompt window for the structures to replace, and the structures to replace with.

The selection to substitute within must be a structure selection. To get a structure selection, click with the middle mouse button (not the left), and extend it, if necessary, with the right mouse button. If you begin with the left button, you will get an informational message "Select the structure to substitute within", because the selection was of characters, rather than structures.

---

**Delete Structure: Meta-Control-S**[Editor Command]

---

Removes all occurrences of a structure or sequence of structures within the current selection. SEdit prompts the user in the SEdit prompt window for the structures to delete.

---

**16.1.9 General Commands**

---

---

**Arglist: Meta-H or HELP**[Editor Command]

---

Shows the argument list for the function currently selected, or currently being typed in, in the SEdit prompt window. If the argument list will not fit in the SEdit prompt window, it is displayed in the main Prompt Window.

---

**Convert Comments: Meta-;**[Editor Command]

---

Converts old style comments in the selected structure to new style comments. This converter notices any list that begins with an asterisk (\*) in the INTERLISP package (IL:\*) as an old style comment. Section 16.1.11, Options, describes the converter options .

---

**Edit: Meta-O**[Editor Command]

---

Edits the definition of the current selection. If the selected name has more than one type of definition, SEdit asks for the type to be edited. If the selection has no definition, a menu pops up. This menu lets the user specify either the type of definition to be created, or no definition if none needs to be created.

---

**Eval: Meta-E or Do-It**[Editor Command]

---

Evaluates the current selection. If the result is a structure, the inspector is called on it, allowing the user to choose how to look at the result. Otherwise, the result is printed in the SEdit prompt window. The evaluation is done in the process from which the edit session was started. Thus, while editing a function from a break window, evaluations are done in the context of the break.

---

**Expand: Meta-X or EXPAND**[Editor Command]

---

Replaces the current selection with its definition. This command can be used to expand macros and translate CLISP.

**Extract: Meta- /** [Editor Command]

Extracts one level of structure from the current selection. If the current selection is an atom, or if there is no selection, the next largest structure containing this atom, or caret, is used. This command can be used to strip the parentheses off a list or a comment, or to unquote a quoted structure.

**Inspect: Meta-I** [Editor Command]

Inspect the current selection.

**Join: Meta-J** [Editor Command]

Joins. This command joins any number of sequential Lisp objects of the same type into one object of that type. Join is supported for atoms, strings, lists, and comments. In addition, SEdit permits joining of a sequence of atoms and strings, since either type can easily be coerced into the other. In this case, the result of the Join will be an atom if the first object in the selection is an atom, otherwise the result will be a string.

**Mutate: Meta-Z** [Editor Command]

Mutates. This command allows the user to do arbitrary operations on a LISP structure. First select the structure to be mutated (it must be a whole structure, not an extended selection). When the user presses Meta-Z SEdit prompts for the function to use for mutating. This function is called with the selected structure as its argument, and the structure is replaced with the result of the mutation.

For example, an atom can be put in upper case by selecting the atom and mutating by the function U-CASE. You can replace a structure with its value by selecting it and mutating by EVAL.

**Quote: Meta-'**  
**Meta-'**  
**Meta-,**  
**Meta-.**  
**Meta-@ or Meta-2**  
**Meta-# or Meta-3** [Editor Command]

Quotes the current selection with the specified kind of quote, respectively, Single Quote, Backquote, Comma, Comma-At-Sign, Comma-Dot, or Hash-Quote.

**Parenthesize: Meta- ) or Meta-0** [Editor Command]

Parenthesizes the current selection, positioning the caret after the new list.

**Parenthesize: Meta- ( or Meta-9** [Editor Command]

Parenthesizes the current selection, positioning the caret at the beginning of the new list. Only a whole structure selection or an extended selection of a sequence of whole structures can be parenthesized.

### 16.1.10 Miscellaneous

---

**Change Print Base: Meta-B**

[Editor Command]

Changes Print Base. Prompts for entry of the desired Print Base, in decimal. SEdit redisplay fixed point numbers in this new base.

---

**Set Package: Meta-P**

[Editor Command]

Changes the current package for this edit. Prompts the user, in the SEdit prompt window, for a new package name. SEdit will redisplay atoms with respect to that package.

---

**Attached Menu: Meta-M**

[Editor Command]

Attaches a menu of the commonly used commands (the SEdit Command Menu) to the top of the SEdit window. Each SEdit window can have its own menu, if desired.

### 16.1.10 Help Menu

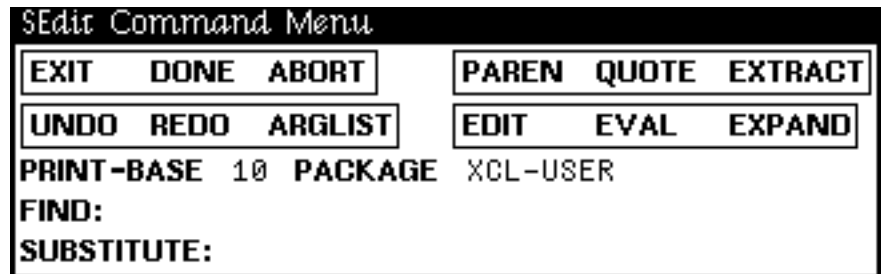
When the mouse cursor is positioned in the SEdit title bar and the middle mouse button is pressed, a Help Menu of commands pops up. The menu looks like this:

| Commands               |       |
|------------------------|-------|
| Abort                  | M-A   |
| Done                   | C-X   |
| Done & Compile         | C-C   |
| Done & Close           | M-C-X |
| Done, Compile, & Close | M-C-C |
| Undo                   | M-U   |
| Redo                   | M-R   |
| Find                   | M-F   |
| Reverse Find           | M-C-F |
| Remove                 | M-C-S |
| Substitute             | M-S   |
| Find Gap               | M-N   |
| Arglist                | M-H   |
| Convert Comment        | M-;   |
| Edit                   | M-O   |
| Eval                   | M-E   |
| Expand                 | M-X   |
| Extract                | M-/   |
| Inspect                | M-I   |
| Join                   | M-J   |
| Mutate                 | M-Z   |
| Parenthesize           | M-(   |
| Quote                  | M-'   |
| Set Print-Base         | M-B   |
| Set Package            | M-P   |
| Attach Menu            | M-M   |

The Help Menu lists each command and its corresponding Command Key. (In the menu, the letter C stands for CONTROL, while M indicates Meta.) The command selected is executed just as if the command had been entered from the keyboard. The menu remembers which command was selected last, and pops up with the mouse cursor next to that same command the next time the menu is used. This provides a very fast way to repeat the same command when using the mouse.

### 16.1.9 Command Menu

The SEdit Attached Command Menu contains the commonly used commands. Use the Meta-M keyboard command to bring up this menu. The menu can be closed, independently of the SEdit window, when desired. The menu looks like:



All of the commands in the menu function identically to their corresponding keyboard commands, except for Find and Substitute.

When Find is selected with the mouse cursor, SEdit prompts in the menu window, next to the Find button, for the structures to find. Type in the structures then select Find again. The search begins from the caret position in the SEdit window.

Similarly, Substitute prompts, next to the Find button, for the structures to find, and next to the Substitute button for the structures to substitute with. After both have been typed in, selecting Substitute replaces all occurrences of the Find structures with the Substitute structures, within the current selection.

To do a confirmed substitute, set the edit point before the first desired substitution, and select Find. Then if you want to substitute that occurrence of the structure, select Substitute. Otherwise, select Find again to go on.

Selecting either Find or Substitute with the right mouse button erases the old structure to find or substitute from the menu, and prompts for a new one.

### 16.1.11 SEdit Window Region Manager

SEdit provides user redefinable functions which control how SEdit chooses the region for a new edit window.

**(Get-Window-Region** *CONTEXT REASON NAME TYPE*)

**[Function]**

This function is called when SEdit wants to know where to place a window it is about to open. This happens whenever the user starts a new SEdit or expands an Sedit icon. The default behavior is to

pop a window region off SEdit's stack of regions that have been used in the past. If the stack is empty, SEdit prompts for a new region.

This function can be redefined to provide different behavior. It is called with the edit *CONTEXT*, a *REASON* for needing a region, the *NAME* of the structure to be edited, and the *TYPE* of the structure to be edited. The edit *CONTEXT* is SEdit's main data structure and can be useful for associating particular edits with specific regions. The *REASON* argument specifies why SEdit wants a region, and will be one of the keywords :CREATE or :EXPAND.

---

**(Save-Window-Region *CONTEXT REASON NAME TYPE REGION*)** [Function]

This function is called whenever SEdit is finished with a region and wants to make the region available for other SEdits. This happens whenever an SEdit window is closed or shrunk, or when an SEdit icon is closed. The default behavior is simply to push the region onto SEdit's stack of regions.

This function can be redefined to provide different behavior. It is also called with the edit *CONTEXT*, the *REASON*, the *NAME*, the *TYPE*, and additionally the window *REGION* that is being released. The *REASON* argument specifies why SEdit is releasing the region, and will be one of the keywords :CLOSE, :SHRINK, or :CLOSE-ICON.

---

**Keep-Window-Region** [Variable]

Default **T**. This flag determines the behavior of the default SEdit region manager, explained above, for shrinking and expanding windows. When set to **T**, shrinking an SEdit window will not give up that window's region; the icon will always expand back into the same region. When set to **NIL**, the window's region is made available for other SEdits when the window is shrunk. Then when an SEdit icon is expanded, the window will be reshaped to the next available region.

This variable is only used by the default implementations of the functions **Get-Window-Region** and **Save-Window-Region**. If these functions are redefined, this flag is no longer used.

---

### 16.1.12 Options

The following parameters can be set as desired.

---

**\*Wrap-Parens\*** [Variable]

This SEdit pretty printer flag determines whether or not trailing close parenthesis characters, ), are forced to be visible in the window without scrolling. By default it is set to **NIL**, meaning that close parens are allowed to "fall off" the right edge of the window. If set to **T**, the pretty printer will start a new line before the structure preceding the close parens, so that all the parens will be visible.

---

**\*Wrap-Search\*** [Variable]

This flag determines whether or not SEdit find will wrap around to the top of the structure when it reaches the end, or vice versa in the case of reverse find. The default is **NIL**.

**\*Clear-Linear-On-Completion\***

[Variable]

This flag determines whether or not SEdit completely re-pretty prints the structure being edited when you complete the edit. The default value is NIL, meaning that SEdit reuses the pretty printing.

**Convert-Upgrade**

[Variable]

Default 100. When using Meta-; to convert old-style single- asterisk comments, if the length of the comment exceeds **Convert-Upgrade** characters, the comment is converted into a double semicolon comment. Otherwise, the comment is converted into a single semicolon comment.

Old-style double-asterisk comments are always converted into new-style triple-semicolon comments.

**16.1.13 Control Functions****(Reset)**

[Function]

This function recomputes the SEdit edit environment. Any changes made in the font profile, or any changes made to SEdit's commands are captured by resetting. Close all SEdit windows before calling this function.

**(Add-Command KEY-CODE FORM &OPTIONAL KEY-NAME COMMAND-STRING HELP-STRING)**

[Function]

This function allows you to write your own SEdit keyboard commands. You can add commands to new keys, or you can redefine keys that SEdit already uses as command keys. If you mistakenly redefine an SEdit command, the function Reset-Commands will remove all user-added commands, leaving SEdit with its default set of commands.

*KEY-CODE* can be a character code, or any form acceptable to il:charcode.

*FORM* determines the function to be called when the key command is typed. It can be a symbol naming a function, or a list, whose first element is a symbol naming a function and the rest of the elements are extra arguments to the function. When the command is invoked, SEdit will apply the function to the edit context (SEdit's main data structure), the charcode that was typed, and any extra arguments supplied in *FORM*. The extra arguments do not get evaluated, but are useful as keywords or flags, depending on how the command was invoked. The command function must return T if it handled the command. If the function returns NIL, SEdit will ignore the command and insert the character typed.

The optional arguments are used to add this command to SEdit's middle button menu. When the item is selected from the menu, the command function will be called as described above, with the charcode argument set to NIL.

*KEY-NAME* is a string to identify the key (combination) to be typed to invoke the command. For example "M-A" to represent the Meta-A key combination, and "M-C-A" for Meta-Control-A.

*COMMAND-NAME* is a string to identify the command function, and will appear in the menu next to the *KEY-NAME*.

*HELP-STRING* is a string to be printed in the prompt window when a mouse button is held down over the menu item.

After adding all the commands that you want, you must call **Reset-Commands** to install them.

For example:

```
(add-command "↑U" (my-change-case t))
(add-command "↑Y" (my-change-case nil))
(add-command "l,r" my-remove-nil
 "M-R" "Remove NIL"
 "Remove NIL from the selected structure"))
(reset-commands)
```

will add three commands. Suppose *my-change-case* takes the arguments *CONTEXT*, *CHARCODE*, and *UPPER-CASE?*. *UPPER-CASE?* will be set to T when *my-change-case* is called from Control-U, and NIL when called from Control-Y. *my-remove-nil* will be called with only *CONTEXT* and *CHARCODE* arguments when Meta-R is typed.

Below are some SEdit functions which are useful in writing new commands.

---

**(Reset-Commands)****[Function]**

This function installs all commands added by **Add-Command**. SEdits which are open at the time of the **Reset-Commands** will not see the new commands; only new SEdits will have the new commands available.

---

**(Default-Commands)****[Function]**

This function removes all commands added by **Add-Command**, leaving SEdit with its default set of commands. As in **Reset-Commands**, open SEdits will not be changed; only new SEdits will have the user commands removed.

---

**(Get-Prompt-Window *CONTEXT*)****[Function]**

This function returns the attached prompt window for a particular SEdit.

---

**(Get-Selection *CONTEXT*)****[Function]**

This function returns two values: the selected structure, and the type of selection, one of NIL, T, or :SUB-LIST. The selection type NIL means there is not a valid selection (in this case the structure is meaningless). T means the selection is one complete structure. :SUB-LIST means a series of elements in a list is selected, in which case the structure returned is a list of the elements selected.

---

**(Replace-Selection *CONTEXT STRUCTURE SELECTION-TYPE*)****[Function]**

---



This function replaces the current selection with a new structure, or multiple structures, by deleting the selection and then inserting the new structure(s). The *SELECTION-TYPE* argument must be one of T or :SUB-LIST. If T the *STRUCTURE* is inserted as one complete structure. If :SUB-LIST, the *STRUCTURE* is treated as a list of elements, each of which is inserted.

### 16.1.13 Programmer's Interface

This programmer's interface to SEdit provides a way to call SEdit directly. This interface is sketchy and will change in the future.

**\*Getdef-Fn\*** [Variable]

This function is called with the arguments *NAME*, *TYPE*, and *OLDDEF*, when SEdit needs to refetch the definition for the named object being edited. When SEdit is first started it gets passed the structure, so this function doesn't get called. But after completion, SEdit refetches because it doesn't know if the Edit Interface (File Manager) changed the definition upon installation. The function returns the new definition.

**\*Fetch-Definition-Error-Break-Flag\*** [Variable]

This flag, along with the error options listed below, determines what happens when the Getdef-Fn errors. The default value is NIL, causing errors to be suppressed. When set to T, the break will be allowed.

**\*Getdef-Error-Fn\*** [Variable]

This function is funcalled with the arguments *NAME*, *TYPE*, *OLDDEF*, and *PROMPT-WINDOW*, when the Getdef-Fn errors, independent of whether or not the break is suppressed. This function should return the structure to be used in place of the unavailable new definition.

**\*Edit-Fn\*** [Variable]

This function is funcalled with the selected structure as its argument from the Edit (M-O) command. It should start the editor as appropriate, or else generate an error if the selection is not editable.

**\*Compile-Fn\*** [Variable]

This function is funcalled with the arguments *NAME*, *TYPE*, and *BODY*, from the compile completion commands. It should compile the definition, *BODY*, and install the code as appropriate.

**(SEdit STRUCTURE PROPS OPTIONS)** [Function]

This function provides a means of starting SEdit directly. *STRUCTURE* is the structure to be edited.

*PROPS* is a property list, which may specify the following properties:

:name - the name of the object being edited

:type - the file manager type of the object being edited. If NIL, SEdit will not call the file manager when it tries to refetch the

definition it is editing. Instead, it will just continue to use the structure that it has.

:completion-fn - the function to be called when the edit session is completed. This function is called with the *CONTEXT*, *STRUCTURE*, and *CHANGED?* arguments. *CONTEXT* is SEdits main data structure. *STRUCTURE* is the structure being edited. *CHANGED?* specifies if any changes have been made, and is one of NIL, T, or :ABORT, where :ABORT means the user is aborting the edit and throwing away any changes made. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments that the function is applied to following the main arguments above.

:root-changed-fn - the function to be called when the entire structure being edited is replaced with a new structure. This function is called with the new structure as its argument. If the value of this property is a list, the first element is treated as the function, and the rest of the elements are extra arguments that the function is applied to following the structure argument.

*OPTIONS* is one or a list of any number of the following keywords:

:fetch-definition-suppress-errors - If this option is provided, any error under the Getdef-Fn will be suppressed, regardless of the :fetch-definition-allow-errors option or the value of **\*Fetch-Definition-Error-Break-Flag\***.

:fetch-definition-allow-errors - If this option is provided, any error under the Getdef-Fn will be allowed to break.

:dontwait - This option specifies that the call to **SEdit** should return as soon as the editor is started, rather than waiting for a completion command.

:close-on-completion - This option specifies that SEdit cannot remain active for multiple completions. That is, the SEdit window cannot be shrunk, and the completion commands that normally leave the window open will in this case close the window and terminate the edit.

:compile-on-completion - This option specifies that SEdit should call the \*Compile-Fn\* to compile the definition being edited upon completion, regardless of the completion command used.

---

## Fixed ARS

**AR 7471** ---- You no longer have to Ctrl-X out of SEdit to compile. To compile, select either of two new SEdit commands. The C-C command (Ctrl -C) compiles and leaves the SEdit window open. The M-C-C command (Meta-Ctrl- C) compiles and closes the window.

**AR 7783** ---- After typing Meta-O, you are prompted to "Select a type of dummy definition to install." The first option is "Optimizers". This title refers to XCL:DEFOPTIMIZERS, and has nothing to do with CL:OPTIMIZE.

**AR 7786** ---- Square bracket characters '[' and ']' are not treated as special characters by SEdit, just as in Common Lisp.

[This page intentionally left blank]

Test script for SEdit version February 24, 1987

SEE CHANGE CONTROL FOR THE FILE SEDIT

FIXED:

7692 M-E breaks on numbers  
7682 Ctrl-W with pending delete selection breaks  
7705 M-H with extended selection breaks  
7717 M-J breaks when nothing selected  
7759 forgets package of mutator candidate  
7731 M-J breaks with a sequence of numbers  
7509 SEDIT.RESET needs IP fonts  
7576 SEDIT.RELOAD should not be part of SEdit

Select a number and hit M-E, the result will be the number.

CTRL-W when there is a pending delete selection will do nothing. The easy test case is single-quote then ctrl-w.

Make an extended selection and hit M-H (help). Sedit will now say "Select function you want the arguments for"

Just after Starting SEdit, or any other time nothing is selected and/or there is no point, M-J used to break. Will now say "Select items to join."

SEdit now remembers the package of the mutate function candidate. Type in the atom abc. Get into the LISP package (M-P, LISP). Select the atom abc. Hit M-Z, and type IL:U-CASE, cr. Now hit M-Z again, the old SEdit would prompt with U-CASE, the new sedit will prompt with IL:U-CASE.

Select a sequence of numbers, or any sequence with a number first. Hit M-J. SEdit will say "Can't join numbers". SEdit will join AB 123 CD, though, as it used to.

With INTERPRESSFONTDIRECTORIES set to NIL, call SEDIT.RESET. It won't try to create IP fonts, and thus won't break.

The function SEDIT.RELOAD is not longer in the sysout.

„ wasn't recognized as a quote type. (SEdit '(BQUOTE (A (\, FOO) B) used to display as '(A (\, FOO) B) and now will display as '(A „FOO B).

And now for all the shift selecting stuff:

Weren't able to Move select a structure into a pending delete selection. Now you can unless the structure overlaps the pending delete selection in some way. SEdit will say "Can't move a structure which overlaps the selection" if you try.

Can move select an object out of a quote, into any destination (exec, same sedit, different sedit), and the object will be replaced by a gap, which is selected pending delete.

Shift selecting something into a string didn't use to be completely undoable. Try Move selecting an atom into a string, then hit undo.

Move selecting something into a different destination when there is a main selection used to leave the selection messed up. Now it doesn't.

---

---

## TWODINSPECTOR

---

---

By: Jan Pedersen (Pedersen.PA @ Xerox.ARPA)

The TWODINSPECTOR package provides a two-dimensional inspector window abstraction, very similar in form to the standard one-dimensional inspector but laid out in rows and columns, instead of just rows.

The top level function is TWODINSPECTW.CREATE

```
(TWODINSPECTW.CREATE DATUM ROWPROPS COLUMNPROPS FETCHFN ^
 STOREFN VALUECOMMANDFN ROWPROPCOMMANDFN ^
 COLUMNPROPCOMMANDFN TITLE TITLECOMMANDFN
 WHERE TOPRIGHT) [Function]
```

Datum is the object to be inspected. Rowprops is a list of properties of the datum which will be laid out vertically, or a function which will be called with datum as an argument and returns such a list. Similarly, columnprops is a list of properties of the datum which will be laid out horizontally, or a function which will be called with datum as an argument and returns such a list. Each pair (rowprop, columprop) specifies a cell of the twodimensional inspector window. Fetchfn is a function which if called with arguments datum, rowprop, and columprop returns the value in that cell. Storefn is a function which if called with arguments newvalue, datum, rowprop, and columprop stores newvalue in the cell.

The cells of the inspector window are selectable. If valuecommandfn is given, it must be a function which will be called with arguments cellvalue, rowprop, columprop, datum, and twodinspectwindow when the cell specified by (rowprop, columprop) is selected. A default valuecommandfn is provided which allows the cellvalue to be inspected, set, or bound to the litatom IT.

Similarly the rowprops and the columnprops themselves are selectable. If rowpropcommandfn is given it must be a function which will be called with args rowprop, datum, and twodinspectwindow when rowprop is selected. If columpropcommandfn is given it must be a function which will be called with args columprop, datum, and twodinspectwindow when columprop is selected. No default rowpropcommandfn or columpropcommandfn is provided. If rowpropcommandfn is not given, the rowprops will not be selectable. Similarly, If columpropcommandfn is not given, the columnprops will not be selectable.

Title will be the title for the window -- a default is provided. Titlecommandfn is a function which will be called with the single argument twodinspectwindow if the middle button is depressed in the title bar of the window.

Where may be a window, in which case it will be used as at least part of the twodinspector (the twodinspector is composed of five window), This is especially useful if where is the result of a previous

call to `TWODINSPECTW.CREATE`. The dimensions of `where` will not be used to position the `twodinspector` unless `topright` is `NIL`.

`Where` may also be region or a position specifying the lower left hand corner of the `twodinspector`. If `where` is `NIL`, the user will be prompted for a position.

`Topright` allows the user to specify the top right-hand corner of the `twodinspector`. `Topright` must be a position, and if given overrides any specification which may have been provided by the argument `where`.

Returns the main window of an attached window group.

The arguments to `TWODINSPECTW.CREATE` are cached on windowprops of the same name on the returned main window.

Several functions are provided for use in the various command functions.

(`TWODINSPECT.REDISPLAY` *TWODINSPECTW* *SOMEROWPROPS* ^  
*SOMECOLUMNPROPS*) [Function]

Redisplay selected cells of `twodinspectw`. `Somerowprops` may either be a single rowprop, a list of rowprops, or `NIL`. `Somecolumnprops` may either be a single columnprop, a list of columnprops, or `NIL`. If either are `NIL` the entire `twodinspectw` is recomputed and redisplayed. Otherwise, the cells specified by the cross product of `somerowprops` and `somecolumnprops` are redisplayed, possibly forcing the entire `twodinspectw` to redisplay if the printed representation of a cell overflows its column width.

(`TWODINSPECT.REPLACE` *TWODINSPECTW* *ROWPROP* *COLUMNPROP* ^  
*NEWVALUE*) [Function]

Replaces the cell specified by (`rowprop`, `columnprop`) with `newvalue` and updates the display.

(`TWODINSPECT.SELECTITEM` *TWODINSPECTW* *ROWPROP* *COLUMNPROP*) [Function]

Selects the cell specified by (`rowprop`, `columnprop`). That cell is inverted and put on the window prop `SELECTION` of `twodinspectw`. If either of `rowprop` or `columnprop` is `NIL`, then the current selection is simply deselected.

(`TWODINSPECT.SELECTROWPROP` *TWODINSPECTW* *ROWPROP*) [Function]

Selects `rowprop`. If `rowprop` is `NIL`, then the currently selected `rowprop` is deselected.

(TWODINSPECT.SELECTCOLUMNPROP *TWODINSPECTW* COLUMNPROP) [Function]

Selects columnprop. If rowprop is NIL, then the currently selected columnprop is deselected.

Note: there is no provision for redisplaying selected row or column props -- although this may be effected by redisplaying the entire twodinspectw.

Since the Twodinspector windows differ stylistically from the standard inspector windows, a stylistically similar onedinspector window is also provided.

(ONEDINSPECTW.CREATE *DATUM PROPS FETCHFN STOREFN VALUECOMMANDFN ^  
PROP COMMANDFN TITLE TITLECOMMANDFN WHERE TOPRIGHT*) [Function]

Datum is the object to be inspected. Props is a list of properties of the datum which will be laid out horizontally, or a function which will be called with datum as an argument and returns such a list. Each prop specifies a cell of the onedimensional inspector window. Fetchfn is a function which if called with arguments datum, and prop returns the value in that cell. Storefn is a function which if called with arguments newvalue, datum, and prop stores newvalue in the cell.

The cells of the inspector window are selectable. If valuecommandfn is given, it must be a function which will be called with arguments cellvalue, prop, datum, and onedinspectwindow when the cell specified by prop is selected. A default valuecommandfn is provided which allows the cellvalue to be inspected, set, or bound to the litatom IT.

Similarly the props themselves are selectable. If propcommandfn is given it must be a function which will be called with args prop, datum, and onedinspectwindow when prop is selected. No default propcommandfn is provided. If propcommandfn is not given, the props will not be selectable.

Title will be the title for the window -- a default is provided. Titlecommandfn is a function which will be called with the single argument onedinspectwindow if the middle button is depressed in the title bar of the window.

Where may be a window, in which case it will be used as at least part of the onedinspector (the onedinspector is composed of three window), This is especially useful if where is the result of a previous call to ONEDINSPECTW.CREATE or TWODINSPECTW.CREATE. The dimensions of where will not be used to position the onedinspector unless topright is NIL.

Where may also be region or a position specifying the lower left hand corner of the onedinspector. If where is NIL, the user will be prompted for a position.

Topright allows the user to specify the top right-hand corner of the onedinspector. Topright must be a position, and if given overrides any specification which may have been provided by the argument where.

Returns the main window of an attached window group.



The arguments to ONEDINSPECTW.CREATE are cached on windowprops of the same name on the returned main window.

(ONEDINSPECT.REDISPLAY *ONEDINSPECTW SOMEPROPS*) [Function]

Redisplay selected cells of onedinspectw. Someprops may either be a single prop, a list of props, or NIL, in which case the entire onedinspectw is recomputed and redisplayed. Otherwise, the cell(s) specified by the someprops are redisplayed, possibly forcing the entire someprops to redisplay if the printed representation of a cell overflows the column width.

(ONEDINSPECT.REPLACE *ONEDINSPECTW PROP NEWVALUE*) [Function]

Replaces the cell specified by prop with newvalue and updates the display.

(ONEDINSPECT.SELECTITEM *ONEDINSPECTW PROP*) [Function]

Selects the cell specified by prop. That cell is inverted and put on the window prop SELECTION of onedinspectw. If prop is NIL, then the current selection is simply deselected.

(ONEDINSPECT.SELECPROP *ONEDINSPECTW PROP*) [Function]

Selects prop. If prop is NIL, then the currently selected prop is deselected.

















---

---

**BACKGROUND-MENU-BUTTONS**

---

---

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on January 28, 1988.

**INTRODUCTION**

BACKGROUND-MENU-BUTTONS makes it easy for the Rooms user to make buttons which do the same things as entries on the background menu.

**EXPLANATION**

When this module is loaded a button labelled "Make Background Button" is placed on the screen. When this button is pressed a menu which looks just like the background menu is raised. Selecting an entry from this menu will create a button which does the same thing as this entry.



Envos Corporation  
1157 San Antonio Road  
Mountain View, California 94043  
(415) 966-6200  
California: (800) 824-6449  
Continental U.S.: (800) 228-5325

September, 1988

Dear ROOMS User:

Attached is the documentation for the Rooms User's modules.

Note that most of these modules include source code. Thus besides being useful in themselves, these modules provide good examples of programmatic use of ROOMS.

Rooms User modules are user-contributed software, and as such are not supported Envos products. We merely distribute these modules in hope that they will prove useful.

Please direct questions, suggestions and problems with Rooms User's modules directly to their authors. We have included the ARPAnet addresses of the authors in the documentation for this purpose.

We encourage you to contribute more Rooms User's modules. The same policies which apply to Lisp User's modules apply here. For more information see the Lisp User's guidelines distributed with the Medley release.

Customer Support

---

---

**>>Module Name<<**

---

---

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

>>Type INTERNAL here if the file is for Internal Use Only<<

This document last edited on >>DATE<<.

**INTRODUCTION**

>>This paragraph should be replaced by an overview of your module.<<

**MODULE EXPLANATIONS****>>Functions, Variables, and Lisp Code Examples<<**

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE *DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN*) [Function]

The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system. Variables look like functions, except that the word "Variable," enclosed in **square brackets**, follows the variable name. Please note that these are the characters [], not the parentheses

This is an example of code. It is in 10-point Terminal font.

Function names, commands, file names, and the like are in 10-point modern.

Be sure to include the following information in any module explanations:

- any file dependencies
- definitions of all arguments
- module, function variable, etc. limitations
- a liberal number of examples for all functions, variables, etc.

# XEROX

Xerox Corporation  
Palo Alto Research Center  
3333 Coyote Hill Road  
Palo Alto, California 94304

## HYPERDESK and HYPERDESK-INIT

The HyperDesk (or "MobyDesk") sysout is a "Swiss Army Sysout" that includes much of the best local functionality. This includes Sketch, DictTool, PCL, CommonLens, Rooms, and Notecards. The HyperDesk-INIT file provides a powerful init file for HyperDesk Users as well as those that may use other sysouts but would like convenient access to HyperDesk functionality. This Init file can make life easier for those who don't hack their own init files by making it easier on those that are called in to do so. A reasonable goal is that users of this class wouldn't need separate init files and instead will be able to tailor thier environment through a control panel and automatic profile saving.

HYPERDESK

Author: Ramana Rao  
Filed on: {EG:PARC:XEROX}<RAO>HYPERDESK>HYPERDESK.TEDIT;5  
Last filed: April 26, 1988 14:14 PDT (Tuesday)  
Copyright © 1987 by Xerox Corporation. All rights reserved.

**\*DRAFT\***

## HYPERDESK

The HyperDesk.SYSOUT is available from <LISP>LYRIC>BASICS> on your favorite Lisp Servers (i.e. IE: and PHYLUM). It contains the SKETCH, DICTTOOL, ROOMS, PCL, COMMON-LENS, and NOTECARDS.

\CC-HYPERDESK-MSG

[Variable]

When HYPERDESK is greeted, it automatically sends a LAFITE message to the maintainer. By default this message is CC'ed to the person using the sysout; this CC'ing can be disabled by setting the variable \CC-HYPERDESK-MSG to NIL in the users Init File.

Loading HYPERDESK-INIT

## Loading HYPERDESK-INIT

The HYPERDESK-INIT file is available to PARC users of the Lyric release of Xerox Lisp. It can be found on the directory {EG:PARC:}<RAO>HYPERDESK>. Including the file command

```
(FILES (FROM LISPUSERS)
 "HYPERDESK-INIT")
```

in your personal INIT file will automatically load HYPERDESK-INIT when you start up a new Lisp.

HYPERDESK-INIT is essentially wrapping around GENERIC-INIT, so users of HYPERDESK-INIT should understand the essentials of GENERIC-INIT. Documentation for it is available from {EG:}<LANNING>LISP>USERS>GENERIC-INIT.TEDIT. Both of these centralized init files set variables and invoke operations that tailor the environment. Much of this behavior can be controlled or preempted by setting user parameters or special control variables either before or after the load of HYPERDESK-INIT. Two variables, LOGINHOST/DIR and HOME-MACHINE-NAME, must be set before the load of HYPERDESK-INIT (or at least setting them should be considered explicitly). These are used as control variables to GENERIC-INIT and are documented there.

An example of a minimal init file can be found on {EG:}<RAO>HYPERDESK>LYRIC-INIT. Any of these INIT files (HYPERDESK-INIT, GENERIC-INIT, LYRIC-INIT, my {EG:}<RAO>LISP>INIT-LYRIC or Stan's {EG:}<LANNING>LISP>LYRIC-INIT) can be

loaded prop into your sysout without any thing happening so you can see what would happen. The one caveat is that they will smash filecoms of files with the same name.

Tailoring HYPERDESK-INIT

## Tailoring HYPERDESK-INIT

There are a number of variables that can be set before and after loading HYPERDESK-INIT to tailor its behavior to ones liking. In addition, users can also tailor GENERIC-INIT by setting variables documented in the GENERIC-INIT documentation. A few of the GENERIC-INIT variables (\*LOAD-UTILITY-OPTIONS\*) are used by HYPERDESK-INIT, so users should use the variables provided by HYPERDESK-INIT as described below before resorting to GENERIC-INIT variables. Unless otherwise noted, all symbols described are in the INTERLISP package.

### Things you MAY do before loading

**\*HYPERDESK-FULL-INIT-P\*** [Variable]

When HYPERDESK-INIT is loaded, it will ask if it should do a full initialization. By default this message will time out, set this variable and continue to a full initialization. This is especially convenient for starting non-hyperdesk sysouts where you don't want to wait for special status hyperdesk functionality to be loaded.

**\*HYPERDESK-BASE-MODULES\*** [Variable]

**\*HYPERDESK-MODULES\*** [Variable]

**\*HYPERDESK-EXTRA-MODULES\*** [Variable]

These variables control what modules are loaded automatically by GENERIC-INIT. HYPERDESK-INIT initvars \*LOAD-UTILITY-OPTIONS\* to the append of these three variables in the order listed. \*HYPERDESK-MODULES\* is initialized to the special status hyperdesk modules (ROOMS, PCL, and COMMON-LENS, and a bunch of window types for ROOMS) unless \*HYPERDESK-FULL-INIT-P\* is NIL in which case \*LOAD-UTILITY-OPTIONS\* will not be touched. Note that these should all be lists of strings where the strings are the name of the package as seen on the Load Utility Menu (i.e. the name of the top level or only file of the module e.g. LoadPCL or COMMON-LENS). See GENERIC-INIT \*LOAD-UTILITY-OPTIONS\*.

## HYPERDESK

|                           |            |
|---------------------------|------------|
| *HYPERDESK-SUITES-P*      | [Variable] |
| *HYPERDESK-AX-POCKETS-P*  | [Variable] |
| *HYPERDESK-AX-ORIGINAL-P* | [Variable] |
| *HYPERFLOOR-SUITE-P*      | [Variable] |
| *HYPERDESK-SUITE-P*       | [Variable] |
| *HYPERDESK-USER-SUITES*   | [Variable] |

HYPERDESK-INIT provides some mechanisms for setting up initial suites. Setting \*HYPERDESK-SUITES-P\* to NIL will disable this facility altogether. Setting \*HYPERDESK-AX-POCKETS-P\* and \*HYPERDESK-AX-ORIGINAL-P\* to NIL will disable deletion of POCKETS and ORIGINAL rooms respectively. Setting \*HYPERFLOOR-SUITE-P\* and \*HYPERDESK-SUITE-P\* to NIL will disable loading of these two default suites respectively (note that disabling HYPERFLOOR suite will also disable the HYPERDESK Suite). Finally \*HYPERDESK-USER-SUITES\* should be a list of fully qualified file names that determine which suites get loaded initially.

Hyper Suites

## HyperFloor and HyperDesk Suites

HyperFloor and HyperDesk Suites provide a basic framework for working with Rooms and other HyperDesk functionality.

HyperFloor provides three rooms: HyperPanel which is basic inclusion room, Help where eventually various buttons for getting documentation will be put, and the Boiler Room where buttons for fixing things and dealing with the HyperDesk in general will be put. One particularly important button in the Boiler Room is one that allows you to reload the HyperFloor so you can import any recent additions to the Boiler or Help Rooms.

HyperDesk provides two rooms: Mail Room and Office. Users should copy these rooms to make suites of their own and avoid working in these if they want to build their own suite.

---

---

## LAFITE-WINDOW-TYPES

---

---

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on January 28, 1988.

### **INTRODUCTION**

This module provides window types for Lafite windows. Lafite is the Interlisp-D mail program. Loading this module enables you to save Lafite windows in your suite files.

### **EXPLANATIONS**

We provide window types for Lafite browsers and for the Lafite status window.

Note: we do not provide window types for message windows, as these are far more transient than the status window and browsers.



---

---

**OFFICE.SUITE**

---

---

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on January 28, 1988.

**INTRODUCTION**

This is a sample suite file.

Beginning Rooms users might load this, augment it as desired, and then use the "Dump Suite" command to save their own version of it.

This suite also gives some ideas about how one might use the facilities which Rooms provides.

**INSTALLATION**

Load the file OFFICE.SUITE. This file (like all suite files) will load ROOMS if it is not already loaded. It will then create the rooms in the OFFICE suite. Use the Overview to see these rooms.

---

---

**RANDOM-WINDOW-TYPES**

---

---

By: Doug Cutting (Cutting.PA@Xerox.COM),  
Stan Lanning (Lanning.PA@Xerox.COM) and  
Ramana Rao (Rao.PA@Xerox.COM)

This document last edited on April 21, 1988.

This module contains window type definitions for many LispUsers modules.

These definitions serve two purposes. Their primary purpose is to allow one to save windows created by these modules in suites. Secondly they provide good examples of window type definitions for the programmer attempting to define his own.

Modules covered are:

WHO-LINE  
CALENDAR  
PRINTERMENU  
CROCK  
BICLOCK  
ADDRESSBOOK  
PHONE-DIRECTORY  
GRID-ICONS

In addition, support is provided for the FILEWATCH LispUsers module. This works by adding a property to each room which notes whether FILEWATCH is on in that room. This property is not inherited, i.e. including a room in which FILEWATCH is does not turn FILEWATCH on.

A listing of RANDOM-WINDOW-TYPES can be found in Appendix A of this document.

---

---

**>>Module Name<<**

---

---

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

This document last edited on >>DATE<<.

**INTRODUCTION**

>>This paragraph should be replaced by an overview of your module.<<

**MODULE EXPLANATIONS****>>Functions, Variables, and Lisp Code Examples<<**

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE *DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN*) [Function]

The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system. Variables look like functions, except that the word "Variable," enclosed in **square brackets**, follows the variable name. Please note that these are the characters [], not the parentheses

This is an example of code. It is in 10-point Terminal font.

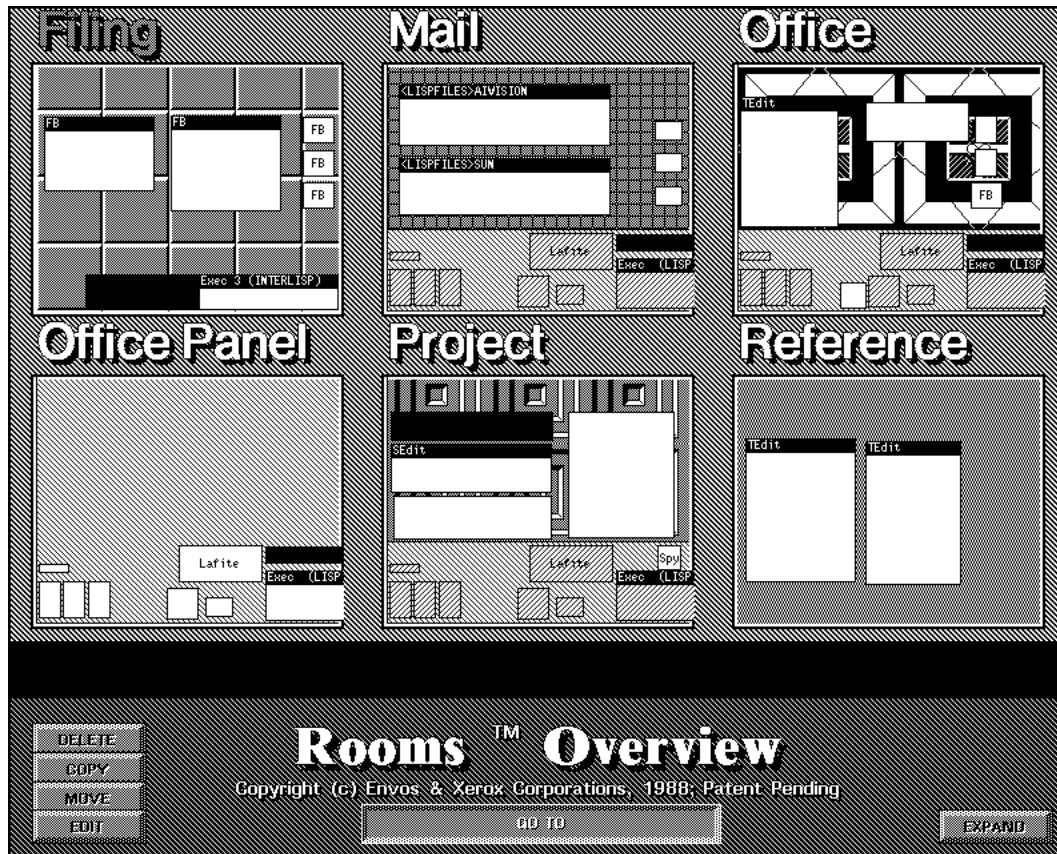
Function names, commands, file names, and the like are in 10-point modern.

Be sure to include the following information in any module explanations:

- any file dependencies
- definitions of all arguments
- module, function variable, etc. limitations
- a liberal number of examples for all functions, variables, etc.



# ROOMS™



## Overview

ROOMS is a powerful interface to the Medley lisp environment window management system. ROOMS effectively increases the size of the screen and allows you to organize the work environment to facilitate management of complex parallel tasks.

For example a user may be developing and debugging a program, writing a paper, doing background research, reading mail, and filing all more or less at the same time. With limited screen space a user performing all these activities loses productivity shuffling through windows, icons, and other assorted screen clutter in order to reestablish the context of a task. ROOMS solves this problem of context switching by allowing users to create workspaces (called rooms), each one analogous to another screen, and containing only those tools needed for a specific task. Moreover, ROOMS provides methods for easily moving from one room to another and customizing the "look" of each room further aiding in efficient task-switching.

By providing dedicated workspaces, ease of navigation between workspaces, and a graphical link between each workspace and its related task, ROOMS helps users minimize context recovery time caused by task switching. In addition ROOMS provides an easy means to develop highly graphical custom interfaces for

end-user applications. The overall result of this seamless integration of form and functionality is increased productivity in performing tasks in the Envos Software Development Environment.

## Research Background

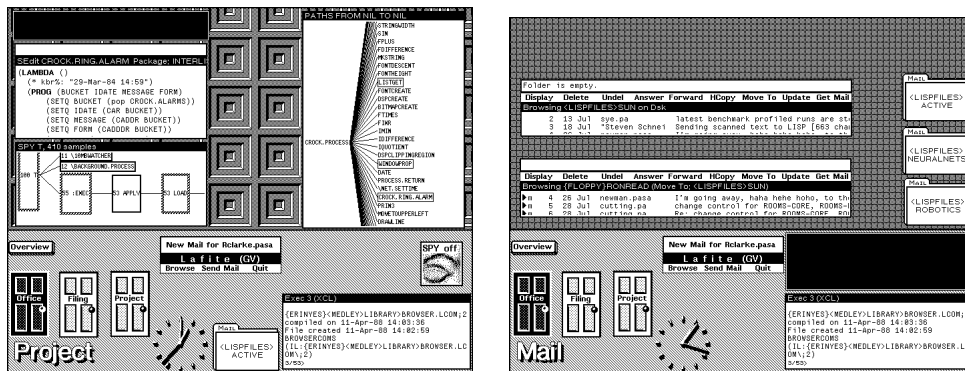
The ROOMS system design resulted from the following observations by members of the Xerox Palo Alto Research Labs:

- 1) Most intellectual work requires coordinating many sources of information, e.g. notes, drafts, spreadsheets, program listings, etc.
- 2) Before the widespread dissemination of desktop workstations and personal computers, knowledge workers typically spread paper containing these items out on a desk (or dining room table). But the small screens of computer workstations do not allow such a lavish use of space when working on many concurrent tasks. For example, it takes 22 average PC screens or 10 19" workstation screens to equal the area of a typical desk.
- 3) The result for overlapped windows systems is a kind of electronic messy desk, where the user spends large amounts of time moving, shrinking, and resizing windows in order to switch tasks.
- 4) An interface designed to assist a user to work on concurrent tasks rather than assume linear work habits would have the following features:
  - Fast task switching and fast task resumption
  - Easy to re-acquire mental task context
  - Access to a large amount of information
  - Fast Access to information
  - Low user overhead
  - Engaged tools shareable among several tasks
  - Collections of engaged-tools shareable among tasks
  - Task-specific presentation of shared-engaged tools.

How Envos' implementation of ROOMS accomplishes these goals is described in the next section "The ROOMS Design".

## The ROOMS Design

Rooms solves the problem of user task switching by allowing users to create a number of screen-sized workspaces called Rooms. The figure below shows two Rooms.



Two example Rooms (a) used for programming and (b) used for reading mail

In each room, there are a number of small icon-like objects called Doors. When a door is selected with the mouse, the user has the illusion of transiting to a new room containing windows. Each room is related to a different major task, such as working on a particular project, reading mail or file management. In the room are a number of engaged tools related to the task. When a user re-enters a room it appears exactly as when it was left.

The basic notion of the Rooms scheme is simple. But in order for this basic notion to be successful the ROOMS system design has many features to help manage window interaction among rooms and to keep the user from getting lost.

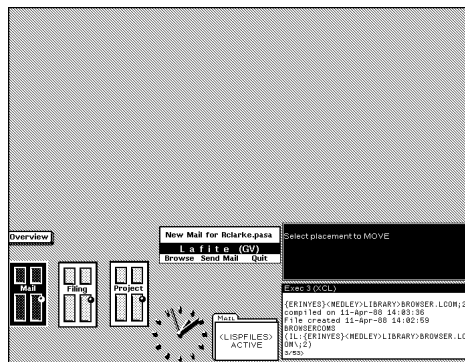
## Task Interaction

---

When working in ROOMS, there are usually a number of engaged tools that the user wants to share among different tasks. Examples of these might include a code editor, a text editor or a clock. The following ROOMS design features allow for the sharing of engaged tools:

**Placements** A window in a room has a specific location and shape. This is known as a Placement. The same window can exist in several rooms, with different locations, shapes, etc., in each room. Actions done on a shared window in one room are reflected in another room. Note that the black promptwindow in the Mail and Project rooms has a different location and shape in each room.

**Inclusions** Rooms can be included within other rooms. This allows collections of rooms to share a common set of windows, known as Inclusions. The band of windows and icons common in both parts of the Mail and Project rooms is a control panel, shown below, implemented as an included room.



Example of an included room called Office Panel

**Baggage** When moving between rooms you may carry windows with you. For example, you may wish to bring program code from one workspace to a workspace where you are doing documentation. This is accomplished by holding down the Move or Copy key when selecting a door. You will then be prompted to select the windows you wish to carry with you to the next room. The windows will have the same presentation in the new room.

**Pockets** You can also have a constant piece of baggage called a Pocket. A Pocket is a room dynamically included in all rooms. Whichever windows are placed in your Pocket ( a clock say) will automatically occur (at the same location and presentation attributes) in all rooms.

## Navigation

---

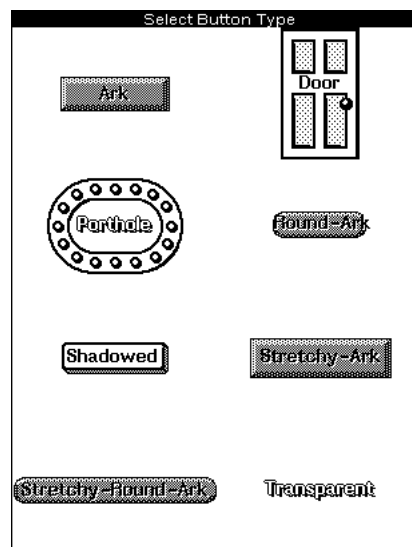
The organization of the user's workspace into a number of workspaces creates a potential navigational problem. ROOMS solves this problem with a number of navigational aids.

- Doors** Doors provide the basic link mechanism between rooms. Selecting a door with the mouse "moves" you to the new room containing windows associated with that room.
- Back Doors** Rooms allows you to create a Back Door in a room. If you have created a Back Door you can easily move back to the task you were previously working on before you entered the present room.
- Rooms Menu** Selecting the Go To Room sub-item from the ROOMS background menu presents you with a list of Room names. Selecting one will move you to that Room.
- Overview** The main feature of the Overview is a set of Room pictographs as shown in the opening diagram. From the Overview you can see the overall layout of a room and the tools it contains. The room pictographs can be instantly expanded one at a time, allowing you to browse through the windows in the entire set of rooms. In addition you can enter any room directly via the Overview.

## Tailorability

---

Several mechanisms are provided to help users tailor rooms. First, simply creating, moving, deleting and shaping windows in the usual way causes things to exist in rooms. Thus ROOMS preserves the natural interaction with the user. Second, special background entries are provided to allow the user to create new doors and other conveniences of construction. Below is an example of the standard set of Doors available to users.



At the Overview level, it is possible to copy, move or delete window pictographs within a room and between rooms and have the changes reflected in the rooms themselves. And finally, Rooms has a simple layout language for creating unique backgrounds for Rooms. By using the structure editor, SEdit, on this layout language, users can run arbitrary procedures on the entrance and exit of a room and can compute



specialized backgrounds for rooms. Below is an example of a SEdit of a rooms layout expression for the Project room shown earlier.

```
SEdit Project Package: XCL-USER
(:INCLUSIONS ("Office Panel") :BACKGROUND
 ((:REGION (0 1/4 1.0 3/4) :SHADE
 (:EVAL ROOMS:RENAISSANCE-BITMAP)
 :BORDER 2)
 (:TEXT "Project" :FONT
 (IL:HELVETICA 36 IL:BOLD)
 :POSITION (10 . 10))))
```

### Suites

Having designed an environment to suit your needs, ROOMS allows you to restore that state if there is a need to reload your system. Subsets of rooms can be grouped together to form Suites. Suites can be saved to a file and later restored to a new environment or shared with other users. Most system windows have a ROOMS window-type specification which allows them to be saved and restored from a suite. Thus a Filebrowser window that you have open on a particular system can be saved and the window placement and contents will be restored when you reload the suite onto a new system.

### Buttons

Buttons are a unique user-interface device that provides for the execution of any lisp command at the click of the mouse. Buttons can be of any shape and size. Doors are examples of buttons that move you from one room to another. For example selecting the following button will perform a directory listing of the local disk.



The button definition is shown below:

```
SEdit Directory Package: XCL-USER
(:TEXT "Directory" :FONT
 (IL:HELVETICA 12
 (IL:BOLD IL:REGULAR
 IL:REGULAR)))
:SHADOWS :ARK :TYPE :STRETCHY-ROUND-ARK
:HELP "Bring up local disk directory"
:ACTION (DIR "{dsk}<lispfiles>")
:INVERTED? NIL)
```

### Customization of Rooms

When used as the interface to an application, ROOMS provides you with a complete programmatic interface to allow application specific customization. This includes the ability to have applications create and switch rooms under programmatic control as well as enable users to design custom buttons and window-types.

## References

---

A Multiple, Virtual-Workspace Interface to Support User Task Switching, Austin Henderson/Stuart Card, 1987 ACM CHI + GI Conference Proceedings on Human Factors in Computing Systems and Graphic Interfaces

For further information about ROOMS contact your Envos Marketing Representative, or call toll-free in the Continental United States 1-800-228-5325, or in California 1-800-824-6449.

Envos Corporation

1157 San Antonio Road

Mountain View, California 94043

(415) 966-6200

ROOMS is a trademark of Xerox Corporation

ROOMS is licensed to Envos by Xerox for use in the Envos Software Development Environment

---

---

## ROOMS USERS' RULES

---

---

This document describes the rules and procedures for "Rooms Users" modules. This document is mainly for Rooms Users' module writers, but users should also understand the rules.

### **DEVELOPING A ROOMS USERS' MODULE**

A Rooms Users' module is a useful program made available to the general Rooms community. Neither the author nor the custodian of Rooms Users' imputes any warranty of suitability or responsibility for errors.

Rooms Users' modules should be easily distinguishable from released library and Lisp Users' modules. In particular, this means that a Rooms Users' module may not have the same name as a Library or Lisp Users' module and should be visibly different. Rooms Users' modules derived from released software should be announced to the public only after communicating with the organization responsible for that released software.

Testing is important. If you make significant changes to a Rooms Users' module, enlist users at your site as alpha testers. A Rooms Users' module is not shoddy software; it is software made available outside the regular release channels.

### **ROOMS USERS' MODULE OWNERSHIP**

A module submitted for Rooms Users' remains the "property" of the submitter. Others may not make changes, except for their own private use, without negotiating with the owner (who may already be making similar or incompatible changes).

As the owner of a module, you are not required to fix bugs, but if not, you must be willing to transfer ownership (permanently) to someone who volunteers to fix them. Ownership may pass back and forth among several people as long as they agree.

### **SUBMITTING ROOMS USERS' MODULES**

If you are not an internal user, you should submit your new module to us through e-mail or on a floppy or tape. External users should make sure that they include all relevant information, such as documentation containing an e-mail or US mail address where he/she can be reached.

### **SUBMITTING FILES TO ROOMS USERS'**

As with released software, it is important to submit not just the resulting product, but all the files needed to build and maintain a Rooms Users' module:

1. the file to load ( .LCOM or .dfasl or .SUITE)
2. documentation describing it, following the formatting rules (see below)
3. a source file that can be released (optional)
4. data files, if needed

Modules submitted once are released once. Do not assume that a module submitted for one release will be automatically released in subsequent releases.

**DOCUMENTATION**

No modules will be released without documentation. Documentation can be as simple as a paragraph describing what the module does and how to use it, or it can be as extensive as a dozen-page user manual. All modules should have a file with a .TEDIT extension. Formatting should be done according to the rules outlined in the Rooms Users' Template, included on the Rooms Users floppy or tape as EASYTEMPLATE.TEDIT and also printed in this document. All users, external users included, should follow the Rooms Users' Template rules. If the documentation is large and formatting time consuming, you can also produce an interpress file (with the .ip extension), as well as submitting a .TEdit file. (Be sure to update the interpress file if you update the documentation!) Documentation should include the full address of the submitter.

**COMPATIBILITY**

Any submitted Rooms Users' files should be compilable in a "vanilla" Rooms environment. The file itself should load in any auxiliary modules when necessary.

Thanks for your cooperation.

## Table of Contents

|                                              |     |
|----------------------------------------------|-----|
| INTRODUCTION .....                           | 1   |
| ROOMS USERS' Rules.....                      | 2   |
| ROOMS USERS' Template.....                   | 4   |
| BACKGROUND-MENU-BUTTONS.....                 | 5   |
| OFFICE.SUITE.....                            | 6   |
| RANDOM-WINDOW-TYPES .....                    | 7   |
| TOUCHY-BUTTONS.....                          | 8   |
| UN-HIDE-TTY .....                            | 9   |
| WALLPAPER.....                               | 10  |
| APPENDIX A-RANDOM-WINDOW-TYPES LISTINGS..... | A-1 |

[This page intentionally left blank]

# ROOMS USER'S MODULES



Address comments to:  
ENVOS  
User Documentation  
1157 San Antonio Rd.  
Mountain View, CA 94043  
415-966-6200

---

ROOMS USER'S MODULES

PART NUMBER 300002

SEPTEMBER 1988

Copyright © 1988 by ENVOS Corporation.

All rights reserved.

Copyright protection includes material generated from the software programs displayed on the screen, such as icons, screen display looks, and the like.

ROOMS is a trademark of Xerox Corporation.

The following are Sun Microsystems Inc. trademarks:

SunOS

SunView

Sun Workstation is a registered trademark.

UNIX is a registered trademark of AT&T Bell Laboratories.

---

The information in this document is subject to change without notice and should not be construed as a commitment by ENVOS Corporation. While every effort has been made to ensure the accuracy of this document, ENVOS Corporation assumes no responsibility for any errors that may appear.

Text was written and produced with ENVOS text formatting tools; Xerox printers were used to produce text masters. The typeface is Optima.



---



---

## TOUCHY BUTTONS

---



---

By: Ramana Rao (Rao.pa)

Uses: Rooms

This document last edited on January 22, 88.

### INTRODUCTION

This Rooms Users Package provides a number of "touchy" buttons i.e. buttons that visually depend on the state of the world and change the state of the world when touched. Right now I provide three types of touchy buttons: `includer`, `toggler`, and `once-only`. You can make a touchy button by calling `rooms::make-<touchy-type>`. I will take suggestions for any others that people think may be useful since I'm trying to abstract the touchy technology.

### BUTTON CONSTRUCTORS

(ROOMS::MAKE-INCLUDER *ROOM-NAME*) [Function]

Includers allow you to conveniently mixin or mixout rooms. For example, you can have "Notecards-Mixin" and "Programming-Mixin" Rooms and have includer buttons in your personal "Pockets." Then you can include these functionality traits whenever you need them no matter the room.

(ROOMS::MAKE-TOGGLER *VARIABLE-NAME*) [Function]

Toggle buttons allow you to toggle boolean variables. This should obviously be generalized to something that allows you to select or circulate through value settings.

(ROOMS::MAKE-ONCE-ONLY *FORM INITIAL-TEXT FINAL-TEXT*) [Function]

A once-only button evaluates a form exactly once in a sysout and then displays that it is exhausted. For Example:

```
(ROOMS::MAKE-ONCE-ONLY (IL:PROMPTPRINT "Hello, World") "Fire..."
 "Exhausted")
```

---

---

## UN-HIDE-TTY

---

---

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on April 28, 1988.

ROOMS changes the window system such that open windows are not always visible. Some applications presume that open windows are visible. This module attempts to amend this situation.

Often when an application decides to start using a window which is not visible it gives that window the caret. CHAT is an example of such an application. When IL:CLOSECHATWINDOWFLG is NIL (the default) CHAT windows are left open after their connection is closed. If one leaves such a CHAT window in another room and then attempts to open a CHAT connection, CHAT will merrily re-use the hidden window. But as CHAT takes the caret when it opens a connection, we can identify that window and pull it into the current room.

(ROOMS:UN-HIDE-TTY)

[Function]

If the window with the caret is not in the current room, it is brought into the current room with a call to ROOMS:UN-HIDE-WINDOW. If the window with the TTY is already visible on the screen then it is flashed. If there is no TTY window (i.e. the process with the keyboard has no TTY window) then a message to this effect is printed to the prompt window.

One can use this function in the action of a button, e.g.:

```
(ROOMS:MAKE-BUTTON-WINDOW
 (ROOMS:MAKE-BUTTON :TEXT "Un-Hide TTY" :ACTION '(ROOMS:UN-HIDE-TTY)))
```

Control-Y

[Interrupt]

Brings the window with the caret into the current room by calling ROOMS:UN-HIDE-TTY. This interrupt is installed when UN-HIDE-TTY is loaded. This will not work with applications which have their own interrupt tables, e.g. TEdit and CHAT.

---

---

### ROOMS USERS' MODULES

---

---

#### INTRODUCTION

ROOMS User's modules are written by users of Envos ROOMS. They are supported by the individuals who wrote them, not by Envos. Envos takes no responsibility for the reliability and maintenance of these modules.

Each module is documented, some in detail, others with only enough information to get the user started. At the top of each document, the name of the author is written, along with his electronic mail or U.S. mail address. Please contact the author with any problems you may have.

Note that most of these modules include source code. Thus besides being useful in themselves, these modules provide good examples of programmatic use of ROOMS.

This document contains explanations of each module, written by the author of each. It also contains an explanation of how to submit your own ROOMS User' module. Please read these instructions carefully if you wish submit a module.

---

---

## WALLPAPER

---

---

By: Doug Cutting (Cutting.PA@Xerox.COM) and  
Larry Masinter (Masinter.PA@Xerox.COM)

Uses: SCREENPAPER

This document last edited on August 8, 1988.

### INTRODUCTION

This module provides an easy way to create distinctive backgrounds for your rooms.

All symbols described in this document are in the package ROOMS.

### FUNCTIONS

(MAKE-WALLPAPER-WINDOW &OPTIONAL REGION) [Function]

Makes and returns a Wallpaper window. When the LEFT or MIDDLE mouse button is pressed over Wallpaper windows the user is asked to select a tile size, then a position for the tile. Positions are selected with the LEFT button. Each tile is displayed in the window. If the user presses the MIDDLE button then the background of the current room is changed to be the current tile. The user can abort this process at any time by pressing the RIGHT mouse button.

There is a window type definition for Wallpaper windows so they may be saved in Suites.

(HACK-BACKGROUND SHADE &OPTIONAL ROOM) [Function]

Changes the first background shade specified for ROOM to be SHADE. If ROOM does not paint the background then this function adds a command to the background specification for ROOM which paints the whole screen with SHADE. ROOM defaults to the current room.

## Lyric Manual Test procedures

Filed as: {Eris}<Test>Admin>ManualManual.tedit

This document is a part of the procedures describing how to run tests on the Xerox Lisp Environment. The following is a list of the tests that must be run by hand. These are of two types. 1. Those that are run via the do-test software and thus automatically log errors. These are denoted by a .u extention. 2. Those that must be manually logged.

### Running interactive regression tests using do-test

1. Bring up the LISP.SYSOUT to be tested.
2. In an XCL executive window, load DO-TEST by typing (LOAD "{ERIS}<LISPCORE>INTERNAL>LIBRARY>DO-TEST.DFASL")
3. To run tests using DO-TEST, type (DO-ALL-TESTS :PATTERNS '("filenames") :RESULTS "{Eris}<Test>SubSystemName>SubSystemName.log")

For example, to run all the interactive tests for the debugger and put the results in a file named debugger.log, do the following:

```
(DO-ALL-TESTS
 :PATTERNS '("{Eris}<Test>Env>Debugger>Hand>*.u;")
 :RESULTS "{Eris}<Test>Env>Debugger>Logs>Debugger.log")
```

**Note the importance of using the terminating semicolon on test file names.** Not supplying the semicolon makes it run all versions of that file name!

See {eris}<lispcore>internal>library>do-test.tedit for the full list of features.

### Helpful utilities:

(IL:FILESLOAD WHO-LINE) ;each field is active to help change them.  
(IL:FILESLOAD FILEWATCH) ;use background menu to turn on a monitor of what files you have open

### SUCCESS / FAILURE prompt windows

Some tests will pop up a prompt window requesting certain actions and to look for certain results. At the top of the window are the words SUCCESS and FAILURE. If the results of the test are as prompted, select SUCCESS otherwise select FAILURE. You may move and shrink this window, but do not close it.

Before this window is popped up, you will be asked if you want a SHORT, MEDIUM, or LONG test and you will be presented with a menu. A short test will only test high priority items. LONG will test all cases.

### Reporting results

After running a test for a subsystem, shift-select the log file into a message and address it to: John Sybalsky, the appropriate developer, and the test writer.

Here is a lafite-form for this purpose:

Subject: Test results for >>subsystem name<<  
To: >>developer<<, >>test writer<<, >>documenter<<  
cc: Sybalsky.pa

>> test log <<

### LIST OF MODULES TO BE TESTED:

**EXEC - Tests written by John Park**

Location of Old Detailed Test Procedures: {eris}<test>env>exec>hand>test.proc  
 Approximate time to run test: < 1 hr.

Note:

You cannot use do-all-tests with the Exec test suite. You must instead:

(DO-TEST-FILE '{Eris}<Test>Env>exec>Hand>FOO.u)

where FOO is the name of each .u file in the directory.

These are automatic tests, which feed the input into exec via bksysbuf. Test results are automatically logged in {eris}<lispcore>test>exec>test.report, but you must make sure a new version of the file exists before you run the tests.

Be sure the exec is in the XCL-TEST package.

While running these tests you must not place the caret outside of the exec or do any other work while the tests are running.

*1108 Regression tests assigned to Norm Schuster, 3/18/87.*

### **DEBUGGER - Tests written by Kirk Kelley**

Relevant developer: Andy Daniels

(DO-ALL-TESTS

:PATTERNS '("{Eris}<Test>Env>Debugger>Hand>\*.u;")

:RESULTS "{Eris}<Test>Env>Debugger>Logs>Debugger.log")

Approximate time to run test: < 1 hr.

Note: These are a mixture of automatic tests and those that require interaction. Some of them intentionally pop up break windows since this tests break windows. If in the process of testing you get a break window that is not obviously part of the test, uparrow out of it (type an ↑ or use the one from the title menu) and the next test should appear.

These tests were generated (and run) on an 1108.

*Regression tests assigned to Masa Tateno, 3/18/87.*

### **DEDIT - Tests written by Henry Cate**

Location of test file: {eris}<test>env>DEdit>hand>\*.u

Location of log file: {eris}<test>env>DEdit>logs>DEdit.log

Approximate time to run test: < 1 hr.

*1108 Regression tests assigned to Albert Sahim, 3/18/87.*

### **DISPLAY - Tests written by Peter Reidy**

Location of test procedure: {eris}<test>i/o>display>hand>cursor.proc

Location of test source code (used by both cursor.proc and cursor.test:

{eris}<test>i/o>display>hand>cursor.test

Location of log file: {eris}<test>i/o>display>logs>cursor.log

### **HARDCOPY - Tests written by Peter Reidy**

Location of test plans: {erinyes}<test>lisp>lyric>plans>fx80driver.plan, 4045xlpstream.plan, press/interpress.plan

Location of test procedure files: {eris}<test>i/o>hardcopy>hand>fx80driver.proc, 4045xlpstream.proc, press/interpress.proc

Location of log files: {eris}<test>i/o>hardcopy>hand>fx80driver.log, 4045xlpstream.log, press/interpress.log

Location of test code: {eris}<test>i/o>hardcopy>hand>streamtests.u

Location of test cases: {eris}<lispcore>test>streams>

*Regression tests assigned to >>NAME<<, >>DATE<<.*

**KEYBOARD - Tests written by Henry Cate**

Location of test file: {eris}<test>i/o>Keyboard>hand>\*.u  
 (As of march 10th, there are 4 tests.)

Location of log file: {eris}<test>i/o>Keyboard>logs>Keyboard.log

Approximate time to run test: about 15 minutes.

*Regression tests assigned to Albert Sahim, 3/26/87.*

**PROGRAM ANALYSIS - Tests written by John Park  
(Subsystems: Masterscope, Databasefns, Browser, and Inspector)****Masterscope**

Location of test files: {eris}<test>env>program-analysis>hand>masterscope.u

Location of log file: {eris}<test>env>program-analysis>logs>masterscope.report

Approximate time to run test: 7 minutes.

**Databasefns**

Location of test files: {eris}<test>env>program-analysis>hand>databasefns.u

Location of log file: {eris}<test>env>program-analysis>logs>databasefns.report

Approximate time to run test: 3 minutes.

Note: The data file used by this test is in {eris}<test>env>program-analysis>hand>databasefns.data

**Browser**

Location of test files: {eris}<test>env>program-analysis>hand>browser-part1.u

{eris}<test>env>program-analysis>hand>browser-part2.u

Location of log file: {eris}<test>env>program-analysis>logs>browser.report

Approximate time to run test: 8 minutes.

Note: There are two test files for browser testing. Part 2 must be executed after Part 1.

The data file used by this test is in {eris}<test>env>program-analysis>hand>browser. graph

**Spy**

Location of test files: {eris}<test>env>program-analysis>hand>spy.u

Location of log file: {eris}<test>env>program-analysis>logs>spy.report

Approximate time to run test: 5 minutes.

**Inspector**

Location of test files: {eris}<test>env>program-analysis>hand>inspect.u

Location of log file: {eris}<test>env>program-analysis>logs>inspect.report

Approximate time to run test: 8 minutes.

Note: A fatal bug was discovered - Inspect never returns when \*random-state\* is inspected (AR # 8203).

*Regression tests assigned to Norm Schuster, 3/26/87.*

**PROGRAM SUPPORT - Tests written by John Park  
(Subsystems: DWIM, and PRETTYPRINT)****DWIM**

Location of test files: {eris}<test>env>program-support>hand>dwim.u

Location of log file: {eris}<test>env>program-support>logs>dwim.report

Approximate time to run test: 2 minutes.

Note: The DWIM test is executed by entering (DWIM-TEST). SEE the test file for more info.

**Printing Out Function Definitions (PRETTYPRINT)**

This test is covered by PP and other subsystems of Executative Test .

(See {eris}<test>env>exec>hand>pp.u and also see.u, see-without-comment, ty.u and type.u in the same directory)

**PROCESS CONTROLS (PSW) - Tests written by John Park**

Location of test files: {eris}<test>env>process-controls>hand>psw.u

Location of log file: {erinyes}<test>env>process-controls>logs>psw.report

Approximate time to run test: 2 - 10 minutes.

**SEdit - Tests written by Henry Cate**

For SEdit test, load TEdit. Then change packages with: (cl:in-package 'xcl-test)  
 For do-all-tests, it may work better if the semicolon is not used.

Location of test file: {eris}<test>env>code-editor>hand>\*.u  
 (As of march 2nd, there are 21 tests.)

Location of log file: {eris}<test>env>code-editor>logs>SEdit.log

Approximate time to run test: about a day.

*1108 Regression tests assigned to Albert Sahim, 3/18/87.*

**INSPECTOR - Tests written by Lois Lew**

Location of test files:{eris}<test>env>env>inspector>hand>inspect-allrec.tedit

{eris}<test>env>inspector>hand>inspect-defstruct.tedit

{eris}<test>env>inspector>hand>inspect-macro.tedit

{eris}<test>env>inspector>hand>inspectw.tedit

{eris}<test>env>inspector>hand>inspect-code.tedit

{eris}<test>env>inspector>hand>inspectfieldflg.tedit

Location of log file:create one at:

{eris}<test>env>inspector>logs>inspect-allrec.log

{eris}<test>env>inspector>logs>inspect-defstruct.log

{eris}<test>env>inspector>logs>inspect-macro.log

{eris}<test>env>inspector>logs>inspectw.log

{eris}<test>env>inspector>logs>inspect-code.log

{eris}<test>env>inspector>logs>inspectfieldflg.log

Approximate time to run test: ?

Note: use shift select to copy the material marked to be typed into the appropriate exec window.

*Regression tests assigned to Masa Tateno, 3/26/87.*



# Running AR Test Cases & Recording the Results

## The Pass and Fail commands

The "Pass" and "Fail" commands are defined on <Lispcore>Internal>Library>RELEASETOOLS. The form of use is

**Pass** AR#1 AR#2 AR#3 . . . ;Records the fact that AR Test Cases for the ARs named were run and succeeded.

**Fail** AR#4 AR#5 AR#6 . . . ; Records the fact that AR Test Cases for the ARs named were run and failed.

These two commands record each AR in a log file, along with your user name (so we know who ran it), and the current time (so we know when). These log entries will be used to

- 1) Track the current status of the AR test cases, and
- 2) Prepare the alpha-test log for Beta-test readiness review.

## Running a test

When you run an AR test case, please use the Pass or Fail command to record the fact!

# AR Test Case Status Summary

## As of >>Date<<

>>AR Test &  
Status<<

# Special Files on <Test>ARs>

## **AR-TEST-CASE.Auto-log;1**

This is the file where the "Pass" and "Fail" commands record the results of running an AR test case.  
See {Eris}<Test>Admin>Running-AR-Test-Cases.TEdit for details.

>><<  
>><<

>><<  
>><<

>><<  
>><<

>><<  
>><<

## **SEdit test report**

This report is for tests written and executed up to February 28, 1987 on the <Lyric>Basics>Full.Sysout generated 21-Jan-87.

The following tests are for the integration of the new error system into the Interlisp environment.

The test plan for this report is {Erinyes}<Test>Lisp>Lyric>Plans>SEdit.NoteFile

### **Groups of tests were written and executed on the 1186 for the following commands:**

INTERRUPTS

ABORT  
BASE  
COMMENT  
EVAL  
EXPAND  
EXTRACT  
FIND  
HELP  
JOIN  
MENU  
EDIT DEFINITION  
MUTATE  
PACKAGE  
PARENTHESIZE  
REDO  
SKIP-NEXT  
SUBSTITUTE  
UNDO

DELETE PREVIOUS WORD  
DONE  
REDISPLAY

These require user interaction. They are stored at {Eris}<Lispcore>Test>SEdit>\*.u

### **Still to be formally tested**

The mouse, (shift selection (copy, delete, move) to and from each source:

1. The same SEdit
2. A different SEdit
3. TEdit
4. Exec (each type/profile)

Window. Test each window operation with SEdit in each of stand selection/caret situations. SEdit should reappear in the same situation.

Completion. Make sure SEdit completes properly for each type of definition being edited.

1. def-types
2. fns
3. vars
4. property lists
5. records and datatypes.

Edit interface:

1. ed and editdef
2. df, dv, dc, dp
3. multiple edit drivers: editfromfile, editcallers, masterscope, etc...

## New ARs generated

7553 Not able to parenthesize part of an litatom  
 7604 SEdit, Meta-B does not work with decimals  
 7607 SEdit, problem with Base, change base and try placing edit caret, break  
 7624 SEdit, Attach Menu, come up with incorrect Print-Base  
  
 7629 SEdit, meta-E has trouble evaluating a number, breaks  
 7631 SEdit, meta-E, evaluates all, even when only part is selected  
 7642 SEdit, meta-E, something ought to happen with extended selection  
 7682 SEdit, control-W after single quote breaks  
 7688 SEdit, CASE on a string or litatom doesn't change the case  
  
 7699 SEdit, help, if nothing is selected, nothing happens, should give some feedback  
 7703 SEdit, meta-H, no response against a string  
 7705 SEdit, meta-H, break window when have an extended selection  
 7717 SEdit, meta-J, new, nothing selected, type meta-J, break window  
 7729 SEdit, documentation says prompt window, confusing  
  
 7731 SEdit, try meta-J against an extended selection of numbers, breaks  
 7733 SEdit, meta-J can join atoms and strings, different types  
 7783 SEdit, meta-O, optimizers, cann't use cl:optimize  
 7784 SEdit, meta-O, lets the user get started then says no way  
 7792 SEdit, meta-O some times the message overflows  
  
 7842 SEdit, meta-; documentation on old format, need il:  
 7851 SEdit, combination of meta-0, & meta-/ builds break window  
 7878 SEdit, meta-s, cann't substitue in extra items in a list  
 7879 SEdit, meta-S cann't substitute in a comment  
 7880 SEdit, meta-s, cann't substitute for an extended selection of a number  
  
 7889 SEdit-, meta-n give some feedback when don't do anything  
 7907 SEdit, meta-n, select vs extended selection give different results

## Debugger, Error System, and Unwinder test report

This report is for tests written and executed up to March 24, 1987 on the <Lyric>Basics>Full.Sysout generated 11-Mar-87.

The following tests are for the integration of the new error system into the Interlisp environment.

The source for this report is {Erinyes}<Test>Lisp>Lyric>Plans>Debugger.NoteFile

The print version of this test report is filed at {Eris}<Lispcore>Test>Debugger>Report.IP

The test plan is filed at {Erinyes}<Test>Lisp>Lyric>Plans>Debugger.NoteFile and IP.

Groups of tests were written and executed on the 1109 for all the commands in the Debugger document in the Xerox Common Lisp implementation notes.

These are stored on {eris}<Lispcore>Test>Debugger>BreakWindow.u

Groups of tests were written and executed on the 1109 for most of the functions in IRM chapter 15: Breaking, Tracing, and Advising.

These are mostly automatic tests but some require user interaction. They are stored in {Eris}<Lispcore>Test>Debugger>Debugger.u

For tests of the Xerox extensions to the CML error system, see {Eris}<Lispcore>cml>test>24-\*. {Eris}<Lispcore>cml>test>cl-error.x and {Eris}<Lispcore>cml>test>errorsystem.notefile which is the source for {Eris}<Lispcore>cml>test>24-errorsystem.x.

## Regression Test for ARS

### ARS tested 21-Jan-87 <Lyric>basics>full.sysout

7152 passed  
7780 new  
7797 new  
6503 passed

### New ARs generated

Several problems were discovered and reported as ARs. Each of these have tests. The following are only new ARs generated up to February 28.

#### DEBUGGER.NEWARS

7486 debugger "eval:" undocumented

7522 missing second param causes random error reporting

7601 unnamed proceed cases break compute-proceed-cases

7679 Version .01 Error System documentation edits

7780 argument names of broken fns unbound in debugger

7797 breaking/tracing advised fns does not update brokenfns

7845 exec il:settopval il:helpflag serious-condition attempt-to-change-constant

7848 (IL:NLSETQ (CL:SIGNAL 'ERROR)) breaks

7868 bad package fix readtable change screws up fix

7873: Common functions should be safe to break

AR 7923 il:brkinfolst no longer exists

AR 7908 untrace also unbreaks

AR 7919 TRACE no longer works for undefined subfunctions

AR 7932: (unbreak (sub-fn in super-fn)) has two problems

## Original List of ARs

### ERRORSYSTEM.ARSUMMARY

AR Summary generated on 17-Feb-87 10:33:56

Generated with Query Spec: (AND (Submitter: HAS Kelley))

Sorted with Sort Spec: (Status:)

| Numbe | Date:     | System:        | Subsystem:     | Status:    | Attn:                                                     | Subject:                                           | Priority: | Difficulty | Impact: |
|-------|-----------|----------------|----------------|------------|-----------------------------------------------------------|----------------------------------------------------|-----------|------------|---------|
| 6787  | 4-Nov-86  | Common Lisp    | File System In | New        | Jellinek.pa                                               | CL:OPEN says FILE NOT FOUND for BUSY FILE          |           |            |         |
| 6810  | 5-Nov-86  | Common Lisp    | Streams and I/ | New        | Jellinek                                                  | make-synonym-stream core file read fails           |           | Absolutely |         |
| 6847  | 10-Nov-86 | Common Lisp    | Other          | New        | vanMelle, W                                               | Making straight common-lisp text files using SEdit |           | Unlikely   |         |
| 6987  | 2-Dec-86  | Common Lisp    | Streams and I/ | New        | Jellinek                                                  | make-broadcast-stream should check for list arg    |           |            |         |
| 7033  | 9-Dec-86  | Programming En | Code Editor    | New        | Wozencraft                                                | SEdit global replace breaks                        |           |            |         |
| 7067  | 15-Dec-86 | Common Lisp    | Streams and I/ | New        | Jellinek                                                  | force-output should flush pages buffered in vmem   |           |            |         |
| 7118  | 29-Dec-86 | Programming En | Code Editor    | New        | woz                                                       | Sizing to fit SEdit window region length           |           | Perhaps    |         |
| 7168  | 8-Jan-87  | Common Lisp    | Other          | New        |                                                           | Portable DO-TEST needs expect-errors               |           |            |         |
| 7268  | 19-Jan-87 | Language Suppo | Storage Format | New        | vanmelle                                                  | SPELLFILE should use FILEDATES prop, but doesn't   |           |            |         |
| 7308  | 21-Jan-87 | Programming En | Break Package  | New        | Daniels                                                   | CL:READ should be in the list of break warning fns |           |            |         |
| 7436  | 4-Feb-87  | Communications | Other          | New        | LispCore↑.p                                               | Constant requests for passwords                    |           |            |         |
| 7439  | 4-Feb-87  | Common Lisp    | Error System   | New        | Biggs, Dani                                               | Error system documentation needs re-writing        |           |            |         |
| 7448  | 4-Feb-87  | Programming En | Code Editor    | New        | Wozencraft,                                               | SEdit looses edits of whole lists                  |           |            |         |
| 7451  | 4-Feb-87  |                | New            | Fischer    | Old-Interlisp-Exec comes up in current pkg&rdtbl          |                                                    |           |            |         |
| 7486  | 5-Feb-87  | Programming En | Break Package  | New        | Daniels                                                   | debugger "eval:" undocumented                      |           |            |         |
| 7522  | 6-Feb-87  |                | New            | Daniels    | CERROR missing second param causes random error reporting |                                                    |           |            |         |
| 7601  | 10-Feb-87 | Common Lisp    | Error System   | New        | Daniels                                                   | unnamed proceed cases break compute-proceed-cases  |           |            |         |
| 7679  | 12-Feb-87 | Common Lisp    | Error System   | New        | Daniels,Big                                               | Version .01 Error System documentation edits       |           |            |         |
| 7680  | 12-Feb-87 | Documentation  | Product Descr/ | New        | Biggs                                                     | WHO-LINE mention missing from overview             |           |            |         |
| 7681  | 12-Feb-87 | Text           | TEDIT          | New        | Syabalsky, S                                              | TEDIT not calling EDITBM ?                         |           |            |         |
| 7686  | 12-Feb-87 | Documentation  | Interlisp Refe | New        | Biggs,Sybal                                               | IL => CL function map needed                       |           |            |         |
| 7687  | 12-Feb-87 | Programming En | Code Editor    | New        | Wozencraft                                                | SEdit should reflect a change in readtable         |           |            |         |
| 7696  | 12-Feb-87 | Programming En | Code Editor    | New        | SeditSuppor                                               | SEdit quits refreshing after soft stack overflow   |           |            |         |
| 5717  | 27-May-86 | Windows and Gr | Window System  | Open       | Wozencraft.                                               | ATTACHWINDOW JUSTIFY should work for thin          |           |            |         |
| 6789  | 4-Nov-86  | Common Lisp    | Streams and I/ | Open       | Jellinek.pa                                               | make-concatenated-stream core file problem         |           |            |         |
| 6919  | 18-Nov-86 |                | Open           | Jellinek   | SETFILEINFO does not take FileName in all cases           |                                                    |           | Unlikely   |         |
| 4847  | 5-Dec-85  | Communications | Other          | Fixed      |                                                           | FileCache breaks when running init.firsttime       |           | Hopefully  |         |
| 7098  | 22-Dec-86 | Common Lisp    | Streams and I/ | Fixed      |                                                           | make-string-input-stream breaks on printed double  |           |            |         |
| 4890  | 10-Dec-85 | Operating Syst | Virtual Memory | Declined   |                                                           | Changing CPE and memory boards gives MP9335        |           |            |         |
| 6797  | 4-Nov-86  |                | Declined       |            |                                                           | KEYACTION does not work for MOVE.                  |           |            |         |
| 6799  | 5-Nov-86  | Common Lisp    | Streams and I/ | Declined   | Jellinek                                                  | make-string-input-stream calls OPENSTRINGSTREAM    |           |            |         |
| 7529  | 9-Feb-87  | Language Suppo | Stack and Inte | Incomplete |                                                           | il: fetch  dwimification problems                  |           |            |         |

### DEBUGGER.ARSUMMARY

AR Summary generated on 16-Feb-87 16:23:08

Generated with Query Spec: (AND (Subsystem: IS Break Package))

Sorted with Sort Spec: (Status:)

| Numbe | Date: | System: | Subsystem: | Status: | Attn: | Subject: | Priority: | Difficulty | Impact: |
|-------|-------|---------|------------|---------|-------|----------|-----------|------------|---------|
|-------|-------|---------|------------|---------|-------|----------|-----------|------------|---------|

|      |                       |                        |               |     |             |                                                    |            |
|------|-----------------------|------------------------|---------------|-----|-------------|----------------------------------------------------|------------|
| 6851 | 10-Nov-86             | Programming En         | Break Package | New | masinter    | If a u.d.f. has a functions definition, want the d |            |
|      | Absolutely            |                        | Feature       |     |             | efinition to be unsaved, ala Interlisp-D environme |            |
|      | 23:07:31              | vironment              |               |     |             |                                                    |            |
| 6981 | 1-Dec-86              | Programming En         | Break Package | New | Fischer.PA  | Trace Window Overflow                              | Hopefully  |
|      | Annoying Bug          |                        |               |     |             |                                                    |            |
|      | 08:56:50              | vironment              |               |     |             |                                                    |            |
| 6993 | 3-Dec-86              | Programming En         | Break Package | New | daniels     | Bug in debugger: Buttoning "display edit" from the |            |
|      | Hopefully Moderate    | Annoying Bug           |               |     |             | frame window edits the function cell               |            |
|      | 15:57:33              | vironment              |               |     |             |                                                    |            |
| 7016 | 6-Dec-86              | Programming En         | Break Package | New | jellinek, d | Want -> = commands for debugger                    | Unlikely   |
|      | Serious Design - Impl |                        |               |     |             |                                                    |            |
|      | 07:40:10              | vironment              |               |     |             |                                                    |            |
| 7079 | 17-Dec-86             | Programming En         | Break Package | New | daniels     | Debugger should skip SI::*UNWIND-PROTECT* frames   |            |
|      | i Hopefully Easy      | Annoying Bug           |               |     |             |                                                    |            |
|      | 13:26:10              | vironment              |               |     |             | n reporting errors                                 |            |
| 7084 | 17-Dec-86             | Programming En         | Break Package | New | daniels, va | BREAK..OK doesn't work under Interlisp exec/profil |            |
|      | Absolutely            | Bug                    |               |     |             |                                                    |            |
|      | 18:07:17              | vironment              |               |     |             | e--uses wrong evaluator, gets uba BROKEN           |            |
| 7085 | 17-Dec-86             | Programming En         | Break Package | New | daniels.pa  | UNBREAK (foo :in bar) doesn't find uses in subfunc |            |
|      | Absolutely Moderate   | Serious Bug            |               |     |             |                                                    |            |
|      | 20:36:10              | vironment              |               |     |             | tions                                              |            |
| 7089 | 19-Dec-86             | Programming En         | Break Package | New | daniels.pa  | OPENWP should be on list of unsafe functions to br |            |
|      | Absolutely Easy       | Fatal Bug              |               |     |             |                                                    |            |
|      | 10:23:56              | vironment              |               |     |             | eak                                                |            |
| 7097 | 19-Dec-86             | Programming En         | Break Package | New | Daniels     | Wrong frame current in broken function in debugger |            |
|      | Hopefully Moderate    | Moderate Bug           |               |     |             |                                                    |            |
|      | 22:36:05              | vironment              |               |     |             |                                                    |            |
| 7122 | 29-Dec-86             | Programming En         | Break Package | New | Daniels     | Break window BT frame window doesn't show arg name |            |
|      | Absolutely            | Bug                    |               |     |             |                                                    |            |
|      | 13:47:11              | vironment              |               |     |             | s for broken fn.                                   |            |
| 7213 | 13-Jan-87             | Programming En         | Break Package | New | daniels, pa | Trace replacing old fn defn.                       | Hopefully  |
|      | Serious Bug           |                        |               |     |             |                                                    |            |
|      | 02:10:08              | vironment              |               |     |             | vel                                                |            |
| 7215 | 13-Jan-87             | Programming En         | Break Package | New | daniels     | Arguments displayed during trace are random.       |            |
|      | Perhaps               | Moderate Feature       |               |     |             |                                                    |            |
|      | 02:16:43              | vironment              |               |     |             |                                                    |            |
| 7236 | 14-Jan-87             | Programming En         | Break Package | New | daniels     | Information in breakpoint, backtrace, and frame wi |            |
|      | Absolutely Moderate   | Moderate Bug           |               |     |             |                                                    |            |
|      | 16:07:14              | vironment              |               |     |             | ndow is printed with inconsistent package / readta |            |
|      |                       |                        |               |     |             | ble.                                               |            |
| 7267 | 16-Jan-87             | Programming En         | Break Package | New | woz         | Bad bahavior of backtrace window inspect menu      |            |
|      | Unlikely Easy         | Annoying Design - UI   |               |     |             |                                                    |            |
|      | 16:06:54              | vironment              |               |     |             |                                                    |            |
| 7295 | 20-Jan-87             | Programming En         | Break Package | New | Daniels, Bi | Need to finish documentation for the new DEBUGGER  |            |
|      | Absolutely Hard       | Serious Documentation  |               |     |             |                                                    |            |
|      | 17:45:26              | vironment              |               |     |             | ggs                                                |            |
| 7296 | 20-Jan-87             | Programming En         | Break Package | New | Fischer, Bi | New Step and trace need documentation              |            |
|      | Absolutely Moderate   | Serious Documentation  |               |     |             |                                                    |            |
|      | 17:48:23              | vironment              |               |     |             | ggs                                                |            |
| 7308 | 21-Jan-87             | Programming En         | Break Package | New | Daniels     | CL:READ should be in the list of break warning fns |            |
|      | Absolutely            | Design - UI            |               |     |             |                                                    |            |
|      | 16:14:01              | vironment              |               |     |             |                                                    |            |
| 7355 | 26-Jan-87             | Programming En         | Break Package | New | Pavel, Dani | Want better support for debugging interpreted code |            |
|      | Design - UI           |                        |               |     |             |                                                    |            |
|      | 23:27:01              | vironment              |               |     |             | els                                                |            |
| 7371 | 28-Jan-87             | Programming En         | Break Package | New | Daniels     | Stack overflow condition should get normal debugge |            |
|      | Absolutely            | Moderate Design - UI   |               |     |             |                                                    |            |
|      | 12:01:49              | vironment              |               |     |             | r window                                           |            |
| 7383 | 29-Jan-87             | Programming En         | Break Package | New | Daniels     | ENTER-DEBUGGER-P should say yes for STORAGE-       |            |
|      | CONDIT Hopefully Easy | Moderate Design - Impl |               |     |             |                                                    |            |
|      | 11:41:06              | vironment              |               |     |             | IONS                                               |            |
| 7384 | 29-Jan-87             | Programming En         | Break Package | New |             | STACK-OVERFLOW errors should get a new window      |            |
|      | Hopefully Easy        | Annoying Design - Impl |               |     |             |                                                    |            |
|      | 11:43:08              | vironment              |               |     |             |                                                    |            |
| 7402 | 30-Jan-87             | Programming En         | Break Package | New | Daniels     | Closing a debugger window does not always abort    |            |
|      | Moderate Bug          |                        |               |     |             |                                                    |            |
|      | 12:56:11              | vironment              |               |     |             |                                                    |            |
| 7441 | 4-Feb-87              | Programming En         | Break Package | New | Daniels     | Can't break Interlisp NLAMBDA's                    | Absolutely |
|      | Serious Bug           |                        |               |     |             |                                                    |            |
|      | 14:57:56              | vironment              |               |     |             |                                                    |            |
| 7445 | 4-Feb-87              | Programming En         | Break Package | New | Daniels     | INSPECTCODE from debugger should always inspect se |            |
|      | Hopefully             | Moderate Bug           |               |     |             |                                                    |            |
|      | 16:51:54              | vironment              |               |     |             | lected frame                                       |            |
| 7474 | 4-Feb-87              | Programming En         | Break Package | New | Daniels     | The debugger should rebind *READ-SUPPRESS* to NIL  |            |
|      | Absolutely Easy       | Serious Bug            |               |     |             |                                                    |            |
|      | 21:28:01              | vironment              |               |     |             |                                                    |            |
| 7486 | 5-Feb-87              | Programming En         | Break Package | New | Daniels     | debugger "eval:" undocumented                      |            |



|      |                        |                |                        |                      |             |                                                    |  |            |  |
|------|------------------------|----------------|------------------------|----------------------|-------------|----------------------------------------------------|--|------------|--|
|      | 12:00:08 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 7510 | 5-Feb-87               | Programming En | Break Package          | New                  | Daniels     | Debugger gives lexical environment to EVAL too oft |  |            |  |
|      | Absolutely             |                | Moderate Bug           |                      |             |                                                    |  |            |  |
|      | 18:22:32 vironment     |                |                        | en                   |             |                                                    |  |            |  |
| 7589 | 10-Feb-87              | Programming En | Break Package          | New                  | daniels     | Want !EVAL debugger command                        |  |            |  |
|      | Feature                |                |                        |                      |             |                                                    |  |            |  |
|      | 15:10:43 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 7592 | 10-Feb-87              | Programming En | Break Package          | New                  |             | HELPDEPTH no longer controls break depth           |  |            |  |
|      | 17:47:02 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 7742 | 13-Feb-87              | Programming En | Break Package          | New                  | Daniels     | ADVISE doesn't work on Common Lisp functions       |  |            |  |
|      | Absolutely             |                | Serious Bug            |                      |             |                                                    |  |            |  |
|      | 18:53:10 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 7748 | 13-Feb-87              | Programming En | Break Package          | New                  | Daniels     | Want better name for Breakpoint, tracing functions |  |            |  |
|      | Hopefully Easy         |                | Minor Feature          |                      |             |                                                    |  |            |  |
|      | 19:25:46 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 118  | 17-Mar-84              | Programming En | Break Package          | Open                 | Pavel       | Want EDIT command in debugger to work even when co |  |            |  |
|      | Perhaps Hard           |                | Moderate Feature       |                      |             |                                                    |  |            |  |
|      | 0:19:30 vironment      |                |                        | de is compiled.      |             |                                                    |  |            |  |
| 494  | 4-Apr-84               | Programming En | Break Package          | Open                 | Masinter    | HELPFLAG = BREAK! should cause breaks on every err |  |            |  |
|      | Absolutely             |                | Moderate Feature       |                      |             |                                                    |  |            |  |
|      | 11:24:03 vironment     |                |                        | or                   |             |                                                    |  |            |  |
| 1032 | 8-May-84               | Programming En | Break Package          | Open                 | Masinter    | CML: Want to trace & break macros, like "push, pop |  |            |  |
|      | Unlikely Moderate      |                | Annoying Feature       |                      |             |                                                    |  |            |  |
|      | 17:21:55 vironment     |                |                        | " etc                |             |                                                    |  |            |  |
| 3972 | 25-Jun-85              | Programming En | Break Package          | Open                 | Masinter.pa | Want to be able to trace variables at execution ti |  |            |  |
|      | Perhaps                |                | Moderate Feature       |                      |             |                                                    |  |            |  |
|      | 12:46:56 vironment     |                |                        | me                   |             |                                                    |  |            |  |
| 4164 | 1-Aug-85               | Programming En | Break Package          | Open                 | Biggs, Masi | Want extended documentation of the Break Package   |  |            |  |
|      | Absolutely Hard        |                | Annoying Documentation |                      |             |                                                    |  |            |  |
|      | 16:18:20 vironment     |                |                        | nter.pa              |             |                                                    |  |            |  |
| 5099 | 29-Jan-86              | Programming En | Break Package          | Open                 | Burton      | want shift selection from the backtrace menu       |  |            |  |
|      | Unlikely               |                | Minor Feature          |                      |             |                                                    |  |            |  |
|      | 10:06:54 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 6231 | 27-Jul-86              | Programming En | Break Package          | Open                 | Masinter    | EDIT in break window under Common Lisp won't edit  |  |            |  |
|      | Hopefully Moderate     |                | Serious Feature        |                      |             |                                                    |  |            |  |
|      | 10:29:22 vironment     |                |                        | the expression       |             |                                                    |  |            |  |
| 6542 | 29-Sep-86              | Programming En | Break Package          | Open                 | Masinter    | TRACE window doesn't hold when page is full        |  |            |  |
|      | Hopefully              |                | Annoying Feature       |                      |             |                                                    |  |            |  |
|      | 08:56:25 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 6959 | 25-Nov-86              | Programming En | Break Package          | Open                 | Masinter    | Can't advise ARGTYPE = 2 functions                 |  |            |  |
|      | Absolutely             |                | Fatal Bug              |                      |             |                                                    |  |            |  |
|      | 15:25:50 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 6960 | 25-Nov-86              | Programming En | Break Package          | Open                 | daniels.pa  | Want one-shot breakpoints                          |  | Unlikely   |  |
|      | Moderate Feature       |                |                        |                      |             |                                                    |  |            |  |
|      | 15:46:22 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 7364 | 27-Jan-87              | Programming En | Break Package          | Open                 | Daniels     | (TRACE) gives ("nil not a function")               |  |            |  |
|      | Serious Bug            |                |                        |                      |             |                                                    |  |            |  |
|      | 19:10:55 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 7386 | 29-Jan-87              | Programming En | Break Package          | Open                 |             | ADVISE AROUND uses u.d.f. IL:\SAFEAPPLY            |  |            |  |
|      | Absolutely             |                | Fatal Bug              |                      |             |                                                    |  |            |  |
|      | 13:01:11 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 7391 | 29-Jan-87              | Programming En | Break Package          | Open                 |             | Want !EVAL in debugger                             |  |            |  |
|      | Moderate Design - Impl |                |                        |                      |             |                                                    |  |            |  |
|      | 14:32:21 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 7618 | 11-Feb-87              | Programming En | Break Package          | Open                 | Daniels     | BREAK :IN broken                                   |  | Absolutely |  |
|      | Serious Design - Impl  |                |                        |                      |             |                                                    |  |            |  |
|      | 12:36:42 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 1034 | 8-May-84               | Programming En | Break Package          | Fixed                |             | TRACE of (C)EXPR*s fails to show arguments         |  |            |  |
|      | Perhaps Easy           |                | Serious Design - UI    |                      |             |                                                    |  |            |  |
|      | 17:24:30 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 1035 | 8-May-84               | Programming En | Break Package          | Fixed                |             | TRACEing shifts (TTYDISPLAYSTREAM) into trace wind |  |            |  |
|      | Absolutely Moderate    |                | Moderate Bug           |                      |             |                                                    |  |            |  |
|      | 17:25:02 vironment     |                |                        | ow                   |             |                                                    |  |            |  |
| 1036 | 8-May-84               | Programming En | Break Package          | Fixed                |             | Confusing appearance of TRACE's value-return line  |  |            |  |
|      | Hopefully Moderate     |                | Annoying Design - UI   |                      |             |                                                    |  |            |  |
|      | 17:25:59 vironment     |                |                        |                      |             |                                                    |  |            |  |
| 1122 | 18-May-84              | Programming En | Break Package          | Fixed                |             | change STKNARGS, STKARG to see args of Lambda-     |  |            |  |
|      | nosp Perhaps           |                | Moderate Design - UI   |                      |             |                                                    |  |            |  |
|      | 12:25:45 vironment     |                |                        | read (for break)     |             |                                                    |  |            |  |
| 1694 | 24-Jul-84              | Programming En | Break Package          | Fixed                |             | (BREAK (FOO IN BAR)) replaces recordname FOO with  |  |            |  |
|      | Unlikely Hard          |                | Moderate Bug           |                      |             |                                                    |  |            |  |
|      | 10:04:53 vironment     |                |                        | FOO-IN-BAR in CREATE |             |                                                    |  |            |  |
| 1970 | 28-Aug-84              | Programming En | Break Package          | Fixed                |             | ↑B while a traced function is being executed confu |  |            |  |
|      | Absolutely             |                | Moderate Design - UI   |                      |             |                                                    |  |            |  |
|      | 09:46:40 vironment     |                |                        | ses break package    |             |                                                    |  |            |  |
| 2478 | 19-Oct-84              | Programming En | Break Package          | Fixed                |             | RETFROM inside function doesn't unwind break windo |  |            |  |
|      | Absolutely             |                | Annoying Bug           |                      |             |                                                    |  |            |  |
|      | 16:47:01 vironment     |                |                        | ws                   |             |                                                    |  |            |  |
| 2686 | 14-Nov-84              | Programming En | Break Package          | Fixed                |             | Repeatable Race break package: causes Break window |  |            |  |
|      | Absolutely             |                | Moderate Bug           |                      |             |                                                    |  |            |  |

|          |                    |                |                               |            |         |                                                    |  |            |
|----------|--------------------|----------------|-------------------------------|------------|---------|----------------------------------------------------|--|------------|
| 20:39:12 | vironment          |                |                               |            |         | acting like top-level typescript window            |  |            |
| 5556     | 24-Apr-86          | Programming En | Break Package                 | Fixed      |         | AUTOBACKTRACEFLG vs. TRACE                         |  |            |
|          | Absolutely         |                | Bug                           |            |         |                                                    |  |            |
| 11:48:47 | vironment          |                |                               |            |         |                                                    |  |            |
| 5807     | 11-Jun-86          | Programming En | Break Package                 | Fixed      |         | AUTOBACKTRACEFLG = ALWAYS breaks TRACE             |  |            |
|          | Absolutely         |                | Serious Bug                   |            |         |                                                    |  |            |
| 09:38:52 | vironment          |                |                               |            |         |                                                    |  |            |
| 5940     | 28-Jun-86          | Programming En | Break Package                 | Fixed      | biggs   | BT in break windows isn't very useful.             |  | Absolutely |
|          | Moderate           |                | Design - Impl                 |            |         |                                                    |  |            |
| 02:06:18 | vironment          |                |                               |            |         |                                                    |  |            |
| 6007     | 8-Jul-86           | Programming En | Break Package                 | Fixed      |         | Break handling under CMLEXEC should be CMLreading  |  |            |
|          | Absolutely         |                | Annoying Design - Impl        |            |         |                                                    |  |            |
| 11:03:28 | vironment          |                |                               |            |         |                                                    |  |            |
| 6128     | 19-Jul-86          | Programming En | Break Package                 | Fixed      |         | Want to be able to include 1-cell arglist debuggin |  | Absolutely |
|          | Design - Impl      |                |                               |            |         |                                                    |  |            |
| 04:17:22 | vironment          |                |                               |            |         | g info in compiled code, have printcode etc know a |  |            |
|          |                    |                |                               |            |         | bout them                                          |  |            |
| 6445     | 8-Sep-86           | Programming En | Break Package                 | Fixed      |         | Can't revert from the menu any more                |  | Absolutely |
|          | Moderate           |                | Bug                           |            |         |                                                    |  |            |
| 17:57:29 | vironment          |                |                               |            |         |                                                    |  |            |
| 6712     | 27-Oct-86          | Programming En | Break Package                 | Fixed      |         | Break package obscures system variables            |  |            |
|          | Absolutely         |                | Moderate Bug                  |            |         |                                                    |  |            |
| 09:44:31 | vironment          |                |                               |            |         |                                                    |  |            |
| 6796     | 4-Nov-86           | Programming En | Break Package                 | Fixed      |         | EVAL in debugger doesn't print result              |  | Absolutely |
|          | Moderate           |                | Design - Impl                 |            |         |                                                    |  |            |
| 17:59:31 | vironment          |                |                               |            |         |                                                    |  |            |
| 6817     | 6-Nov-86           | Programming En | Break Package                 | Fixed      |         | Backtrace windows (and printed backtraces) show to |  |            |
|          | Hopefully          |                | Annoying Design - UI          |            |         |                                                    |  |            |
| 12:33:52 | vironment          |                |                               |            |         |                                                    |  |            |
| 6818     | 6-Nov-86           | Programming En | Break Package                 | Fixed      |         | Stack frame display for CL EVAL frames should show |  |            |
|          | Hopefully          |                | Moderate Annoying Feature     |            |         |                                                    |  |            |
| 12:40:05 | vironment          |                |                               |            |         |                                                    |  |            |
| 6927     | 19-Nov-86          | Programming En | Break Package                 | Fixed      |         | debugger windows pop up on top of one another.     |  |            |
|          | Absolutely         |                | Annoying Design - UI          |            |         |                                                    |  |            |
| 11:54:28 | vironment          |                |                               |            |         |                                                    |  |            |
| 6952     | 24-Nov-86          | Programming En | Break Package                 | Fixed      |         | PAGEHEIGHT remains debugger window height after de |  |            |
|          | Absolutely         |                | Annoying Bug                  |            |         |                                                    |  |            |
| 12:01:36 | vironment          |                |                               |            |         |                                                    |  |            |
| 7121     | 29-Dec-86          | Programming En | Break Package                 | Fixed      |         | ?= under breakpoint shows broken fn, rather than a |  |            |
|          | Absolutely         |                | Bug                           |            |         |                                                    |  |            |
| 13:44:59 | vironment          |                |                               |            |         |                                                    |  |            |
| 7155     | 7-Jan-87           | Programming En | Break Package                 | Fixed      | Fischer | Debugger BTV! command :print-junk option broken    |  |            |
|          | Absolutely         |                | Easy Annoying Bug             |            |         |                                                    |  |            |
| 11:23:37 | vironment          |                |                               |            |         |                                                    |  |            |
| 7329     | 23-Jan-87          | Programming En | Break Package                 | Fixed      |         | Help info on debugger menu is "NIL"                |  | Hopefully  |
|          | Annoying           |                | Bug                           |            |         |                                                    |  |            |
| 18:09:49 | vironment          |                |                               |            |         |                                                    |  |            |
| 7598     | 10-Feb-87          | Programming En | Break Package                 | Fixed      |         | ↑ out of break under "EVAL" unwinds too far        |  |            |
|          | Absolutely         |                | Serious Design - Impl         |            |         |                                                    |  |            |
| 19:18:26 | vironment          |                |                               |            |         |                                                    |  |            |
| 7599     | 10-Feb-87          | Programming En | Break Package                 | Fixed      |         | "PROCEED" shows too much, starts at wrong frame    |  |            |
|          | Absolutely         |                | Moderate Design - Impl        |            |         |                                                    |  |            |
| 19:22:56 | vironment          |                |                               |            |         |                                                    |  |            |
| 6237     | 28-Jul-86          | Programming En | Break Package                 | Closed     |         | Change break package uses a special variable to *  |  |            |
|          | Easy               |                | Annoying Design - Impl        |            |         |                                                    |  |            |
| 14:01:45 | vironment          |                |                               |            |         |                                                    |  |            |
|          |                    |                |                               |            |         | WINDOW-BREAK* from WBREAK to decide whether to sta |  |            |
|          |                    |                |                               |            |         | rt a new window or to use the same one instead of  |  |            |
|          |                    |                |                               |            |         | MOVDs when WBREAK(T) or WBREAK(NIL)                |  |            |
| 2347     | 8-Oct-84           | Programming En | Break Package                 | Declined   |         | Break of (REPLACE IN FOO) gives erroneous error me |  |            |
|          | Perhaps            |                | Annoying Design - UI          |            |         |                                                    |  |            |
| 14:23:41 | vironment          |                |                               |            |         |                                                    |  |            |
| 4102     | 17-Jul-85          | Programming En | Break Package                 | Declined   |         | ssage (REPLACE was undefined)                      |  |            |
|          | Annoying           |                | Design - UI                   |            |         | want ~ to be treated equivalent to ↑ in break wind |  | Perhaps    |
| 19:13:39 | vironment          |                |                               |            |         |                                                    |  |            |
| 744      | 17-Apr-84          | Programming En | Break Package                 | Superseded |         | (Superseded by AR 1035) ↑ out of break can switch  |  |            |
|          | Perhaps            |                | Moderate Annoying Design - UI |            |         |                                                    |  |            |
| 16:25:26 | vironment          |                |                               |            |         |                                                    |  |            |
| 1740     | 2-Aug-84           | Programming En | Break Package                 | Superseded |         | TTY stream to TRACE window                         |  |            |
|          | Moderate           |                | Bug                           |            |         | editing under a break (superseded by AR 162)       |  |            |
| 14:00:53 | vironment          |                |                               |            |         |                                                    |  |            |
| 2809     | 6-Dec-84           | Programming En | Break Package                 | Superseded |         | (Superseded by AR 5556) Setting                    |  |            |
|          | AUTOBACKTRACEFLG = |                | Perhaps                       |            |         |                                                    |  |            |
| 09:56:17 | vironment          |                |                               |            |         |                                                    |  |            |
|          |                    |                |                               |            |         | ALWAYS causes a break when a traced fn is called:  |  |            |
|          |                    |                |                               |            |         | "TRACE - UNBOUND ATOM"                             |  |            |
| 2863     | 12-Dec-84          | Programming En | Break Package                 | Superseded |         | Harmony: tracing functions (superceded by AR       |  |            |
|          | 2863)              |                |                               |            |         |                                                    |  |            |
| 16:10:44 | vironment          |                |                               |            |         |                                                    |  |            |
| 2967     | 29-Dec-84          | Programming En | Break Package                 | Superseded |         | (Superseded by AR 1035) BREAK windows can be left  |  |            |
|          | Unlikely           |                | Minor Bug                     |            |         |                                                    |  |            |
| 08:52:57 | vironment          |                |                               |            |         | on screen after REVERT to broken fn.               |  |            |

|      |           |                        |                        |              |                                                    |
|------|-----------|------------------------|------------------------|--------------|----------------------------------------------------|
| 3487 | 14-Mar-85 | Programming En         | Break Package          | Superseded   | (Superseded by AR 1035) Display stream switched to |
|      | 09:13:59  | Perhaps                | Annoying Design - Impl |              |                                                    |
|      |           | vironment              |                        |              | break window when HELPFLAG set to break!           |
| 3540 | 25-Mar-85 | Programming En         | Break Package          | Superseded   | (Superseded by AR5556) When                        |
|      | 08:39:38  | AUTOBACKTRACEFLG=ALWAY | Hopefully              | Moderate Bug |                                                    |
|      |           | vironment              |                        |              | SI, any TRACEd function stops with UNBOUND ATOM TR |
|      |           |                        | ACE                    |              |                                                    |
| 5219 | 14-Feb-86 | Programming En         | Break Package          | Superseded   | superseded by AR494: HELPFLAG = BREAK! doesn't     |
|      | 11:41:33  | alw Absolutely         | Moderate Bug           |              |                                                    |
|      |           | vironment              |                        |              | ays work                                           |
| 6099 | 16-Jul-86 | Programming En         | Break Package          | Superseded   | (Superseded by AR 5556) AUTOBACKTRACEFLG of        |
|      | 17:46:31  | ALWAYS Perhaps         | Annoying Bug           |              |                                                    |
|      |           | vironment              |                        |              | or ALWAYS! breaks TRACE                            |
| 7011 | 6-Dec-86  | Programming En         | Break Package          | Superseded   | debugger entry frame odd Absolutely                |
|      | 06:53:47  | Serious Design - UI    |                        |              |                                                    |
|      |           | vironment              |                        |              |                                                    |
| 7083 | 17-Dec-86 | Programming En         | Break Package          | Superseded   | [SUPERSEDED BY 7121] != doesn't work for il:broken |
|      | 18:01:11  | Absolutely             | Bug                    |              |                                                    |
|      |           | vironment              |                        |              | fns                                                |
| 7247 | 15-Jan-87 | Programming En         | Break Package          | Superseded   | Superseded by AR7236: Frame inspector window in De |
|      | 19:16:18  | Absolutely Moderate    | Serious Bug            |              |                                                    |
|      |           | vironment              |                        |              | bugger always uses IL package, readtable           |
| 311  | 27-Mar-84 | Programming En         | Break Package          | Obsolete     | HELPPFIX is calling EDITE with Type=FNS when there |
|      | 9:29:34   | Absolutely Easy        | Annoying Design - UI   |              |                                                    |
|      |           | vironment              |                        |              | is no fn                                           |
| 2223 | 21-Sep-84 | Programming En         | Break Package          | Obsolete     | BREAK frame window opens twice Perhaps             |
|      | 15:09:15  | Annoying Design - Impl |                        |              |                                                    |
|      |           | vironment              |                        |              |                                                    |
| 2533 | 30-Oct-84 | Programming En         | Break Package          | Obsolete     | Break window becomes Toplevel                      |
|      | 09:41:33  | Moderate Bug           |                        |              |                                                    |
|      |           | vironment              |                        |              |                                                    |
| 2598 | 7-Nov-84  | Programming En         | Break Package          | Obsolete     | Got UNBOUND ATOM "↑" and "?=" in break window      |
|      | 17:07:01  | Perhaps                | Annoying Bug           |              |                                                    |
|      |           | vironment              |                        |              |                                                    |
| 2614 | 8-Nov-84  | Programming En         | Break Package          | Obsolete     | MaxBkMenuHeight no longer effective Perhaps        |
|      | 14:55:36  | Minor Bug              |                        |              |                                                    |
|      |           | vironment              |                        |              |                                                    |
| 2928 | 19-Dec-84 | Programming En         | Break Package          | Obsolete     | Break window frame inspect window opened twice whe |
|      | 08:09:37  | Perhaps                | Minor Performance      |              |                                                    |
|      |           | vironment              |                        |              | n first created                                    |
| 2968 | 29-Dec-84 | Programming En         | Break Package          | Obsolete     | REVERT to Nlambda-nospread with LOCALVAR arg       |
|      | 10:47:57  | trans Absolutely       | Moderate Bug           |              |                                                    |
|      |           | vironment              |                        |              | forms arg to (LIST arg)                            |
| 6538 | 26-Sep-86 | Programming En         | Break Package          | Obsolete     | TRACE keeps opening new windows, rather than reusi |
|      | 18:23:55  | Absolutely             | Serious Bug            |              |                                                    |
|      |           | vironment              |                        |              | ng one trace window.                               |
| 6829 | 6-Nov-86  | Programming En         | Break Package          | Obsolete     | CLOSEW of a breakwindow sends ↑ instead of il:↑    |
|      | 21:34:57  | Minor Bug              |                        |              |                                                    |
|      |           | vironment              |                        |              |                                                    |
| 539  | 6-Apr-84  | Programming En         | Break Package          | Incomplete   | Break middle-button pop-up menu should switch the  |
|      | 10:20:11  | Perhaps                | Annoying Design - UI   |              |                                                    |
|      |           | vironment              |                        |              | TTY to the break window                            |

## **DEdit test report**

This report is for tests written and executed up to February 28, 1987 on the <Lyric>Basics>Full.Sysout generated 21-Jan-87.

The test plan for this report is {Erinyes}<Test>Lisp>Lyric>Plans>DEdit.tedit.

### **Groups of tests were written and executed on the 1186 for the following commands:**

After, Before, Delete, Replace, SWitch, (), () out, Undo, Find, Swap, Reprint, Edit, EditCom, Break, Eval, Exit.

These require user interaction. They are stored at {Eris}<Lispcore>Test>DEDIT>high-level.u

### **New ARs generated**

None

Scripts for testing the record package (assuming currently in **Common Lisp User** package)

## Record type - Record

**type:**

```
(il:record record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (oddp (length il:datum))))
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some string"))
```

**type:**

```
(inspect record-test-record)
```

**choose "inspect"**

*--should produce a window with three numbered elements 1 (a b c), 2 "some string", 3 nil*

```
((a b c) some string
```

```
1 (a b c)
2 "some string"
3 nil
```

**type:**

```
(inspect record-test-record)
```

**choose "as a record"**

**choose "record-test-name"**

*--should produce a window with four elements where a and alpha have the same value.*

```
((a b c) some string nil)
a (a b c)
gamma nil
bravo "some string"
alpha (a b c)
```

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window identical to the one in the previous step.*

**type:**

```
(ed 'record-test-name 'records)
```

**delete gamma from the element list.**

**type cntl-x to save edit.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with three elements where a and alpha have the same value and there is no gamma.*

**return to the edit window and type gamma back in where it belongs.**

**type cntl-x to save again.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list*

**close the edit window**

**type:**

```
(in-package 'interlisp)
```

**type:**

```
(inspect user::record-test-record 'user::record-test-name)
```

--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.

```
((a b c) some string nil) Inspector
user::a (user::a user::b user::c)
user::gamma nil
user::bravo "some string"
user::alpha (user::a user::b user::c)
```

**type:**

```
(cl:in-package 'user)
```

**type:**

```
(il:replace (record-test-name gamma) il:of record-test-record il:with '(a (b (c (d (e (f
(g)))))))
(inspect record-test-record)
```

**select inspect**

--should get:

```
((a b c) some string (
1 (a b c)
2 "some string"
3 (a (b #))
```

**type:**

```
(setq il:inspectprintlevel '(5 . 5))
(inspect record-test-record)
```

**select inspect**

--should get:

```
((a b c) some string (
1 (a b c)
2 "some string"
3 (a (b (c (d (
```

**type:**

```
(setq il:inspectprintlevel '(5 . 1))
(inspect record-test-record)
```

**select inspect**

--should get:

```
((a b c) some string (
1 (a ...)
2 "some string"
3 (a ...)
```

**type:**

```
(setq il:inspectprintlevel '(2 . 5))
```

---

## Record type -Typerecord

**type:**

```
(il:typerecord record-test-name (alpha bravo gamma) (il:synonym alpha a))
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some
```

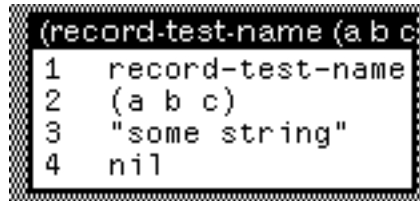
```
string"))
```

**type:**

```
(inspect record-test-record)
```

**choose "inspect"**

*--should produce a window with four numbered elements 1 record-test-name 2 (a b c), 3 "some string", 4 nil*



```
(record-test-name (a b c)
 1 record-test-name
 2 (a b c)
 3 "some string"
 4 nil)
```

**type:**

```
(setq il:maxinspectcdrlevel 2)
(inspect record-test-record)
```

**choose "inspect"**

*--should produce a window with two numbered elements similar to the previous step and the elements number 3 and 4 in a list labeled il:&&*

**type:**

```
(setq il:maxinspectcdrlevel 50)
```

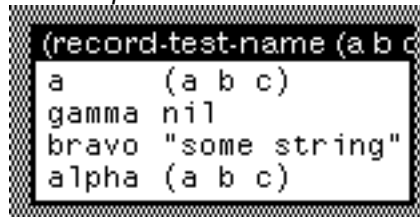
**type:**

```
(inspect record-test-record)
```

**choose "as a record"**

**choose "record-test-name"**

*--should produce a window with four elements where a and alpha have the same value.*



```
(record-test-name (a b c)
 a (a b c)
 gamma nil
 bravo "some string"
 alpha (a b c))
```

**type:**

```
(inspect record-test-record)
```

**choose "as record-test-name"**

*--should produce a window identical to the one in the previous step.*

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window identical to the one in the previous step.*

**type:**

```
(ed 'record-test-name 'records)
```

**delete gamma from the element list.**

**type cntl-x to save edit.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with three elements where a and alpha have the same value and there is no gamma.*

**return to the edit window and type gamma back in where it belongs.**

**type cntl-x to save again.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list*

**close the edit window**

**type:**

```
(in-package 'interlisp)
```

**type:**

```
(inspect user::record-test-record 'user::record-test-name)
```

*--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.*

```
(record-test-name (a b c) some string nil) Ins
user::a (user::a user::b user::c)
user::gamma nil
user::bravo "some string"
user::alpha (user::a user::b user::c)
```

**type:**

```
(cl:in-package 'user)
```

## Record type - Proprecord

**type:**

```
(il:proprecord record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (evenp (length
il:datum))))
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some
string"))
```

**type:**

```
(inspect record-test-record)
```

**choose "inspect"**

*--should produce a window with four numbered elements 1 alpha, 2 (a b c), 3 bravo, 4 "some string".*

```
(alpha (a b c) bravo some string)
1 alpha
2 (a b c)
3 bravo
4 "some string"
```

**type:**

```
(inspect record-test-record)
```

**choose "as a PLIST"**

*--should produce a window with two elements, alpha and bravo with their associated values.*

```
(alpha (a b c) bravo some string)
alpha (a b c)
bravo "some string"
```

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with four elements where a and alpha have the same value.*



```
(alpha (a b c) bravo some string)
a (a b c)
alpha (a b c)
bravo "some string"
gamma nil
```

**type:**

```
(ed 'record-test-name 'records)
```

**delete gamma from the element list.**

**type cntl-x to save edit.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with three elements where a and alpha have the same value and there is no gamma.*

**return to the edit window and type gamma back in where it belongs.**

**type cntl-x to save again.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list*

**close the edit window**

**type:**

```
(in-package 'interlisp)
```

**type:**

```
(inspect user::record-test-record 'user::record-test-name)
```

*--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.*

```
(alpha (a b c) bravo some string) Inspector
user::a (user::a user::b user::c)
user::alpha (user::a user::b user::c)
user::bravo "some string"
user::gamma nil
```

**type:**

```
(cl:in-package 'user)
```

## Record type - Datatype

**type:**

```
(il:datatype record-test-name (alpha bravo gamma) (il:synonym alpha a))
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some string"))
```

**type:**

```
(inspect record-test-record)
```

**choose "inspect"**

*--should produce a window with four elements where a and alpha have the same value.*

```
#<record-test-name @
a (a b c)
alpha (a b c)
bravo "some string"
gamma nil
```

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window identical to the one in the previous step.*

**type:**

```
(in-package 'interlisp)
```

**type:**

```
(inspect user::record-test-record 'user::record-test-name)
```

*--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.*

```
#<record-test-name @ 375,74770> Inspect
user::a (user::a user::b user::c)
user::alpha (user::a user::b user::c)
user::bravo "some string"
user::gamma nil
```

**type:**

```
(cl:in-package 'user)
```

## Record type - Arrayrecord

**type:**

```
(il:arrayrecord record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (cond
(il:datum t))))
(setq record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some
string"))
```

**type:**

```
(inspect record-test-record)
```

*--should produce a window with three numbered elements 1 (a b c), 2 "some string", 3 nil*

```
#<arrayp @ 377,11
1 (a b c)
2 "some string"
3 nil
```

**type:**

```
(setq il:maxinspectarraylevel 2)
(inspect record-test-record)
```

*--should produce a window similar to the previous one but with only two elements showing.*

**type:**

```
(setq il:maxinspectarraylevel 300)
```

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with four elements where a and alpha have the same value.*

```
#<arrayp @ 377,11514>
a (a b c)
gamma nil
bravo "some string"
alpha (a b c)
```

**type:**

```
(ed 'record-test-name 'records)
```

**delete gamma from the element list.**

**type cntl-x to save edit.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with three elements where a and alpha have the same value and there is no gamma.*

**return to the edit window and type gamma back in where it belongs.**

**type cntl-x to save again.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list*

**close the edit window**

**type:**

```
(in-package 'interlisp)
```

**type:**

```
(inspect user::record-test-record 'user::record-test-name)
```

*--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.*

```
#<arrayp @ 377,11514> Inspector
user::a (user::a user::b user::c)
user::gamma nil
user::bravo "some string"
user::alpha (user::a user::b user::c)
```

**type:**

```
(cl:in-package 'user)
```

## Record type - Assocrecord

**type:**

```
(il:assocrecord record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (not (atom
(car il:datum))))))
(setf record-test-record (il:create record-test-name alpha il:_ '(a b c) bravo il:_ "some
string"))
```

**type:**

```
(inspect record-test-record)
```

**choose "inspect"**

*--should produce a window with two numbered elements 1 (alpha a b c), 2 (bravo . "some string")*

```
((alpha a b c) (bravo . some string))
1 (alpha a b c)
2 (bravo . "some string")
```

**type:**

```
(inspect record-test-record)
```

**choose "as an ALIST"**

*--should produce a window with two elements, alpha and bravo and their associated values.*

```
((alpha a b c) (bravo . some string))
alpha (a b c)
bravo "some string"
```

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with four elements and a has the same value as alpha.*

```
((alpha a b c) (bravo . some string))
a (a b c)
alpha (a b c)
bravo "some string"
gamma nil
```

**type:**

```
(ed 'record-test-name 'records)
```

**delete gamma from the element list.**

**type cntl-x to save edit.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with three elements where a and alpha have the same value and there is no gamma.*

**return to the edit window and type gamma back in where it belongs.**

**type cntl-x to save again.**

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window with four elements where a and alpha have the same value and gamma is returned to the list*

**close the edit window**

**type:**

```
(in-package 'interlisp)
```

**type:**

```
(inspect user::record-test-record 'user::record-test-name)
```

*--should produce a window with four elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.*

```
((alpha a b c) (bravo . some string)) Inspecto
user::a (user::a user::b user::c)
user::alpha (user::a user::b user::c)
user::bravo "some string"
user::gamma nil
```

**type:**  
 (cl:in-package 'user)

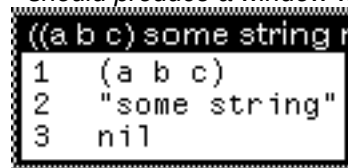
## Record type - Accessfns

**type:**  
 (IL:ACCESSFNS RECORD-TEST-NAME  
 ((ALPHA (CAR IL:DATUM)  
 (SETQ IL:DATUM (CONS IL:NEWVALUE  
 (CDR IL:DATUM)))))  
 (BRAVO (CADR IL:DATUM)  
 (SETQ IL:DATUM (CONS (CAR IL:DATUM)  
 (CONS IL:NEWVALUE  
 (CDDR IL:DATUM)))))  
 (GAMMA (CADDR IL:DATUM)  
 (SETQ IL:DATUM (LIST (CAR IL:DATUM)  
 (CADR IL:DATUM)  
 IL:NEWVALUE))))  
 (IL:CREATE (LIST '(a b c) "some string" NIL))  
 (IL:TYPE? (ODDP (LENGTH IL:DATUM))))  
 (setq record-test-record (il:create record-test-name)))

**type:**  
 (inspect record-test-record)

### choose "inspect"

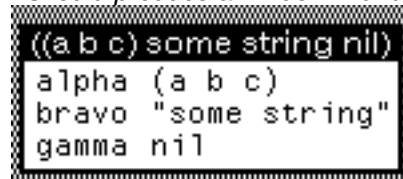
--should produce a window with three numbered elements 1 (a b c), 2 "some string". 3 ,nil



```
((a b c) some string nil)
1 (a b c)
2 "some string"
3 nil
```

**type:**  
 (inspect record-test-record 'record-test-name)

--should produce a window with three elements: alpha, bravo and gamma with their associated values.



```
((a b c) some string nil)
alpha (a b c)
bravo "some string"
gamma nil
```

**type:**  
 (ed 'record-test-name 'records)

**delete the gamma entry from the element list.**

**type cntl-x to save edit.**

**type:**  
 (inspect record-test-record 'record-test-name)  
 --should produce a window with two elements where there is no gamma.

**return to the edit window and type the gamma entry back in where it belongs.**

**type cntl-x to save again.**

**type:**  
 (inspect record-test-record 'record-test-name)  
 --should produce a window with three elements where gamma is returned to the list

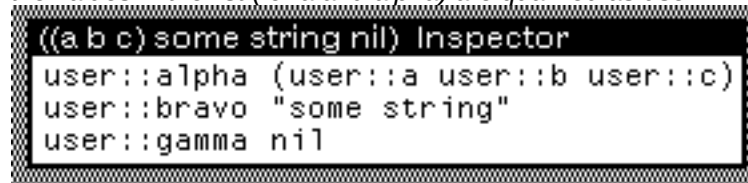
**close the edit window**

**type:**  
 (in-package 'interlisp)

**type:**

```
(inspect user::record-test-record 'user::record-test-name)
```

*--should produce a window with three elements where user::a and user::alpha have the same value. All the values in the list (for a and alpha) are qualified as user.*



```
((a b c) some string nil) Inspector
user::alpha (user::a user::b user::c)
user::bravo "some string"
user::gamma nil
```

**type:**

```
(cl:in-package 'user)
```

## Record type - Blockrecord

**type:**

```
(IL:DATATYPE RECORD-TEST-NAME
 ((ALPHA IL:POINTER)) alpha il:_ '(a b c))
(IL:BLOCKRECORD RECORD-TEST-NAME1
 ((BRAVO IL:WORD) (GAMMA IL:WORD)))
(SETQ RECORD-TEST-RECORD (IL:CREATE RECORD-TEST-NAME))
```

**type:**

```
(inspect record-test-record)
```

*--should produce a window with alpha and the list (a b c)*



```
#<record-test-r
alpha (a b c)
```

**type:**

```
(inspect record-test-record 'record-test-name)
```

*--should produce a window identical to the one in the previous step.*

**type:**

```
(inspect record-test-record 'record-test-name1)
```

*--should produce a window with bravo and gamma, each with two numbers*



```
#<record-tes
bravo 254
gamma 39654
```

**type:**

```
(ed 'record-test-name1 'records)
```

**delete the gamma entry from the element list.**

**type cntl-x to save edit.**

**type:**

```
(inspect record-test-record 'record-test-name1)
```

*--should produce a window with one element where there is no gamma.*

**return to the edit window and type the gamma entry back in where it belongs.**

**type cntl-x to save again.**

**type:**

```
(inspect record-test-record 'record-test-name1)
```

*--should produce a window with two elements where gamma is returned to the list*

close the edit window

**type:**

```
(in-package 'interlisp)
```

**type:**

```
(inspect user::record-test-record 'user::record-test-name1)
```

*--should produce a window with two elements where .All the values in the list are qualified as user.*

```
#<record-test-name
user::bravo 254
user::gamma 39654
```

**type:**

```
(cl:in-package 'user)
```

## Record type - Subrecords

**type:**

```
(il:record record-test-name (alpha bravo gamma) (il:synonym alpha a) (il:type? (oddp (length
il:datum))))
(il:record record-test-name1 (xray zebra record-test-name)
(il:subrecord record-test-name))
(setq record-test-record (il:create record-test-name1 alpha il:_ '(a b c) bravo il:_ "some
string" zebra il:_ "hi"))
```

**type:**

```
(inspect record-test-record)
```

**choose "inspect"**

*--should produce a window with three numbered elements 1 nil 2 "hi" and 3 the list ((a b c) "some string" nil)*

```
(nil hi ((a b c) some string nil)) Inspector
1 nil
2 "hi"
3 ((a b c) "some string" nil)
```

**type:**

```
(inspect record-test-record)
```

**choose "as a record"**

**choose "record-test-name1"**

*--should produce a window with three elements the first one of which is a list of a list, a string and nil .*

```
(nil hi ((a b c) some string nil)) Inspector
record-test-name ((a b c) "some string" nil)
zebra "hi"
xray nil
```

**type:**

```
(inspect record-test-record 'record-test-name1)
```

*--should produce a window identical to the one in the previous step.*

**type:**

```
(ed 'record-test-name1 'records)
```

delete zebra from the element list.

type cntl-x to save edit.

**type:**

```
(inspect record-test-record 'record-test-name1)
--should produce a window with no zebra and the value that used to be zebra is now record-test-name.
```

**return to the edit window and type zebra back in where it belongs.**

**type cntl-x to save again.**

**type:**

```
(inspect record-test-record 'record-test-name1)
--should produce a window with three elements the first one of which is a list of a list, a string and nil and
zebra is returned to the list
```

**close the edit window**

**type:**

```
(in-package 'interlisp)
```

**type:**

```
(inspect user::record-test-record 'user::record-test-name1)
--should produce a window with three elements where all the values not in quotes are qualified as user.
```



The screenshot shows a window titled "(nil hi ((a b c) some string nil)) Inspector". Inside the window, the following structure is displayed:

```
user::record-test-name ((user::a user::b user::c)
user::zebra "hi"
user::xray nil)
```

**type:**

```
(cl:in-package 'user)
```

-----



Script to check code inspectors on frames  
(can be in any package)

**type `cntl-b` to cause a break.**

**Place the mouse in the break window and press the middle button**

**select `BT!`**

**choose a frame that does not have a `\` and select it with the middle button.**

**When a menu appears choose `InspectCode`**

*Observe the code window -- it should scroll and allow selections but not allow any changing.*

**return to the window caused by the `BT!`**

**choose a frame that does have a `\` and select it with the middle button.**

**When a menu appears choose `InspectCode`**

*Observe the code window -- it should scroll and allow selections but not allow any changing.*

-----

**return to the window caused by the `BT!`**

**select an entry with the left button.**

**a window will appear that has a title (whatever entry you chose) Frame.**

**In that window select a element in the left hand column with the left button.**

**Press the middle button over the same element.**

**select `"set"`**

**type: `5`**

*-- the value of the chosen element should become 5*

**Find a list in the right hand column of the frame window and select it with the left button.**

**Press the middle button and choose `"inspect"`**

*-- an inspect window with the elements of the list will appear*

Script for testing defstruct and the inspector with defstruct  
(assuming currently in **Common Lisp User** package)

**type:**

```
(cl:in-package "USER")
```

**type:**

```
(defstruct-rec (a "hello there" :type IL:pointer) (b 5 :type il:integer) (c 4 :type il:fixp) (d
2.3 :type il:floating) (e 4.5 :type il:floatp) (f -5 :type il:signedword :read-only t) (g T :type
il:flag) (h 7 :type il:bits3) (i 4 :type il:byte) (j 7 :type il:word) (k '(a b c d) :type
il:xpointer))
```

-- it should reply rec

**type:**

```
(setf rec1 (make-rec :e 2.3))
```

**type:**

```
(inspect rec1)
```

-- a window that looks like the following appears



```
IL:WITH S) S)
(EQ (IL:REPLACE BRAVO IL:OF RECORD-TEST-RECORD
IL:WITH ALFA) ALFA)))
```

(DO-TEST |refetch-arrayrecord|

```
(AND (EQ (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD) S)
(EQ (IL:FETCH BRAVO IL:OF RECORD-TEST-RECORD) ALFA)
(EQ (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD) S)))
```

(DO-TEST |rereplace-arrayrecord|

```
(AND (EQ (IL:REPLACE ALPHA IL:OF RECORD-TEST-RECORD
IL:WITH ALFA) ALFA)
(EQ (IL:REPLACE BRAVO IL:OF RECORD-TEST-RECORD
IL:WITH S) S)))
```

(DO-TEST |typeglobalvariable-GETFN: BMOBJ.GETFN3|  
(EQ (IL:FETCH BRAVO IL:OF RECORD-TEST-RECORD) ALFA)  
(EQ (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD) S)))

(DO-TEST |rereplace-assocrecord|

```
(AND (EQ (IL:REPLACE ALPHA IL:OF RECORD-TEST-RECORD
IL:WITH ALFA) ALFA)
(EQ (IL:REPLACE BRAVO IL:OF RECORD-TEST-RECORD
IL:WITH S) S)))
```

(DO-TEST |typeglobalvariable-assocrecord|  
(EQ (SYMBOL-PACKAGE (IL:TYPEGLOBALVARIABLE  
(QUOTE RECORD-TEST-NAME))))

```
(FIND-PACKAGE "XCL-TEST"))))
```

```
(DO-TEST |using-assocrecord|
 (SETQ RECORD-TEST-RECORD3
 (IL:CREATE RECORD-TEST-NAME
 IL:USING RECORD-TEST-RECORD GAMMA IL:_ S))
 (AND (EQ (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD)
 (IL:FETCH ALPHA IL:OF RECORD-TEST-RECORD3))
 (EQ (IL:FETCH BRAVO IL:OF RECORD-TEST-RECORD)
 (IL:FETCH BRAVO IL:OF RECORD-TEST-RECORD3))
 (EQ (IL:FETCH GAMMA IL:OF RECORD-TEST-RECORD3) S)))
```

```
(DO-TEST |reusing-assocrecord|
 (SETQ RECORD-TEST-RECORD3
 (IL:CREATE RECORD-TEST-NAME
 ILuote (a b c d)) il:xpointer nil (rec 20 il:xpointer) rec-k)
```

Script for INSPECTALLFIELDSFLG  
(assuming in the **Interlisp** package)

**Type:**

```
(CL:IN-PACKAGE "INTERLISP")
```

**Type:**

```
(SETQ INSPECTALLFIELDSFLG T)
(INSPECT (CREATEREGION 1 2 3 4) 'REGION)
```

*-The window should have the fields TOP, PTOP, RIGHT, PRIGHT, HEIGHT, WIDTH, BOTTOM, and LEFT. It should look like the following:*

| (1 2 3 4) Ins |   |
|---------------|---|
| TOP           | 5 |
| PTOP          | 6 |
| RIGHT         | 3 |
| PRIGHT        | 4 |
| HEIGHT        | 4 |
| WIDTH         | 3 |
| BOTTOM        | 2 |
| LEFT          | 1 |

**Type:**

```
(SETQ INSPECTALLFIELDSFLG NIL)
(INSPECT (CREATEREGION 1 2 3 4) 'REGION)
```

*-The window should have the fields HEIGHT, WIDTH, BOTTOM, and LEFT. It should look like the following:*

| (1 2 3 4) Ins |   |
|---------------|---|
| LEFT          | 1 |
| BOTTOM        | 2 |
| WIDTH         | 3 |
| HEIGHT        | 4 |

**Type:**

```
(SETQ INSPECTALLFIELDSFLG T)
```

Script for testing Inspect macro interface  
Start in **interlisp** package

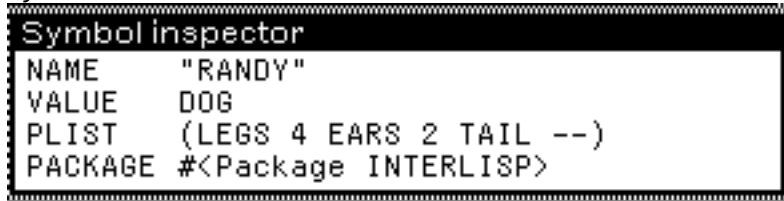
**type:**

```
(CL:IN-PACKAGE "INTERLISP")
```

**type:**

```
(DEFINEQ (DOGSP (ANIMAL) (OR (EQUAL (EVAL ANIMAL) 'DOG) (EQUAL (EVAL ANIMAL) 'dog))))
(PUTPROPS RANDY LEGS 4 EARS 2 TAIL 1 HAIR LONG)
(SETQ RANDY 'DOG)
(INSPECT 'RANDY)
```

--you will see:



Symbol inspector

|         |                         |
|---------|-------------------------|
| NAME    | "RANDY"                 |
| VALUE   | DOG                     |
| PLIST   | (LEGS 4 EARS 2 TAIL --) |
| PACKAGE | #<Package INTERLISP>    |

**Type:**

```
(SETQ INSPECTMACROS (CONS '((FUNCTION DOGSP) PROPNames GETPROP PUTPROP) INSPECTMACROS))
(INSPECT 'RANDY)
```

--you will see:



RANDY Inspector

|      |      |
|------|------|
| LEGS | 4    |
| EARS | 2    |
| TAIL | 1    |
| HAIR | LONG |

**Type:**

```
(SETQ INSPECTMACROS (CDR INSPECTMACROS))
```

Testing the programmatic interface to INSPECTW  
(assuming in **Interlisp** package)

**type:**

```
(CL:IN-PACKAGE "INTERLISP")
```

**type:**

```
(DEFINEQ (FETCHFROMLIST (AL PRP)
 (COND ((EQUAL PRP (QUOTE FIRST))
 (CAR (EVAL AL)))
 ((EQUAL PRP (QUOTE SECOND))
 (CADR (EVAL AL)))
 ((EQUAL PRP (QUOTE THIRD))
 (CADDR (EVAL AL)))
 ((EQUAL PRP (QUOTE FOURTH))
 (CADDR (EVAL AL))))))

(DEFINEQ (STOREINLIST (AL PRP NV)
 (SET AL
 (COND ((EQUAL PRP (QUOTE FIRST))
 (CONS NV (CDR (EVAL AL))))
 ((EQUAL PRP (QUOTE SECOND))
 (CONS (CAR (EVAL AL))
 (CONS NV (CDDR (EVAL AL))))))
 ((EQUAL PRP (QUOTE THIRD))
 (CONS (CAR (EVAL AL))
 (CONS (CADR (EVAL AL))
 (CONS NV
 (CDDR (EVAL AL))))))
 ((EQUAL PRP (QUOTE FOURTH))
 (CONS (CAR (EVAL AL))
 (CONS (CADR (EVAL AL))
 (CONS (CADDR (EVAL AL))
 (CONS NV NIL)))))))))

(DEFINEQ (PROPCOM (PRP AL INS)
 (SET AL (REVERSE (EVAL AL)))))

(DEFINEQ (VALCOM (VAL PRP AL INS)
 (SET AL (REVERSE (EVAL AL)))
 (INSPECTW.REDISPLAY INS NIL)))

(DEFINEQ (TITLECOM (INS AL)
 (SET AL (QUOTE (X Y Z A)))
 (INSPECTW.REDISPLAY INS NIL)))

(DEFINEQ (SELECTF (PRP VF INS)
 (PRINTOUT T T "YOU PICKED " PRP)
 (if VF then
 (PRINTOUT T " OR RATHER, THE VALUE OF " PRP T)
 else (PRINTOUT T " IT IS A PROPERTY" T))))

(DEFINEQ (PROPP (PRP AL)
 (LIST AL PRP)))

(SETQ ALIST ' (A B C D))

(SETQ WIND (INSPECTW.CREATE 'ALIST ' (FIRST SECOND THIRD FOURTH) 'FETCHFROMLIST 'STOREINLIST))
```

-- an inspect window appears



select a name (first,second, third or fourth) with the left button, then press the middle button - select set.

type in a new value for the selected element, confirm that the value changes.

Type ALIST on the exec window and notice that the list has changed.

**type:**

```
(INSPECTW.CREATE 'ALIST ' (FIRST SECOND THIRD FOURTH) 'FETCHFROMLIST 'STOREINLIST 'PROPCOM)
```

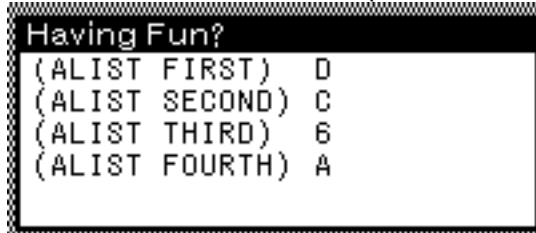
select a name (first, second, third or fourth) with the left button, then press the middle button. It should appear as if nothing has happened. Position the mouse in the title bar of the inspect window, press down the middle button - select refetch. *The values of Alist should be reversed.* Type ALIST on the exec window and notice that the list has changed.

**Close all inspector windows**

**type:**

```
(INSPECTW.CREATE 'ALIST '(FIRST SECOND THIRD FOURTH) 'FETCHFROMLIST 'STOREINLIST "HELLO THERE GUESS WHO?" 'VALCOM 'TITLECOM "Having Fun?" 'SELECTF WIND 'PROPP)
```

- the inspector window should appear where you placed the first one. Shape the window so you can view all of it. It should look like this (notice the new title):



**select something from the left-hand column**

-a window will appear telling you what you selected and that it is a property.

**press the middle button on the same selection**

- a message "HELLO THERE GUESS WHO" will appear in the prompt window

**select something from the right hand column**

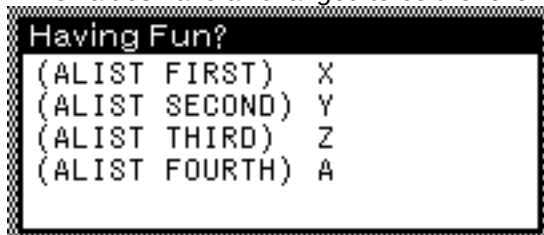
-a message will appear in the same window that appeared two steps back telling you what property the value you selected belongs to and that it is a value.

**press the middle button on the same selection**

-the values reverse themselves again

**Put the mouse in the title bar and press the middle button**

-The values have all changed to be the following.



## Testing Report

### Tested:

- different ways of creation: using, copying, reusing, smashing
- type?, typenamep, typename are tested for each type of record that maintains typename info: typerecord, arrayrecord, datatype
- for record types with and without typename info, provided a test form, and made sure it works via type?
- field name synonyms
- typenamep.- on an 1109
- records which turn into datatypes with type#>255 The sysout has more than 255 datatypes already, so this is well tested.
- defstruct's with types on the slots.
- checked that type number (as built by \typeglobalvariable) are in the current package.
- Similarly, tested the interaction of packages with Interlisp records. (i.e. that the record name and the field names are in the same package.) Made sure that package qualified versions of things work with fetch and replace.
- recursive records with above.
- new record types: confirmed that the irm is correct.
- editing of records with SEdit (multiple completion).
- inspected any type of object the system can create
- specified type in ASTYPE arg to INSPECT
- user options INSPECTALLFIELDSFLG, MAXINSPECTCDRLEVEL, MAXINSPECTARRAYLEVEL, INSPECTPRINTLEVEL
- extensibility via INSPECTMACROS
- confirmed programmatic interface to INSPECTW's
- code inspectors on frames (both \realframep frames and others.),
- inspected values in frame variables.
- the setters on defstruct slots (including readonly-ness) for lists, vectors, :named and otherwise.
- examples of propprintfn

### Left to be tested:

- typenamep should be tested on each machine. and was only tested on the 1109
- code inspectors on frames (both \realframep frames and others.), should be tested from both compilers
- all tests should be run compiled and interpreted on each machine.

### Outstanding bugs and comments

7630 Smashing-arrayrecord fails

6950 WITH doesn't compile correctly. (new)

6606 re: record redeclaration.

4007 want to have - in record field names (closed)

1046 ensure that the editors can be successfully invoked from the inspector.

1820 make sure you can edit terminal tables

2019 problem with \applyinspectmacro (open)

2613 try inspecting processes (running and dead), other system types.

Note: there is no interaction between the inspector and \*print-circle\* and \*print-pretty\*



### 3. How to Create Sketch Elements

---

The elements a sketch consists of are text, boxed text, lines, boxes, polygons, curves, closed curves, circles, ellipses, arcs, arrowheads, and bit maps. This chapter explains how to create and change each kind of element, and how to change the way new elements will look.

---

#### To Use Text in a Sketch

---

In Sketch, text is provided by text elements. Each text element has some characters, a control point that positions it, and properties that determine the way it looks (e.g., boldness, font family) and how it is justified relative to its position (e.g., left, right, or center justification). A new text element is added by typing it in (see the section “To Type In Text,” below). You edit existing text elements by selecting within them and typing (see the sections “To Insert Characters Into a Piece of Existing Text” and “To Replace Characters in a Piece of ExistingText”). The size, font family, boldness, and italic properties, and the location of the text relative to its position, are changed using the Change command (see the section “To Change the Way Text Looks”).

In a sketch window, the three mouse buttons provide quick access to text editing and line drawing. Figure 8 summarizes these mouse button functions.

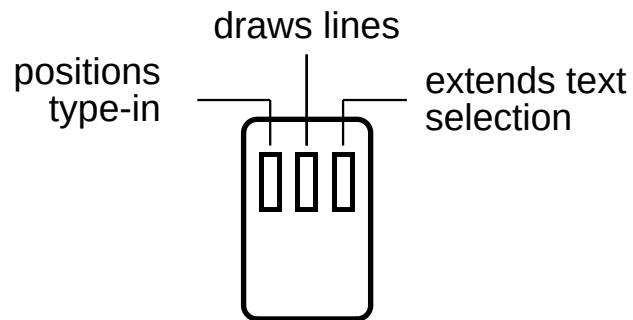


*Figure 8. Mouse button text-editing and line-drawing functions*

---

#### To Type In Text

---

Move the cursor to where you would like the text to be and press the left mouse button. The caret shape (I) will appear. Type the text. Typing a carriage return will start another line.

If a vertical bar  appears instead of a caret, the position you selected is in existing text (see the following sections). You cannot create a new text element in the middle of an existing text element. Any characters you type will be added to the existing text. To create a new text element, hold down the left button and move the cursor until the vertical bar  changes into the caret shape . At this point you can create a new text element by typing new text.



1. Put the cursor where the new text should be.



2. Type the text.  
In this case, "a box."

*Figure 9. Steps to insert new text*

When characters are typed, a new piece of text is centered around the position of the caret. The alignment of the text relative to this position can be changed. For example, the text can be changed so that this position is its left edge. See the sections "To Change the Justification of Text" and "To Change the Properties of New Text."

---

## To Insert Characters Into a Piece of Existing Text

---

Move the cursor to the place in the text where you want to insert the characters and click the left button. When you press the left button, the vertical bar  will appear between the characters. Type the characters. The typed characters will appear where the vertical bar is. If you hold the left button down while moving the cursor, the vertical bar will follow.

If you place the cursor over a piece of text and the vertical bar does not appear, the text may be contained in a group or it may be overflow from a text box. If it is in a group, you can edit this text only by ungrouping it first. See the section "To Use Groups." If it is overflow from a text box, you can reshape the text box to make it large enough to hold the text. See the section "To Change the Size of a Text Box."

You can insert characters at the beginning of a text element by selecting the left half of the first character. The vertical bar will appear in front of the first character when the cursor is positioned correctly. If you move out of the text, the vertical bar will change to a caret shape . If you type when the caret shape is visible, you will create a new text element rather than inserting characters into the existing one. You can insert characters at the end of a text element by selecting the right half of the last character.

## To Replace Characters in a Piece of Existing Text

---

Move the cursor to in front of the first character to be replaced; press and release the left button. A vertical bar (|) will appear in front of that character. Move the cursor to the last character to be replaced; press and release the right button. After you press the right button, the text that will be replaced when you type is shown white-on-black. Type the new characters (see figure 10).

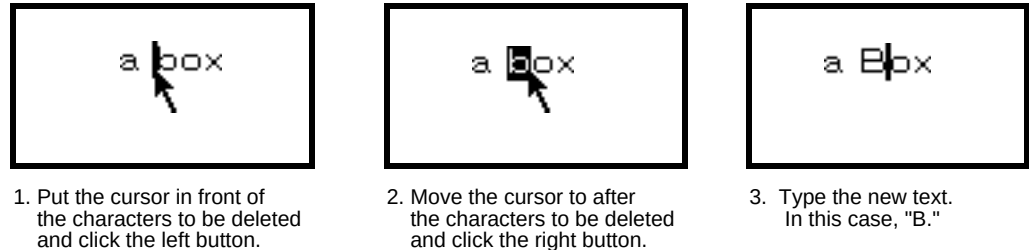


*Figure 10. Steps to edit existing text*

## To Delete Characters in a Piece of Existing Text

---

Select the characters using the left and right buttons as described above in the section "To Replace Characters in a Piece of Existing Text" and press the delete key.

## To Delete Text During Type-in

---

When you are typing text, you can use the backspace (BS) key to delete the previous character, and control-W to delete the previous word.

## To Change the Way Text Looks

---

You can change the properties of text by using the Change command. After selecting the Change command, select a piece of text or a collection of pieces of text (see the section "To Select Sketch Elements") that you wish the change to apply to. You will then be presented with a menu (see figure 11) of possible ways of changing the text.

## To Change the Bold and Italic Properties of Text

---

Select the Change command, move the cursor to the control point of the text (which will be marked with a ) and press and release the left button. The menu shown in figure 11 will appear. Select one of the items Bold, Unbold, Italic, or Unitalic.

Bold will make the selected text element appear in bold letters. Unbold will remove the bold property from the text if it was previously bold. Italic will make the selected text element appear in italics. Unitalic will change italic text to roman text.



Figure 11. Menu offered for changing text elements

---

## To Change the Size of Text

Select the Change command, move the cursor to the control point of the text (which will be marked with a ) and press and release the left button. The menu shown in figure 11 will appear. Select one of the items Smaller Font, Larger Font, or Set Font Size.

Smaller Font will make the characters appear in the next smaller font. Larger Font will make the characters appear in the next larger font. Set Font Size will prompt you for a size to make the text.

Note: The font size is changed at the scale in which the text was originally entered. If you are viewing the text from a *zoomed view* (a window that has had its scale changed) the text size may not change. Or it may change more than a single size. If you have difficulty getting the size you want, enter a new piece of text, adjust its size, make the original piece of text have the same size (see the section "To Make Several Pieces of Text Look Alike"), and then delete the new text.

---

## To Change the Font Family of Text

Select the Change command, move the cursor to the control point of the text (which will be marked with a ) and press and release the left button. The menu shown in figure 11 will appear. Select the item Different Font. A menu of the known font families and the item Other will appear. Select the name of the family you want or, if it is not there, select Other. If Other is selected, a small window with

the message "New family:" will appear above the sketch window. The caret will be blinking in it. Type the name of a font family, ending with a carriage return. If the selected or entered font family is not available in the size of the selected text element, a message is printed and nothing is changed.

When selecting the control point of the text, you can also select more than one text or text box element (see the section "To Select Sketch Elements"). If more than one text element is selected, any that have the same size as the first selected text element are changed to the new font family.

Note: the search for fonts encompasses any directories on DISPLAYFONTDIRECTORIES and may take a few minutes. (If font file servers are down or slow, it may take even longer.) If Sketch doesn't find a font that you believe exists, you can make that font the default font, both family and size (see the section "To Change the Properties of New Text") and retry the change.

## To Change the Justification of Text

Select the Change command, move the cursor to the control point of the text (which will be marked with a ) and click the left button. The menu shown in figure 11 will appear. Select one of the items Left Justify, Center Justify, Right Justify, Top Justify, Bottom Justify, Middle Justify, or Baseline Justify. These commands will change where the text appears relative to its control point. Table 1 shows the effects of the different commands on horizontal and vertical justification.

| Vertical<br>Horizontal | Baseline Justify  | Middle Justify    | Top Justify       | Bottom Justify    |
|------------------------|-------------------|-------------------|-------------------|-------------------|
|                        | Center Justify    | Left Justify      | Right Justify     |                   |
| Center Justify         | Play              | Play              | Play <sup>+</sup> | Play <sub>+</sub> |
| Left Justify           | <sub>+</sub> Play | <sup>+</sup> Play | <sup>+</sup> Play | <sub>+</sub> Play |
| Right Justify          | Play <sub>+</sub> | Play <sup>+</sup> | Play <sup>+</sup> | Play <sub>+</sub> |

<sup>+</sup> - control point of the text

*Table 1. How the justification properties affect the position of text relative to its control point*

## To Make Several Pieces of Text Look Alike

Select the Change command, hold down one of the shift keys, move the cursor to the control point of the text that looks the way you want, and click the left button. Using the procedure described

in the section “To Use Menus and Submenus,” select the text elements you want to change to look like this one. When finished, release all the mouse buttons and the shift key. The menu shown in figure 11 will appear. Select the item Look Same. This will make all the selected text items be the same font size, face, and alignment as the one you selected first. The change is made to both boxed and unboxed text. This command is convenient for making text that was entered at different scales look the same.

---

## To Use Boxed Text

---

Sketch has the capability of framing text and justifying text within the frame. The element that supports this is called a text box. Whenever the caret is inside a text box, the characters typed become part of the text within the box. The text in a box is broken into lines between words so that each line fits within the width of the box. If a single word in the text is wider than the box, it spills over. If there are more lines than fit in the height of the box, they spill over also. The characters outside the box cannot be selected for editing; the box must be enlarged to allow these to be edited.

A text box can be created by boxing a text element (see “To Put a Box Around Existing Text”), by typing a control-carriage return (see “To Create a Box to Put Text In”), or by using the **TEXT BOX** command (see “To Create a Sized Box to Put Text In”). The justification properties of the text specify the position of the text within the box (see “To Reposition the Text Within a Box”). The text inside a text box has the same looks properties as a text element (see the sections “To Use Text in a Sketch” and “To Change the Way Text in a Box Looks”).

The frame around the text is called its box. It has control points at its upper-right and lower-left corners. Moving one of these changes the size of the box (see “To Change the Size of a Text Box”). The box has thickness and dashing pattern properties that affect the frame (see “To Change the Border Thickness of a Text Box” and “To Make a Dashed Border Around a Text Box”) and a filling property that shades the part of the box not occupied by text (see “To Change the Filling of a Text Box”). Figure 12 shows some examples of text boxes.

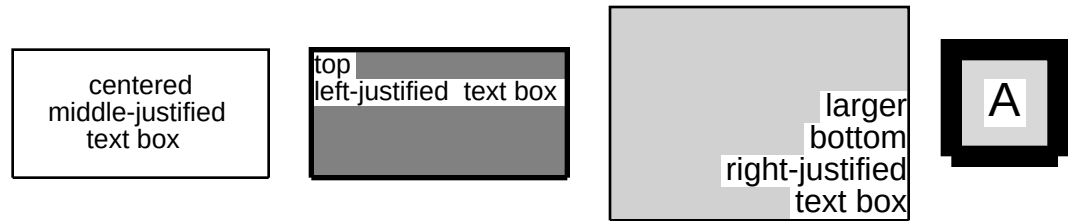


Figure 12. Examples of text boxes

---

## To Put a Box Around Existing Text

Select the Change command from the command menu, move the cursor to the control point of the text you wish to box, and press and release the left button. A large menu titled Change Text How? will appear (see figure 11). Select the Box the Text item. The text will become part of a text box.

---

## To Create a Box to Put Text In

Hold down the control key and press the return key. If the caret is inside a text box, a new text box of the same size will appear below it. Any characters typed will now go into the new box. If the caret is not in a text box, a new box will appear at the current cursor position. Note: you should hold down the control key until the new box appears.

---

## To Create a Sized Box to Put Text In

Select the **Text Box** command from the command menu, move the cursor to one corner of the desired box location, press and hold the left button, move the cursor to the diagonally opposite corner, and release the button. While you are moving to the opposite corner, a gray outline of the box will be shown. When you release the button, the gray outline will be replaced by a solid one. You can stop this command by releasing the button when the cursor is outside the window.

---

## To Change the Size of a Text Box

Move either control point of the text box using one of the point-moving methods described in the section "To Move Elements."

---

## To Change the Border Thickness of a Text Box

Select the Change command, move the cursor to one of the corner points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the item Box Thickness. This will bring up a menu titled Change Size How? that contains the items Smaller Line, Larger Line, and Set Line Size. Selecting Smaller Line will

make the box outline be one size smaller than it is. Selecting Larger Line will make it one size larger than it is. Selecting Set Line Size will bring up a number pad menu (see figure 13). In this case, you should enter the size in screen points (1/72 of an inch) that you want the box thickness to be. The box thickness can be set to zero with the Set Line Size command, but the Smaller Line command will not make it less than one. If the thickness is zero, the box around the text won't appear, but any filling will remain and the text will still justify itself within the box's boundary.

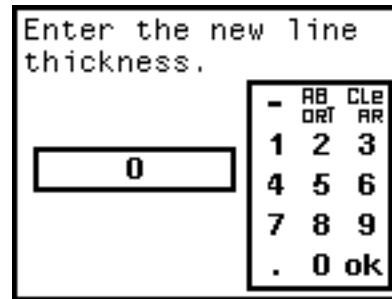


Figure 13. The number pad menu for entering line thickness

---

## To Make a Dashed Border Around a Text Box

You can make the border of a text box dashed by specifying a *dashing pattern* (see the section "To Make a Dashed Line"). A dashing pattern is a sequence of numbers that indicates how many brush marks should be on and off. To specify a dashing pattern, first select the Change command, then move the cursor to one of the corner points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the Dashing item. This will bring up a menu titled New Dashing Pattern?, similar to the one shown in figure 34, that contains several dashing patterns and the items Other and No Dashing. Selecting one of the dashing patterns will make the border have that pattern. Selecting the Other item will bring up a series of number pad menus in which you enter, alternatively, the size of the black portion of the pattern and the size of the white portion of the pattern. The pattern can have as many alternations as you like. Number pad menus will continue to appear until you enter zero. When you enter zero, a menu is brought up showing what your new dashing pattern looks like and requesting confirmation. If you like the pattern, select Yes. If you don't like it, select No. If you select No, you will be given a chance to enter another series of sizes. If you select Yes, the box outline will be dashed according to your pattern. The pattern you entered will be remembered and will appear in the dashing menu next time you change the dashing.





Figure 14. Change menu for text boxes

## To Change the Filling of a Text Box

Select the Change command, move the cursor to one of the control points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the Filling item. From this point, follow the procedure given in the section "To Change the Filling of a Box."

## To Change the Way Text in a Box Looks

Select the Change command, move the cursor to one of the corner points of the text box (which will be marked with a ) and press and release the left button. The menu shown in figure 14 will appear. Select the item The Text. The menu shown in figure 15 will appear. Select one of the items Different Font, Smaller Font, Larger Font, Set Font Size, Bold, Unbold, Italic, or Unitalic. The action of these items is described in the section "To Change the Way Text Looks."

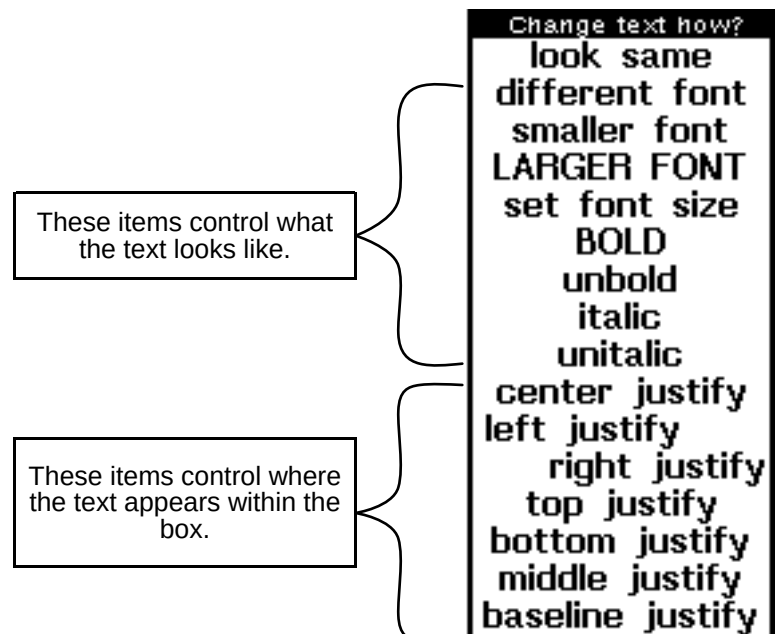


Figure 15. Menu offered for changing boxed text elements

## To Remove a Box From Around Text

---

Select the Change command, move the cursor to one of the corner points of the text box (which will be marked with a  and press and release the left button. The menu shown in figure 14 will appear. Select the item Unbox the Text. The box and all of its properties (such as filling) will be removed. If you want to keep the filling but not display the box, set the border thickness to zero (see the section “To Change the Border Thickness of a Text Box”).

## To Reposition the Text Within a Box

---

Select the Change command, move the cursor to one of the corner points of the text box (which will be marked with a  and press and release the left button. The menu shown in figure 14 will appear. Select the item The Text. The menu shown in figure 15 will appear. Select one of the items Left Justify, Center Justify, Right Justify, Top Justify, Bottom Justify, Middle Justify, or Baseline Justify. Left Justify will move each line of the text to the left edge of the box. Right Justify will move lines to the right edge. Center Justify will cause each line to be centered between the left and right edges. Top Justify will move the text so that the first line is at the top edge of the box. Bottom Justify will move it so that the last line of text is at the bottom edge. Middle Justify will move it so that the middle of the middle line of text is halfway between the top and bottom edges. Baseline Justify will move it so that the baseline position of the middle line of text is halfway between the top and bottom edges.

## To Make the Text in Several Text Boxes Look Alike

---

The procedure is the same as the one described in the section “To Make Several Pieces of Text Look Alike.”

---

## To Add Lines to a Sketch

---

Most sketch elements other than text, such as boxes, circles, and curves, are made up of lines. This section describes all the methods for adding lines, but does not tell you how to change their properties. The line properties are thickness, brush shape, and dashing, and you can change them using the information in the section “To Change the Way Lines in Elements Look.”

## To Add a Line

---

To add a single line to a sketch, move the cursor into the sketch window and press the middle mouse button. While the middle button is held down, the cursor changes to . A mark will

follow the cursor on grid points, described in the section “To Use the Grid Display”). Move the cursor to where you want one end of the new line and release the middle button. Press and hold the middle button again and move the cursor to where you want the other end. A line will be stretched from the first position to the cursor position and will follow it. When the line is in the right place, release the middle button. If you move outside the window, the stretched line will disappear. If you release the button while the cursor is outside the window, no line is added. This provides a way of aborting if you change your mind while placing the second end of a line. If you move back into the window with the button still down, the line will reappear. If you want to change the position of the first point, click the left button while inside the sketch window, then start this procedure over.

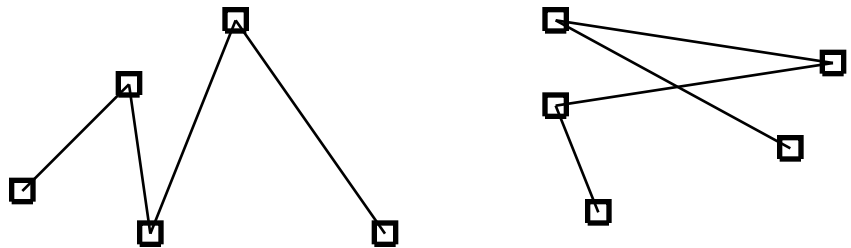


*Figure 16. Two examples of connected lines with control points highlighted*

---

## To Add a Series of Connected Lines

You can add a series of connected lines using either the mouse or the  command. Using the mouse enables you to see and reposition the lines as you draw them, while using the  command may sometimes be faster.

---

### With the Mouse

Move the cursor to over the Defaults menu command in the Sketch command menu, press and hold the left button, and slide the cursor out the right side through the triangle. A menu that includes the item Line will appear. Move the cursor over the Line item and slide out its right side. A menu that includes the item Mouse Line Specs will appear. Move the cursor over the Mouse Line Specs item and release the left button. A fourth (and final) menu with the title Connect Middle Button Lines? will appear. Select the Yes item from this menu. This procedure (which you only have to do once) has changed the mode of adding lines with the middle button to construct a series of lines. If you now draw lines as before, a new line will be stretched to the cursor position from the end of the previous line. To start a new series of lines, press either the left or the right mouse button in the sketch window. (A new series of lines is also started when you select a menu command.) You can stop the drawing of an individual line segment by moving the mouse cursor outside the window.

This does not start a new series of lines; moving back into the window and pressing the middle button picks up where the last line ended.

If you want to change the mode from entering connected lines back to unconnected lines, follow the above procedure but select the No item from the Connect Middle Button Lines? menu.

---

### With the Command

---

Select the  from the Sketch command menu. The cursor will change to  and an  will follow the cursor. Move the cursor to each point that the lines go through and click the left mouse button. Each time you specify an endpoint, it is marked with a . After all the points have been specified, move the cursor outside the window and press the left mouse button. After you click outside the window, the line will be drawn connecting the . Figure 16 shows two examples of connected lines.

---

### To Add Boxes to a Sketch

---

Sketch boxes can be either filled or unfilled; in addition, their borders may vary in thickness and brush shape, or be dashed. To add a box to a sketch, select the  command from the Sketch command menu. The cursor will change to . Move the cursor to one corner of the rectangle where you want the box. Press the left button. Holding the left button down, move the cursor to the opposite corner and release the button. While the left button is down, the rectangle will be highlighted in gray. When you release the button, a box will be added. To abort, move the cursor outside the window and click the left button. Note: the specified box must be entirely within the window. Figure 17 shows some examples of boxes.



*Figure 17. Three examples of boxes.  
The left one has its control points highlighted*

## To Change the Filling of a Box

Select the Change command from the command menu, then select the box or boxes whose filling you wish to change using the procedure described in the section "To Select Sketch Elements." You can include elements other than boxes as long as the first element you select is a box. The menu shown in figure 18 will appear.



Figure 18. Menu of properties you can change for boxes

Select the Filling item. A menu titled New Filling? similar to the one shown in figure 19 will appear.

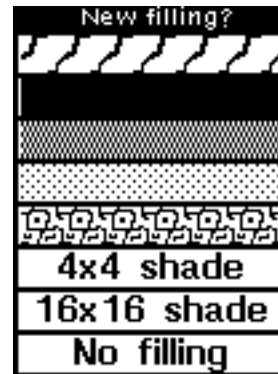


Figure 19. Menu of textures that can fill boxes

If the filling you want is shown on the menu, select it. If you want to remove the filling from a box, select the No Filling item. To create a shade that is not on the menu, select either the 4x4 Shade or the 16x16 Shade item. Selecting either item will cause the shade editor to appear (see figure 20). The 4x4 Shade item will allow you to create a 4-bit-by-4-bit shade. The 16x16 Shade item will allow you to create a 16-bit-by-16-bit shade. Construct the shade you want in the lower part of the window by turning points black with the left button and white with the middle button. Selecting Quit will cause the box to be filled with the newly constructed shade. The shade you created will be added to the menu for future selection. Advanced user note: the function (SK.CACHE.FILLING SHADE) can be called to add a filling to the menu.

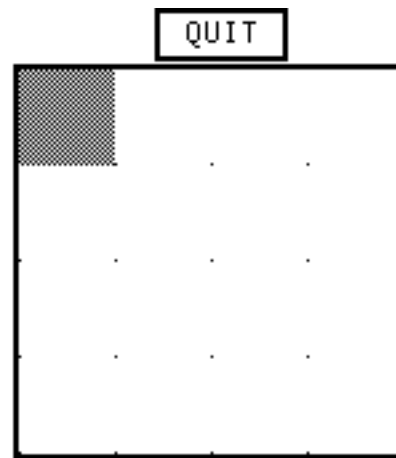


Figure 20. The shade editor

To change the thickness, dashing, or brush shape of a box's border, see the section "To Change the Way Lines in Elements Look."

---

## To Add Polygons to a Sketch

---

Select the  command from the Sketch command menu. The cursor will change to  and an  will follow the cursor. Move the cursor to each vertex of the polygon and click the left button. When you select a point, it is marked with a . When you have selected all the points, move the cursor outside the window and click the left button. The points you selected will become the control points of the polygon. To abort, move the cursor outside the window and click the left button before selecting any points. Figure 21 shows some examples of polygons.

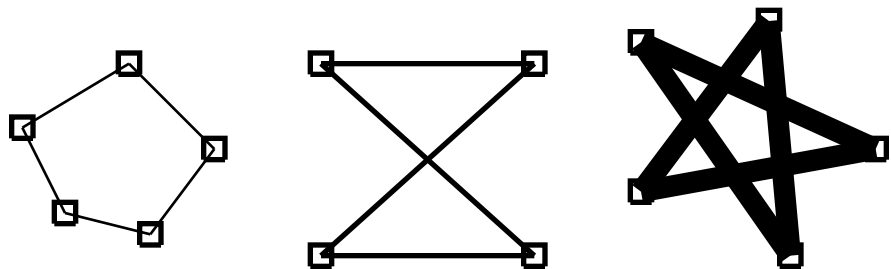


Figure 21. Three examples of polygons with control points highlighted

You can move any of the control points (using one of the point-moving commands described in the section "To Move Elements," below) to change the polygon.

## To Add Curves to a Sketch

There are two kinds of curves you can add to a sketch: open curves and closed curves. This section tells you how to add each kind of curve.

### To Add an Open Curve

Select the  command from the Sketch command menu. The cursor will change to  and an  will follow the cursor. Move the cursor to the points you want the curve to go through and click the left button. When you select a point, it is marked with a . When you have selected all the points, move the cursor outside the window and click the left button. The points where you clicked will become the control points of the curve. To abort, move the cursor outside the window and click the left button before selecting any points. Figure 22 shows some examples of curves.

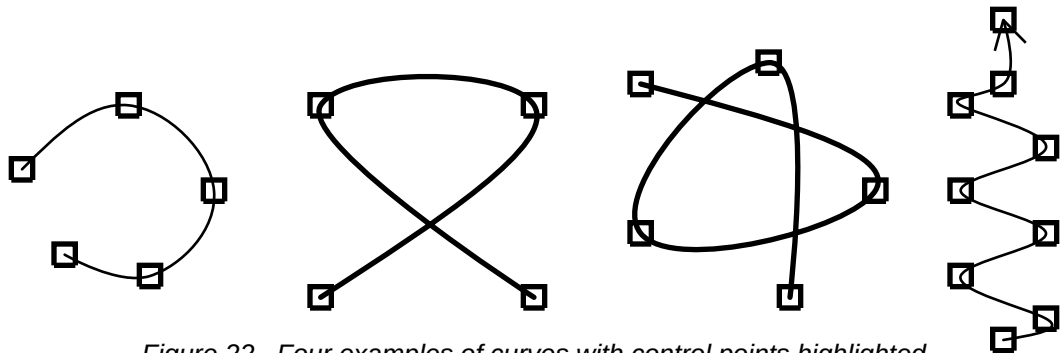


Figure 22. Four examples of curves with control points highlighted

You can move any of the control points (using one of the point-moving commands described in the section “To Move Elements”) to change the shape of the curve. In general, when you construct a curve, the closer together the points are the sharper the curve is; the farther apart they are, the smoother it is. The best way to learn how the control points affect the shape is to enter lots of different curves and move their points around.

### To Add a Closed Curve

Select the  command from the Sketch command menu. The cursor will change to  and an  will follow the cursor. Move the cursor to the points you want the curve to go through and click the left button. When you select a point, it is marked with a . When you have selected all the points, move the cursor outside the window and click the left button. The points where you clicked will become the control points of the closed curve. To abort, move the cursor outside the window and click the left button before selecting any points. Figure 23 gives some examples of closed curves.

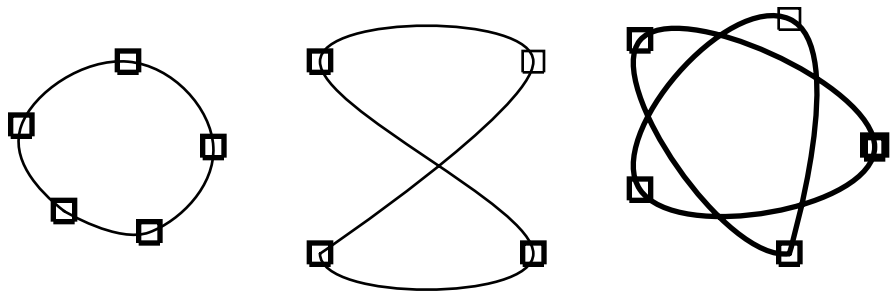


Figure 23. Three examples of closed curves with control points highlighted

You can move any of the control points (using one of the point-moving commands) to change the shape of the closed curve.

---

## To Add Circles to a Sketch

---

Select the  command from the Sketch command menu. The cursor will change to . Move the cursor to the point you want to be the center of the circle and click the left button. The selected point is marked with  and the cursor changes to . Move the cursor to a point you want to be on the radius of the circle and click the left button. The circle will be added. The two points you selected are the control points of the circle. Either can be moved using one of the point-moving commands to change the radius and location of the circle. To abort this command, move the cursor outside the window and click the left button. Figure 24 shows two examples of circles.

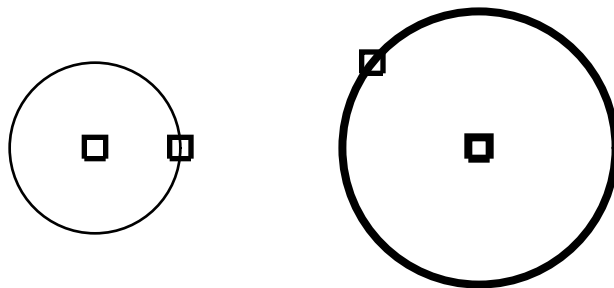


Figure 24. Two examples of circles with control points highlighted



## To Add Ellipses to a Sketch

Select the  command from the Sketch command menu. The cursor will change to . Move the cursor to the point you want to be the center of the ellipse and click the left button. The selected point is marked with  and the cursor changes to . Move the cursor to the point you want to determine one radius and the orientation of the major axis of the ellipse (see figure 25). Click the left button. The selected point is marked with  and the cursor changes to . Move the cursor to any point that is the same distance from the center as you want the second radius to be and click the left button. The ellipse will be added.

The control points of the ellipse are the center and first radius point you selected and the point on the ellipse at the minor radius. Any of them can be moved using one of the point-moving commands to change the size, orientation, and location of the ellipse. To abort this command, click the left button with the cursor outside the window.

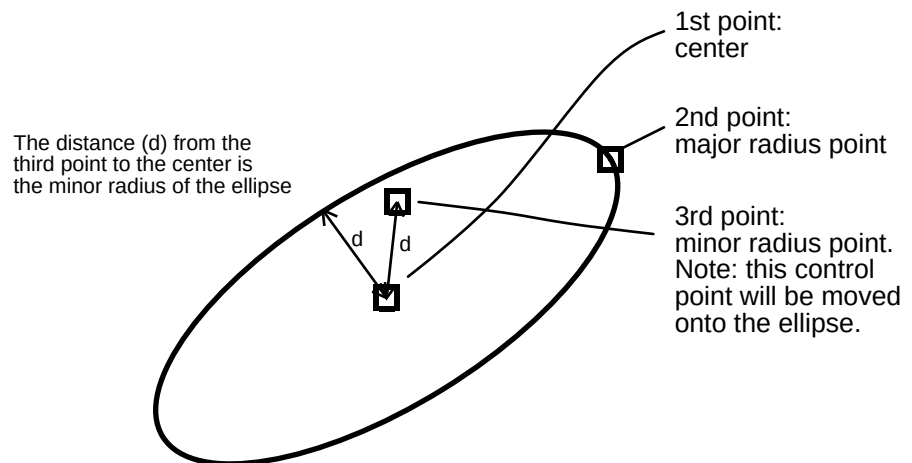


Figure 25. The control points for an ellipse

## o Use Arcs in a Sketch

In Sketch, partial circles are provided by the arc command. An arc is characterized by three control points (see figure 26) and a direction. The shape and angle of the arc can be changed by moving any of the control points (using one of the point-moving commands). The angle of the arc can also be changed by setting the number of degrees the arc spans. (See the section "To Set the Number of Degrees an Arc Spans," below.) The direction determines whether the arc is traversed in a clockwise or a counterclockwise direction from the starting point. An arc has

thickness (size), brush shape, and dashing properties; it can also have arrowheads.

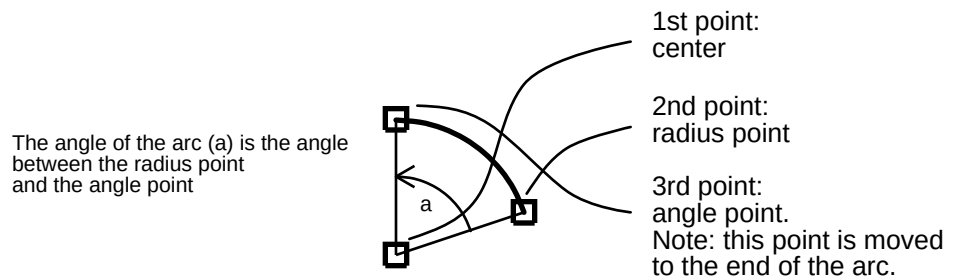


Figure 26. The control points of an arc

## To Add an Arc

Select the  command from the Sketch command menu. The cursor will change to . Move the cursor to the center of the arc and click the left button. The selected point is marked with  and the cursor changes to . Move the cursor to one end of the arc and click the left button. This point determines the radius of the arc and its first end (see figure 26). The arc will begin at this point.

The selected point is marked with  and the cursor changes to . Move the cursor to the other end of the arc and click the left button. The arc will be added. To abort this command, click the left button with the cursor outside the window.

## To Set the Number of Degrees an Arc Spans

Select the Change command, move the cursor to one of the control points of the arc (which will be marked with a ) and click the left button. The menu shown in figure 27 will appear.



Figure 27. Menu of ways to change an arc

Select the Angle item. A number pad menu titled Enter Arc Angle in Degrees will appear (see figure 28). Enter the number of degrees that the arc should span by selecting digits from the number pad. When you are done, select the OK item. The selected arc (or arcs) will be changed to span the indicated number of degrees. The third control point (the angle point) is moved to accomplish this. To abort the command, select the Abort item from the number pad.

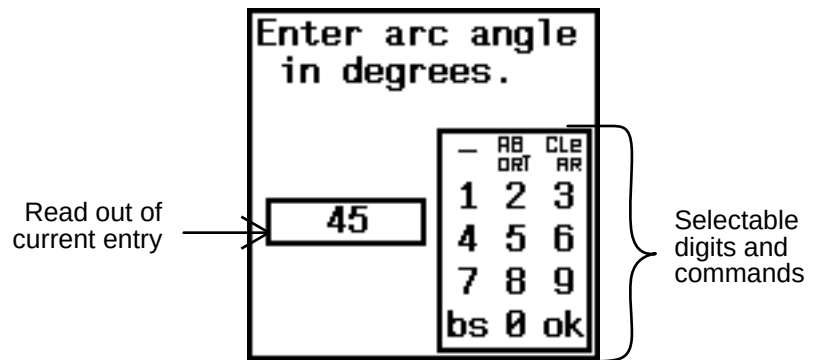


Figure 28. Number pad menu for changing the angle of an arc

## To Reverse the Direction of an Arc

Select the Change command, move the cursor to one of the control points of the arc (which will be marked with a ) and click the left button. The menu shown in figure 27 will appear. Select the Direction item. A menu titled Which Way Should the Arc Go? with the items Clockwise and Counterclockwise will appear. Select Clockwise if you want the arc to go in a clockwise direction. Select Counterclockwise if you want the arc to go in a counterclockwise direction. Figure 29 shows the same arc with different directions.

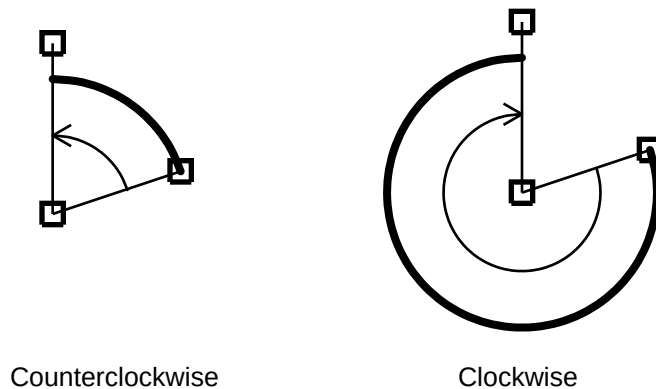


Figure 29. The direction of an arc

## To Use Arrowheads in a Sketch

Arrowheads can be added to lines, open curves, and arcs. An arrowhead can be added to either or both ends of an element. It is then a property of the element and moves with it if it moves.

An arrowhead has a shape, a size, and an angle (see figure 30). The default arrowhead shape, size, and angle specify what a newly added arrowhead looks like. If you want to add several arrowheads that look alike, it is easiest to change the default arrowhead properties (see the section "To Change the Properties of New Arrowheads") before you add the arrowheads. The shape, size,

and angle of arrowheads on existing elements can be changed using the Change command.

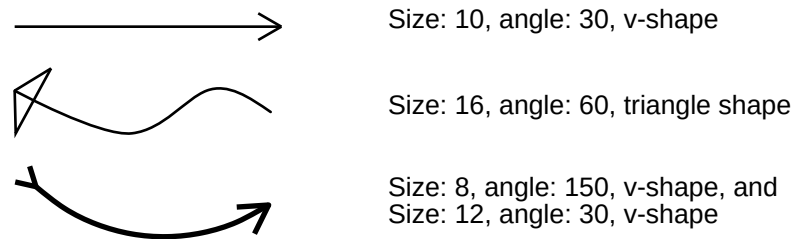


Figure 30. Sample arrowheads

## To Add an Arrowhead to an Existing Element

Select the Change command from the command menu, then select the element or elements you wish to add an arrowhead (or arrowheads) to using the procedure described in the section "To Select Sketch Elements." You can include elements that do not allow arrowheads (such as text or circles) as long as the element first selected does. A menu will appear that contains all of the properties that can be changed in the first element you selected. It will include the item Arrowheads. Select it. The menu shown on the left in figure 31 will appear.

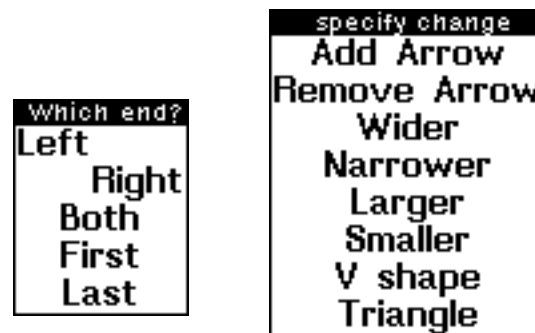


Figure 31. The menus for changing arrowheads.  
The left one specifies which end of the line or curve will be changed.  
The right one specifies how it changes

This menu allows you to specify the end or ends of the selected element(s) to which the arrowhead(s) will be added. Selecting Left will add the arrowhead to the leftmost end of the element (or the topmost if both ends have the same x position). Selecting Right will add the arrowhead to the rightmost end of the element (or the bottommost if both ends have the same y position). Selecting Both will add the arrowhead to both ends of the element. Selecting First will add the arrowhead to the end of the element that was entered first when this element was added. Selecting Last will add the arrowhead to the end of the element that was entered last.

After you select one of these items, the menu shown on the right in figure 31 will appear. Select the item Add Arrow. Arrowheads will be added to the specified ends of the selected elements that do not

already have arrowheads. To abort this operation, click the left button outside any menu instead of selecting an item.

## **To Remove an Arrowhead**

---

Select the Change command from the command menu, then select the element or elements you wish to remove an arrowhead (or arrowheads) from using the procedure described in the section “To Select Sketch Elements.” You can include elements that do not have arrowheads as long as the element you select first does allow them. A menu will appear that contains all the properties that can be changed in the first element you selected. It will include the item Arrowheads. Select it. The Which End? menu shown in figure 31 will appear. Select the item that describes the end or ends (Left, Right, Both, First, or Last) from which the arrowhead(s) should be removed. (These items are described in the section “To Add an Arrowhead to an Existing Element.”) After you select one of these items, the Specify Change menu shown in figure 31 will appear. Select the item Remove Arrow. Arrowheads will be removed from the specified ends of the selected elements that have arrowheads. To abort this operation, click the left button outside any of the menus instead of selecting an item.

## **To Change the Way an Arrowhead Looks**

---

Select the Change command from the command menu, then select the element or elements whose arrowhead(s) you wish to change using the procedure described in the section “To Select Sketch Elements.” You can include elements that do not have arrowheads as long as the first element you selected allows them. A menu will appear that contains all the properties that can be changed in the first element you selected. It will include the item Arrowheads. Select it. The menu shown on the left in figure 31 will appear. Select the item that describes the end or ends on which the arrowheads to be changed occur. (These items are described in the section “To Add an Arrowhead to an Existing Element.”) After you select an item, the menu shown on the right in figure 31 will appear. See figure 30 for some examples of arrowheads that may help you understand the following instructions.

Selecting the item Wider will increase the angle of the specified arrowhead by 10 degrees. Selecting the item Narrower will decrease the angle of the specified arrowhead by 10 degrees. Selecting the item Larger will increase the length of the specified arrowhead by two screen points. Selecting the item Smaller will decrease the length of the specified arrowhead by two screen points. Selecting the item Triangle will add a base to the arrowhead. Selecting the item V-Shape will remove the base from the arrowhead. To abort this operation, click the left button outside any menu instead of selecting an item.

If you want to change several properties of some arrowheads, it may be easier to set the default arrowhead specifications to be like the arrowhead you want (see the section “To Change the Properties of New Arrowheads”), remove the existing arrowheads, and then add the arrowheads again. The newly added arrowheads will have the default properties.

## To Indicate That New Elements Should Have Arrowheads

---

Move the cursor over the Defaults command, press and hold the left button, and slide the cursor out the right side of the menu. Another menu will appear. In this new menu, with the left button still held down, position the cursor over the Line item and roll out its right side. A third menu will appear. In this menu, position the cursor over the Add Arrowhead item and release the left button. Two of the menus will disappear, and the menu titled Which End? shown on the left in figure 31 will appear. From this menu select the end of the elements you want to have arrowheads. Usually you will want to select Last. After selecting an item, whenever you add a line, curve, or arc, it will have an arrowhead on the specified end or ends.

To indicate that arrowheads should no longer be added to newly created elements, follow the above procedure but select the Neither item from the Which End? menu.

---

## To Change the Way Lines in Elements Look

---

Lines, polygons, curves, closed curves, boxes, ellipses, circles, and arcs have thickness (size), brush shape, and dashing properties. You change these properties using the Change command. Figure 32 shows some examples of different-sized lines.

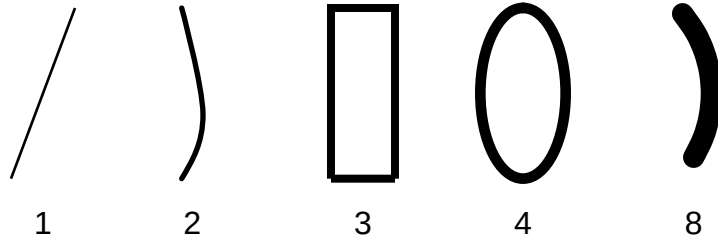


Figure 32. Examples of different sizes of lines

---

## To Change the Size of a Line

---

Select the Change command from the command menu, then select the element(s) you wish to change using the procedure described in the section "To Select Sketch Elements." You can include elements that do not contain lines as long as the element first selected does; the change will not affect the others. A menu will appear that contains all the properties that can be changed in the first element you selected. Select the item Size. The menu shown in figure 33 will appear. Selecting the item Smaller Line will make all the lines one point thinner. Selecting the item Larger Line will make all the lines one point thicker. Selecting the item Set Line Size will bring up a number pad menu with the title Enter the New Line Thickness. Enter the size you want the lines to be by selecting the digits as you would on a calculator. When you have entered the number, select OK; all the lines will be made that

thickness. To abort the command, select the item Abort from the number pad.



Figure 33. The menu of ways to change the line size of an element

## To Make a Dashed Line

Select the Change command from the command menu, then select the element(s) you wish to change using the procedure described in the section "To Select Sketch Elements." You can include elements that do not contain lines as long as the element first selected does; selected elements without lines will be unaffected by the command. A menu will appear that contains all the properties that can be changed in the first element you selected. Select the item Dashing. A menu similar to the one shown in figure 34 will appear.

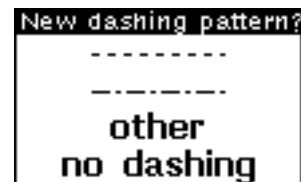


Figure 34. The menu of dashing patterns

Selecting one of the dashing patterns will make all the lines in the selected element(s) dashed with that pattern. Selecting the item No Dashing will make all the lines be solid; that is, it will remove the dashing from them. Selecting the item Other will prompt you for a new dashing pattern.

A dashing pattern is a sequence of numbers that indicates how many brush marks should be on and off. For example, the pattern (1 4 3 8) is , that is one on, four off, three on, eight off, repeated. When you select Other, a number pad menu with the title Number of Points On will appear. Enter the number of points you want to have on. When you have finished, select OK. A number pad menu with the title Number of Points Off will appear. Enter the number of points you want to have off. When you have finished, select OK. Number pads will continue to appear, giving you a chance to specify as long a dashing pattern as you like. After you have entered the last number in your dashing pattern, select OK when the number pad display has zero in it. A menu with the title Is This Pattern OK?, similar to the one shown in figure 35, will appear.



Figure 35. The menu presented to confirm a new dashing pattern

The first item will show the pattern you entered. If this is what you wanted, select the item Yes or the pattern; all the lines in the selected elements will change to that pattern. This pattern will also appear in the menu of dashing patterns the next time you change the dashing of an element. If you want to enter a different pattern, select the item No and you will be prompted for another dashing pattern. If you want to abort, select No and then select Abort from the number pad that appears. Selecting Abort from any number pad will abort the command.

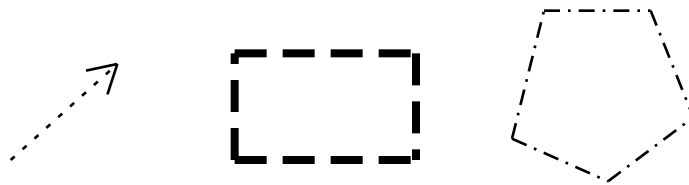


Figure 36. Examples of different dashing patterns for lines

Note: most printers do not support the dashing of splined curves, so curves, circles, ellipses, and arcs will not have dashed lines on hard copy. The effect will only be visible on the display.

---

## To Change the Brush Shape of a Line

Select the Change command from the command menu, then select the element(s) you wish to change using the procedure described in the section "To Select Sketch Elements." You can include elements that do not contain lines as long as the first element you selected does. A menu will appear that contains all the properties that can be changed in the first element you selected. Select the item Shape. The menu titled Pick a Shape shown in figure 37 will appear. Select the brush shape you want. When you have selected one, all of the lines will be painted with that brush. To abort the command, select outside any of the menus.

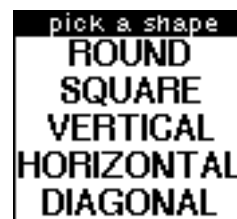


Figure 37. The menu of brush shapes

Note: because most printers do not support all of the available brush shapes, the effect may be visible on the display only.



## To Change the Way New Elements Look

The properties an element has when it is first added to a sketch are called the *default* properties. For example, there is a default size that determines how wide newly added lines, curves, boxes, etc., will be. All these defaults can be changed using the Defaults command and subcommands (see figure 38). If you want to add a collection of elements that are different from the standard, changing the default is often the easiest way to do it. For example, if you want a bunch of extra-thick arrows, you could change the default properties of new lines to have a thickness of two and an arrowhead on the last point specified before you enter the lines for the arrows. Any defaults that you change are saved when the sketch is saved.



Figure 38. Submenu for the Defaults command

TO

### Change the Properties of New Text

**1** Move the cursor over the Defaults command, press the left button, and slide out the right side through the triangle. The middle menu shown in figure 39 will appear. Move the cursor over the Text item and slide out the right through the triangle. The right menu shown in figure 39 will appear.

Error in IMAGEOBJ  
(GETFN: HRULE.GETFN)

Selecting the item Font Size will prompt you for the size that new text should have; supply it using the same procedure described in the section "To Change the Size of Text." Selecting the item Font Family will prompt you for the family that new text should have, which you specify using the procedure described in the section "To Change the Font Family of Text." If the specified font cannot be found in the current default size, an error message is printed and the default is not changed. Selecting the item Horizontal Justification will prompt you for whether the new text should be left, right, or center justified. Selecting the item Vertical Justification will prompt you for whether the new text should be top, bottom, middle, or baseline justified. For more information about text justification, see the section "To Change the Justification of Text." Selecting the item Bold and/or Italic will prompt you for the bold and italic properties that new text should have.

### To Change the Properties of New Text Boxes

Move the cursor over the Defaults command, press the left button, and slide out the right side through the triangle. The middle menu shown in figure 40 will appear. Move the cursor over the Text Box item and slide out the right through the triangle. The right menu shown in figure 40 will appear.

Selecting the item Horizontal Justification will prompt you for whether the text in new text boxes should be left, right, or center justified. Selecting the item Vertical Justification will prompt you for whether the text should be top, bottom, middle, or baseline justified. For more information about text justification, see the section "To Reposition the Text Within a Box." The font size, family, and bold and italic properties for new text boxes are the same as for text. See the section "To Change the Properties of New Text." The line thickness of the box is the same as the thickness of lines, so to change it see the section "To Change the Properties of New Lines."

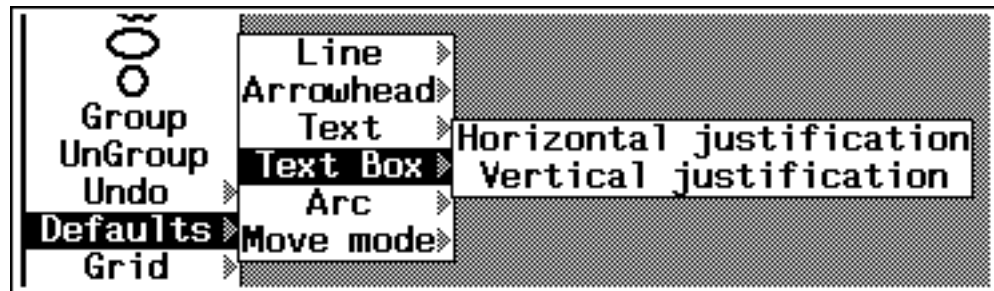


Figure 40. Submenu to set the default position of text inside a text box

---

## To Change the Properties of New Lines

Move the cursor over the Defaults command, press the left button, and slide out the right side through the triangle. The menu shown in figure 38 will appear. Move the cursor over the Line item and slide out the right through the triangle. A menu containing the items Size, Shape, Add Arrowhead, and Mouse Line Specs will appear. Selecting the item Size will prompt you for a number that will become the thickness of any new lines, curves, circles, etc. Selecting the item Shape will prompt you for a brush shape that will become the shape of any new lines, curves, circles, etc. Selecting the item Add Arrowhead will prompt you for which end or ends, if any, of new lines, curves, and arcs should automatically get arrowheads. To change the characteristics new arrowheads have, see below. Selecting the item Mouse Line Specs enables you to choose whether lines created by middle buttoning in the window should be connected.

---

## To Change the Properties of New Arcs

Move the cursor over the Defaults command, press the left button, and slide out the right side through the triangle. A menu that contains (among others) the item Arc will appear (see figure 41). Still holding down the left button, move to over Arc and slide out the right side through the triangle. A menu with the items Clockwise and Counterclockwise will appear (see figure 41). Selecting Clockwise will make new arcs go from their radius point to their angle point in a clockwise direction. Selecting Counterclockwise will make new arcs go from their radius point to their angle point in a counterclockwise direction.

The line thickness of arcs is the same as the thickness of lines. See "To Change the Properties of New Lines" if you want to change arc thickness.

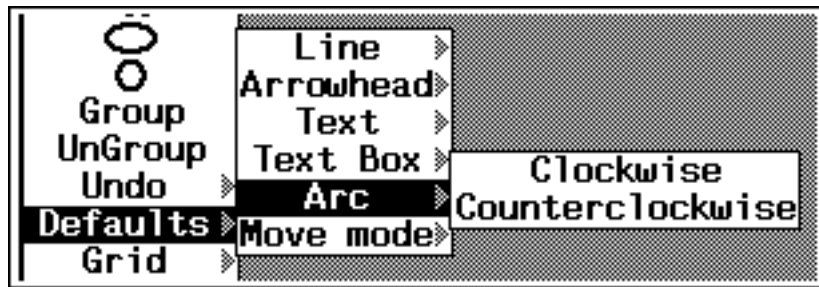


Figure 41. Submenu to change the default direction of arcs

## To Change the Properties of New Arrowheads

Move the cursor over the Defaults command, press and hold the left button, and slide the cursor to the right through the triangle. The menu shown in figure 42 will appear. In this new menu, with the left button still held down, position the cursor over the Arrowhead item and slide it to the right again. A menu containing the commands Size, Angle, and Type will appear (see figure 42).

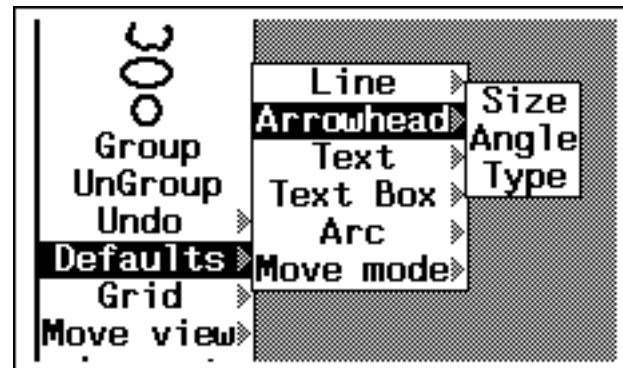


Figure 42. Menu of ways to change the default properties of arrowheads

From this new menu, select the arrowhead property you want to change. Selecting Size will bring up a number pad menu titled New Arrowhead Size in Screen Pts and giving the current default arrowhead size (length of the edges). Enter the number you want to become the size of new arrowheads, then select OK. You can abort the Size command by selecting Abort from the number pad menu.

Selecting Angle will bring up a number pad menu titled New Head Angle in Degrees, which gives the current number of degrees of the angle between the edges of new arrowheads. Enter the angle you want, then select OK. To abort, select the Abort item from the number pad menu.

Selecting the item Type will bring up a menu with the items V-Shape and Triangle, from which you can select the type of end you want new arrowheads to have. If you select V-Shape, your arrowheads will consist of two lines from the head. If you select

Triangle, new arrowheads will be triangles (two lines from the head and a line connecting their endpoints).

After this, whenever you add an arrowhead it will have the specified property.

---

## To Use Bit Maps in a Sketch

---

You can include bit maps and other types of image objects in Sketch drawings. This section describes the procedures for dealing with bit maps because they are a particularly useful kind of image object, but many of the procedures described here apply to other image objects as well. You can place a bit map in a sketch with the standard copy-select mechanism. A bit map has a single control point at its lower-left corner and can be moved, copied, and deleted like other elements. Advanced user note: applying the Change command to an image object or pressing a button down while over its image in the sketch window calls that object's `BUTTONEVENTINFN`, which is often an editor for the object.

---

## To Insert a Bit Map From the Screen

---

Move the cursor into the sketch window and click the left button. A caret () will appear. (If it doesn't appear, click again.) Move the cursor into the background (i.e., so it is not in a window). Hold down the copy key. On 1108 and 1186 keyboards, this can be either shift key or the Copy key; on Alto-style keyboards, it is either shift key. A menu with the single item Snap will appear. Select it.

The cursor will change to . Move the cursor to one corner of the region of the screen you want to include in the sketch. Press the left button. Holding the left button down, move the cursor to the opposite corner and release the button. While the left button is down, the region will be highlighted in gray. When you release the button, the message "Move the figure into place and press the left button" will appear in a small window above the sketch window. If this doesn't happen, you probably forgot to click in the sketch window first, and the bit map was inserted into whatever window was active. When the message appears, move the cursor into the sketch window and place the bit map image where you want it. When it is positioned, click the left button to insert the bit map.

---

## To Insert a Bit Map of a Pop-Up Menu

---

Images of pop-up menus are often useful in illustrating documentation (see, for example, figures 38 and 42). To insert a pop-up menu in a sketch, you must break the process that is popping it up using the following procedure. While the pop-up menu is visible, type the help interrupt character. Initially this is control-G (that is, hold the control key down and type G); however some systems move it onto control-H. When you have typed the help interrupt character, a menu containing the names of all the current processes will appear. Select the process that has popped up the menu, which is usually marked with an asterisk. It is often

the process Mouse. If you selected the correct process, a break window will appear and the menu will still be on the screen. If the menu disappears, you broke the wrong process; move the cursor into the break window that appeared, press the middle button, and select OK from the menu that appears. This will continue the process you did break. Bring up the pop-up menu again, type the help interrupt character, and select a different process from the menu. When you get the pop-up menu image to stay up, follow the procedure described above in "To Insert a Bit Map From the Screen." Important: when you have finished getting the image, move the cursor into the break window that appeared, press the middle button, and select OK from the menu that appears. This will continue the process that popped up the menu. If you forget this, strange things will happen when you next use the pop-up menu.

It is often helpful to consider the background onto which the pop-up menu will appear. This is because you often get parts of the background when you copy a menu image. And the bit map editor Trim command (see below and figure 43) only trims away white space. So if there are black background bits, you will have to edit them out using the bit map editor (see the next section). This step can sometimes be avoided by changing the background to white (by typing (CHANGEBACKGROUND WHITESHADE) into the executive window). It is also helpful to move other windows away from the area where the pop-up menu will appear.

---

## To Touch Up a Bit Map

---

Move the cursor over the bit map's image in the sketch window and press the left button. The menu shown in figure 43 will appear.

Select the Hand Edit item. The cursor will change into  and a large box outline will appear. This outline is the region the bit map editor will occupy. Move the box to the place on the screen where you want the bit map editor window to reside and click the left button. The bit map editor window will appear at that location (see figure 44).



*Figure 43. Menu of commands to edit a bit map.  
You obtain this menu by pressing the left button  
when the cursor is over a bit map image in a sketch window*

Edit the image by pressing the left or middle button in the large area at the bottom of the window. The image can be scrolled using the normal scroll bars if not all of it appears in the editing area. To quit, press the middle button while in the gray area at the upper-right part of the window. A menu will appear. Select OK to have the changes you made put back into the sketch. Select Stop if you want your changes disregarded. After you exit the bit map editor, the image in the sketch window is often incorrect. See the section "How to Clean Up the Display" for instructions on making it pretty again.

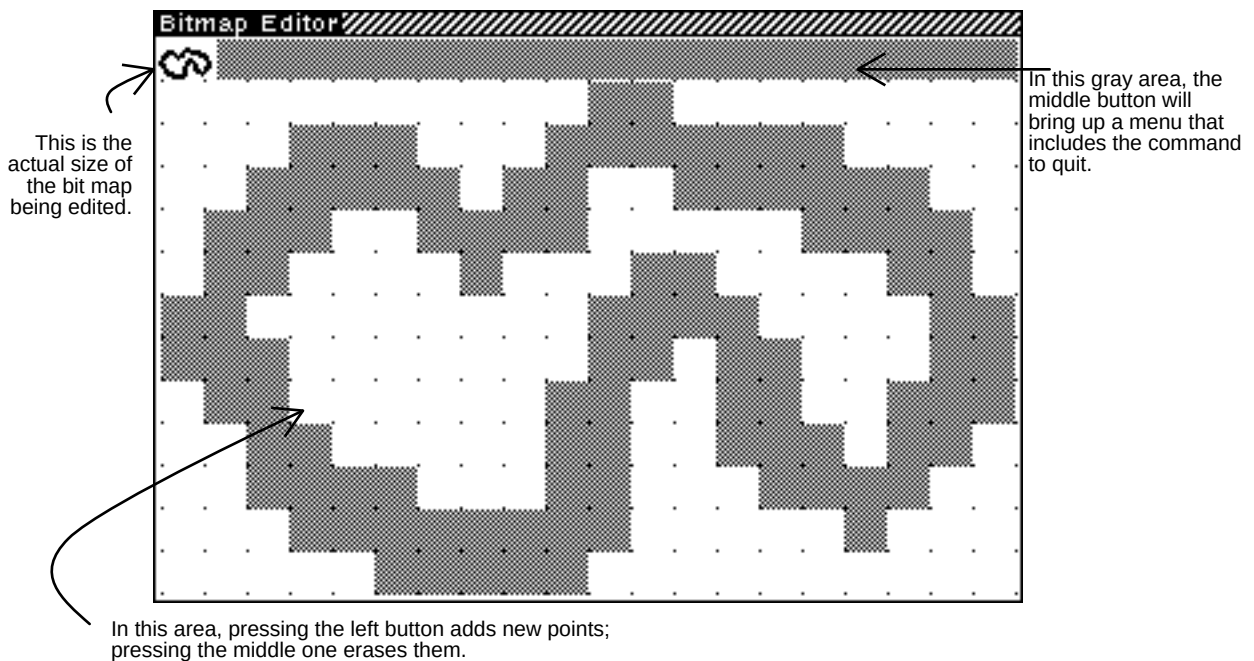


Figure 44. A bit map editor window

Another useful bit map editor command is Trim, available from the Operations on Bitmaps menu shown in figure 43. Trim will remove all the edge rows and columns that contain only white bits, making it easier to place lines and text around the bit map and saving storage space.

---

## To Put a Border Around a Bit Map

---

Move the cursor over the bit map's image in the sketch window and press the left button. The menu shown in figure 43 will appear. Select the Add Border item. This will prompt you for the number of bits you want in the border, then allow you to edit a four-by-four shade that will be put in the border. You can add multiple borders. For example, many of the bit maps in this document have two points of white surrounded by one point of black.

For a complete description of the bit map editor, see the EditBitMap documentation in the *Lisp Library Packages Manual*.



as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.



01UR.TEDIT

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:

{DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC  
 What are the requirements for saving a file in a crash? In a normal LOGOUT  
 and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the  
 upper and lower cases of the same letter as distinct symbols. E.g. if a  
 search string contains a capital X, a lower-case x will not qualify as a  
 match.Middle button On a 2-button mouse, pressing both buttons at once will  
 have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are  
 the requirements for saving a file in a crash? In a normal LOGOUT and boot  
 without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and  
 lower cases of the same letter as distinct symbols. E.g. if a search string  
 contains a capital X, a lower-case x will not qualify as a match.Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect  
 as pressing the middle button on a 3-button mouse. Local copy:  
 {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the  
 requirements for saving a file in a crash? In a normal LOGOUT and boot  
 without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and  
 lower cases of the same letter as distinct symbols. E.g. if a search string  
 contains a capital X, a lower-case x will not qualify as a match.Middle button  
 On a 2-button mouse, pressing both buttons at once will have the same effect  
 as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:  
 {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a  
 crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions  
 Case-sensitive Treating the upper and lower cases of the same letter as  
 distinct symbols. E.g. if a search string contains a capital X, a lower-case  
 x will not qualify as a match.Middle button On a 2-button mouse, pressing  
 both buttons at once will have the same effect as pressing the middle button  
 on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC  
 What are the requirements for saving a file in a crash? In a normal LOGOUT  
 and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the  
 upper and lower cases of the same letter as distinct symbols. E.g. if a  
 search string contains a capital X, a lower-case x will not qualify as a  
 match.Middle button On a 2-button mouse, pressing both buttons at once will  
 have the same effect as pressing the middle button on a 3-button mouse. Local  
 copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the  
 requirements for saving a file in a crash? In a normal LOGOUT and boot  
 without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and  
 lower cases of the same letter as distinct symbols. E.g. if a search string  
 contains a capital X, a lower-case x will not qualify as a match.Middle button  
 On a 2-button mouse, pressing both buttons at once will have the same effect  
 as pressing the middle button on a 3-button mouse.



lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.



**contains a capital X, a lower-case x will not qualify as a match.Middle button**

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? **In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions** Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:

**{ROSE}<REIDY>SPEC>TEDIT.DOC** What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols.

E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: {DSK}<LISPFILLES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC**  
 What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILS>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.Doc  
What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without  
Tedit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same  
letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will  
not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will  
have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC

What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQ PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a **capital X**, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILIES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? **In a normal LOGOUT and boot without TEdit PUT and QUIT?**  
**Definitions** Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK} <LISPFILES>REQT PUBLIC:

**{ROSE}<REIDY>SPEC>TEDIT.DOC** What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains acapital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: {DSK}<LISPFILES>REQT PUBLIC:**  
**{ROSE}<REIDY>SPEC>TEDIT.DOC** What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:  
 {DSK}<LISPFILES>REQT PUBLIC:  
 {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy:  
 {DSK}<LISPFILES>REQT PUBLIC:  
 {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy:  
 {DSK}<LISPFILES>REQT PUBLIC:  
 {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:  
 {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse,

pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.



Local copy: {DSK}<LISPFILS>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? **In a normal LOGOUT and boot without Tedit PUT and QUIT?**

**Definitions** Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:

**{ROSE}<REIDY>SPEC>TEDIT.DOC** What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols.

E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: {DSK}<LISPFILLES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC**  
**What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without**  
**TEDIT PUT and QUIT? Definitions Case-sensitive Treating the upper and lower**  
**cases of the same letter as distinct symbols. E.g. if a search string**  
**contains a capital X, a lower-case x will not qualify as a match.Middle button**

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFFILES>REQT PUBLIC:  
<ROSE><REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: (DSK)<LISPPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.Doc**  
**What are the requirements for saving a file in a crash? In a normal LOGOFF and boot without**  
**TEDIT PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same**  
**letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will**  
**not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will**  
**have the same effect as pressing the middle button on a 3-button mouse.**

Local copy: {DSK}<LISPFILS>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC  
What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFIL>REQ PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LSPFILES>RBQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? **In a normal LOGOUT and boot without TEdit PUT and QUIT?**  
**Definitions** Case-sensitive ~~Treating the upper and lower cases of the same letter as distinct symbols.~~ E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK} <LISPFILES>REQT PUBLIC:

**{ROSE}<REIDY>SPEC>TEDIT.DOC** What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols.

**case sensitive** Treating the upper and lower cases of the same letter as distinct symbols.  
E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.

**Middle button** On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: {DSK}<LISPFILES>REQT  
PUBLIC:**

**{ROSE}<REIDY>SPEC>TEDIT.DOC** What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:  
~~{DSK}<LISPFILES>REQT PUBLIC:  
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.~~

~~Local copy:~~

~~{DSK}<LISPFILES>REQT PUBLIC:  
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.~~

Local copy:

**{DSK}<LISPFILES>REQT PUBLIC:**

**{ROSE}<REIDY>SPEC>TEDIT.DOC** What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:  
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse,

pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILes>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT?

**Definitions** Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILes>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct

symbols. EUROPEAN

CLASSIC

| ^ ~ - ~ ~ ~ ~ ~ ~ ~ ~ ~ ~  
\ 1234567890, ^

ÁÉÍÓÚÀÈÌÒÙ Ø

áéíóúàèìòù ø

ÂÊÎÔÛÄËÏÖ:“”

âêîôûäëïö;»«

ZÅÇÃÕÑMÆ¿

ß å ç ã õ ñ æ œ / E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: {DSK}<LISPFILes>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC**

What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without

TEdit PUT LOGIC

CLASSIC

| ! @ # % ^ & \* ( ) \_ +  
\ 1234567890 = -

< [ ] { }  
> [ ] { } [ ]

∇ ⇔ [ ] - [ ] ≈ Σ [ ]  
∇ ⇔ ∇ [ ] - [ ] ∏ ⊕ [ ]

∃ [ ] ∞ [ ] ⇔ ∇ [ ] ⊆ ∅ →  
∃ [ ] ⊗ ∅ ⇔ ∇ [ ] ⊇ ∅ ←

TIMESROMAN

< [ ] [ ] ⇔ [ ] [ ] { }  
> [ ] [ ] ⇔ [ ] [ ] [ ]

∇ ⇔ [ ] [ ] - [ ] ≈ Σ [ ]  
∇ ⇔ ∇ [ ] [ ] - [ ] ∏ ⊕ [ ]

∃ [ ] ∞ [ ] ⇔ ∇ [ ] ⊆ ∅ →  
∃ [ ] ⊗ ∅ ⇔ ∇ [ ] ⊇ ∅ ←

and QUIT? Definitions Case-sensitive Treating the upper and lower cases of

```

: {ROSE}<READY>SPEC>TEDIT.DOC What are the
requirements for saving a file in a crash? In a
ts for saving a file in a crash? In a
normal mode and boot without TEDIT PUT and QUIT?
UI??Definitions Case sensitive Treating the upper and
as distinct symbols the same letter as distinct
quality has. E.g. if a search string contains a capital
s a word will leave the will not qualify as a match. Middle

```

ts for saving a file in a normal floppy and boot without Tedit PUT and QUIP?

UIP Definitions Case

as distinct symbols, the same letter as distinct qualify as a. E.g. if a search string contains a capital S a word will have the will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

```

Local copy:
{DSK}<LISPFILES>REQT PUBLIC:
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the
requirements for saving a file in a crash? In a
normal LOGOUT and boot without Tedit PUT and QUIT?
Definitions Case-sensitive Treating the upper and
lower cases of the same letter as distinct
symbols. E.g. if a search string contains a
capital X, a lower-case x will not qualify as a
match.Middle button On a 2-button mouse, pressing
both buttons at once will have the same effect as
pressing the middle button on a 3-button mouse.

```

Local copy: {DSK}<LISPFILES>REQT PUBLIC:  
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the  
requirements for saving a file in a crash? **In a normal  
LOGOUT and boot without Tedit PUT and  
QUIT? Definitions** Case-sensitive

Local copy: {DSK}<LISPFILS>REQT PUBLIC  
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements  
crash? In a normal LOGOUT and boot without TEdit PUT and Q  
sensitive Treating the upper and lower cases of the same letter  
E.g. if a search string contains acapital X, a lower-case x will not  
match.Middle button On a 2-button mouse, pressing both button  
same effect as pressing the middle button on a 3-button mouse.

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy:  
{DSK}<LISPFILS>REQT PUBLIC:  
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without Tedit PUT and QUIT?  
Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.  
Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy:  
{DSK}<LISPFILS>REQT PUBLIC:  
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the  
requirements for saving a file in a crash? In a  
normal LOGOUT and boot without Tedit PUT and OULT?

Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT?

Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match. Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.





ZĂÇĂŌÑMÆÆ¿

ßăçăđñ ææ/ E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.

Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: {DSK}<LISPFILES>REQT**

**PUBLIC:**

**{ROSE}<REIDY>SPEC>TEDIT.DOC** What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit

PUTLOGIC

CLASSIC

|!@#%~&\*( )±+  
\\1234567890-=

<[ ]>[ ]<[ ]>[ ]  
>[ ]<[ ]>[ ]<[ ]>[ ]

∇↔[ ]-[ ]≈Σ[ ]  
∇↔∇[ ]-[ ]Π⊕[ ]

∃[ ]∞[ ]↔[ ]>[ ]≤∅→  
∃[ ]=⊗↔[ ]>[ ]∅←  
TIMESROMAN  
<[ ]>[ ]<[ ]>[ ]<[ ]>[ ]  
>[ ]<[ ]>[ ]<[ ]>[ ]

∇↔[ ]-[ ]≈Σ[ ]  
∇↔∇[ ]-[ ]Π⊕[ ]

∃[ ]∞[ ]↔[ ]>[ ]≤∅→  
∃[ ]=⊗↔[ ]>[ ]∅←

and QUIT? Definitions Case-sensitive  
Treating the upper and lower cases of  
the same letter as distinct symbols.  
E.g. if a search string contains a  
capital X, a lower-case x will not  
qualify as a match. Middle button

On a 2-button mouse, pressing both buttons at once  
will have the same effect as pressing the middle  
button on a 3-button mouse. Local copy:

{DSK}<LISPFILES>REQT PUBLIC:  
{ROSE}<REIDY>SPEC>TEDIT.DOC What are the  
requirements for saving a file in a crash? In a  
normal LOGOUT and boot without TEdit PUT and QUIT?  
Definitions Case-sensitive Treating the upper and  
lower cases of the same letter as distinct  
symbols. E.g. if a search string contains a capital  
X, a lower-case x will not qualify as a match. Middle  
button On a 2-button mouse, pressing both buttons  
at once will have the same effect as pressing the  
middle button on a 3-button mouse.

Local copy:  
{DSK}<LISPFILES>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the  
requirements for saving a file in a crash? In a  
normal LOGOUT and boot without TEdit PUT and QUIT?  
Definitions Case-sensitive Treating the upper and  
lower cases of the same letter as distinct  
symbols. E.g. if a search string contains a  
capital X, a lower-case x will not qualify as a  
match. Middle button On a 2-button mouse, pressing  
both buttons at once will have the same effect as  
pressing the middle button on a 3-button mouse.

Local copy:

{DSK}<LISPFILES>REQT PUBLIC:

{ROSE}<REIDY>SPEC>TEDIT.DOC What are the  
requirements for saving a file in a crash? In a  
normal LOGOUT and boot without TEdit PUT and QUIT?  
Definitions Case-sensitive Treating the upper and  
lower cases of the same letter as distinct  
symbols. E.g. if a search string contains a  
capital X, a lower-case x will not qualify as a  
match. Middle button On a 2-button mouse, pressing

both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct

symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT?

~~Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.~~Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

Local copy: {DSK}<LISPFILES>REQT PUBLIC:

~~{ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols.~~

~~E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.~~Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

**Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC**  
**What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button**

On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse. Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

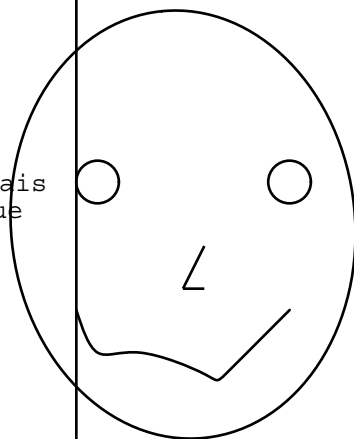
~~Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC~~  
~~What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.~~

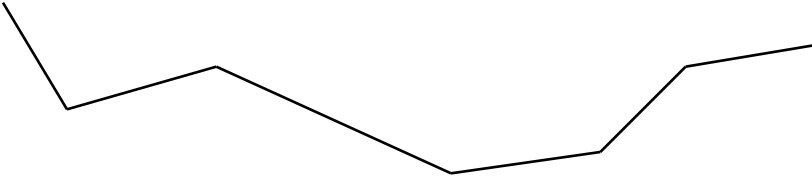
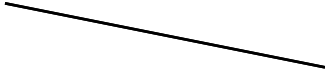
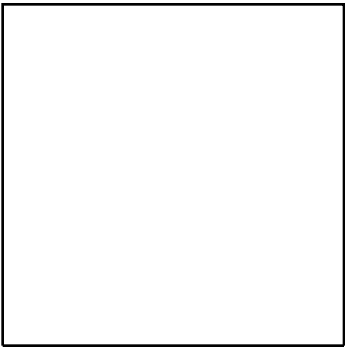
Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC  
What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

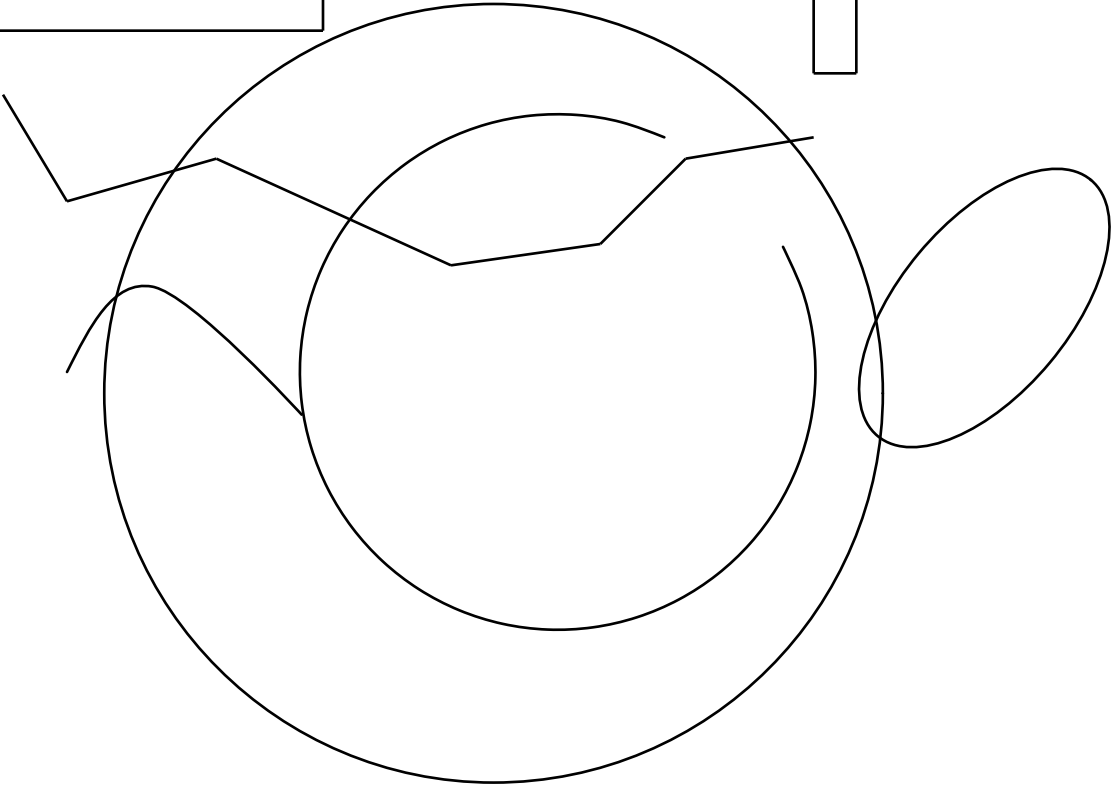
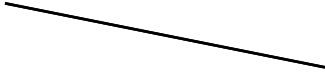
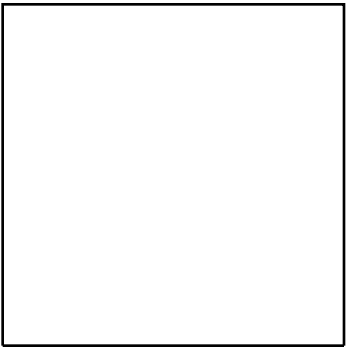
Local copy: {DSK}<LISPFILES>REQT PUBLIC: {ROSE}<REIDY>SPEC>TEDIT.DOC What are the requirements for saving a file in a crash? In a normal LOGOUT and boot without TEdit PUT and QUIT? Definitions Case-sensitive Treating the upper and lower cases of the same letter as distinct symbols. E.g. if a search string contains a capital X, a lower-case x will not qualify as a match.Middle button On a 2-button mouse, pressing both buttons at once will have the same effect as pressing the middle button on a 3-button mouse.

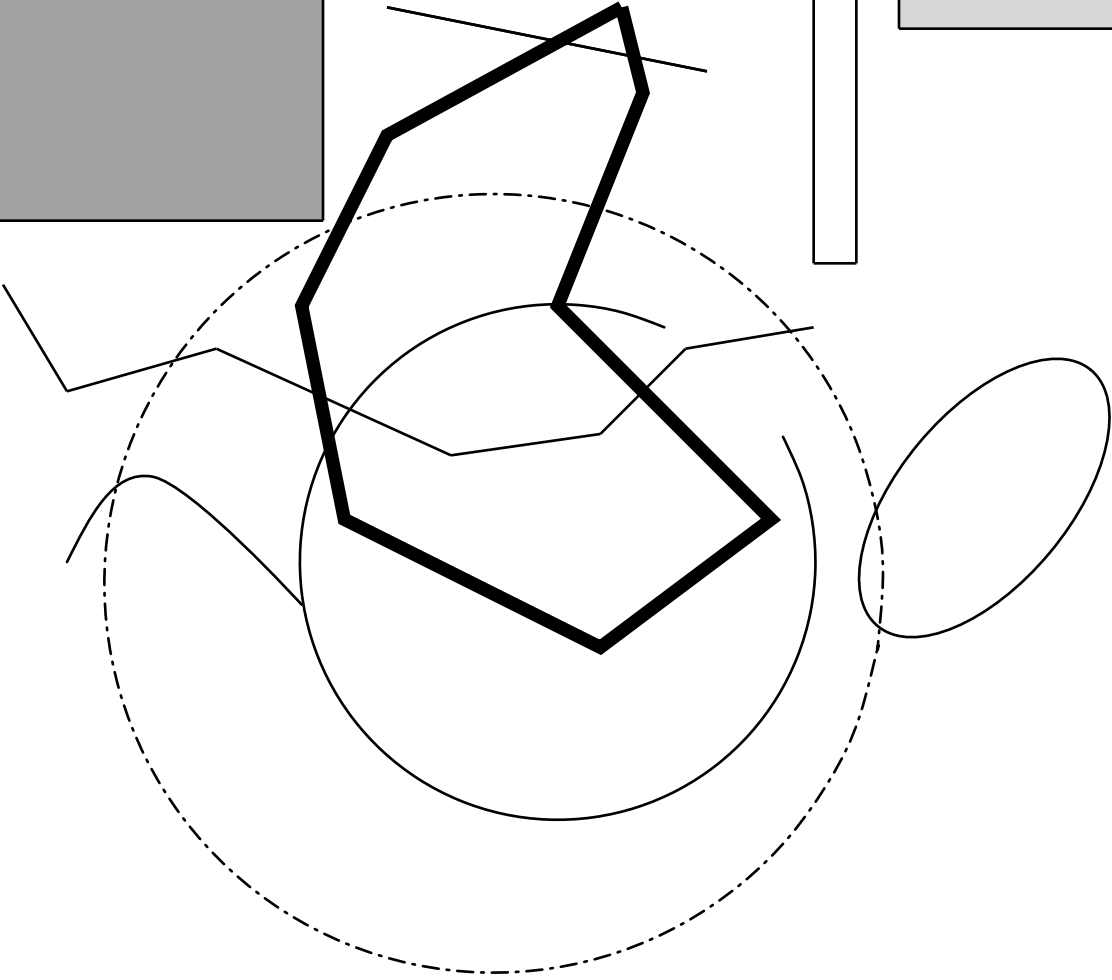
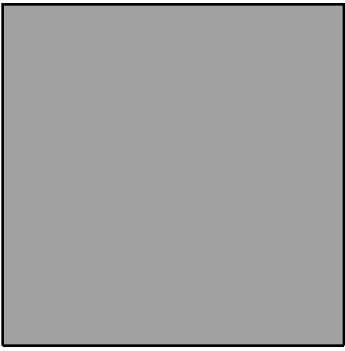


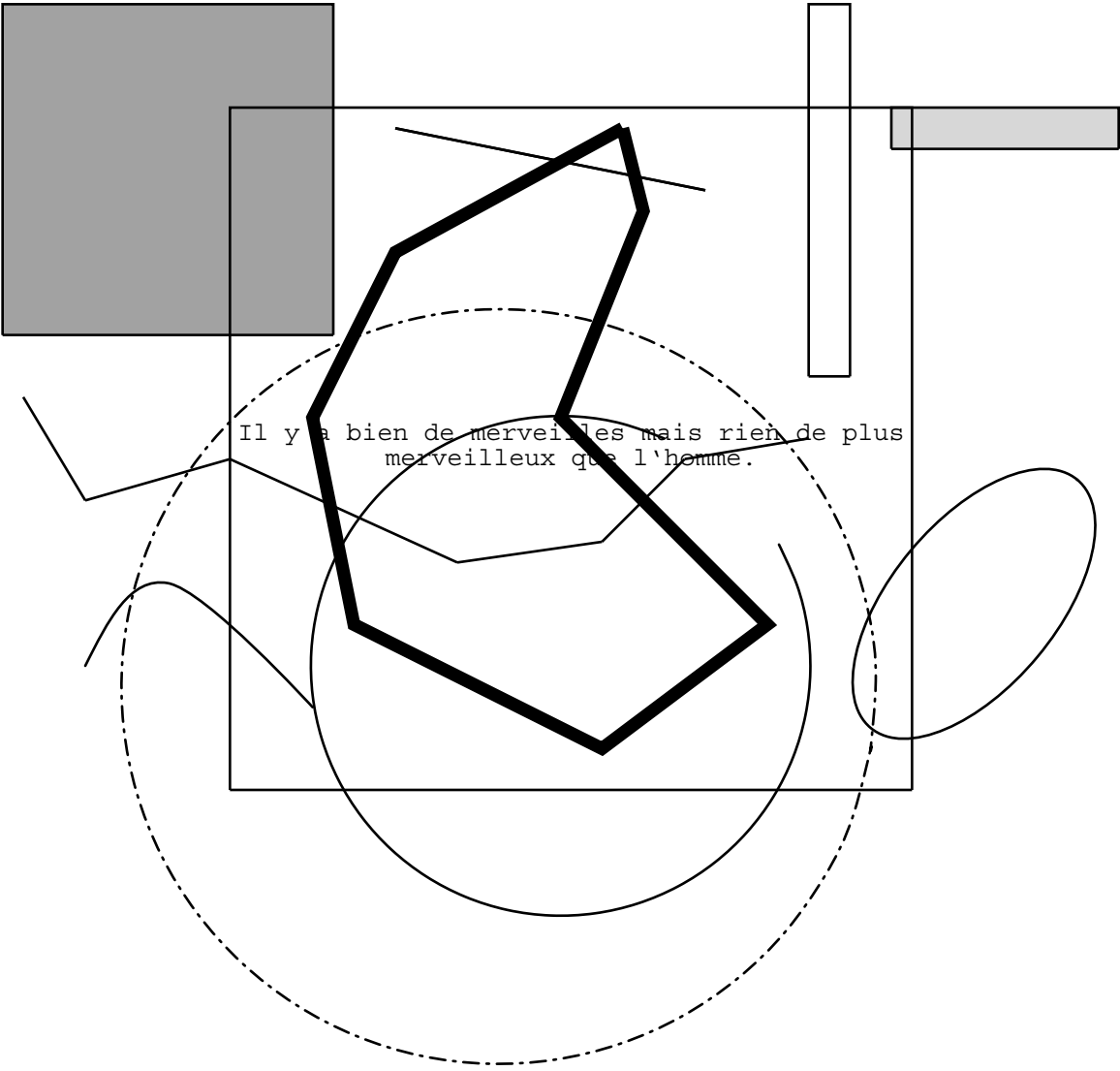
Il y a bien de merveilles, mais  
rien de plus merveilleux que  
l'homme.



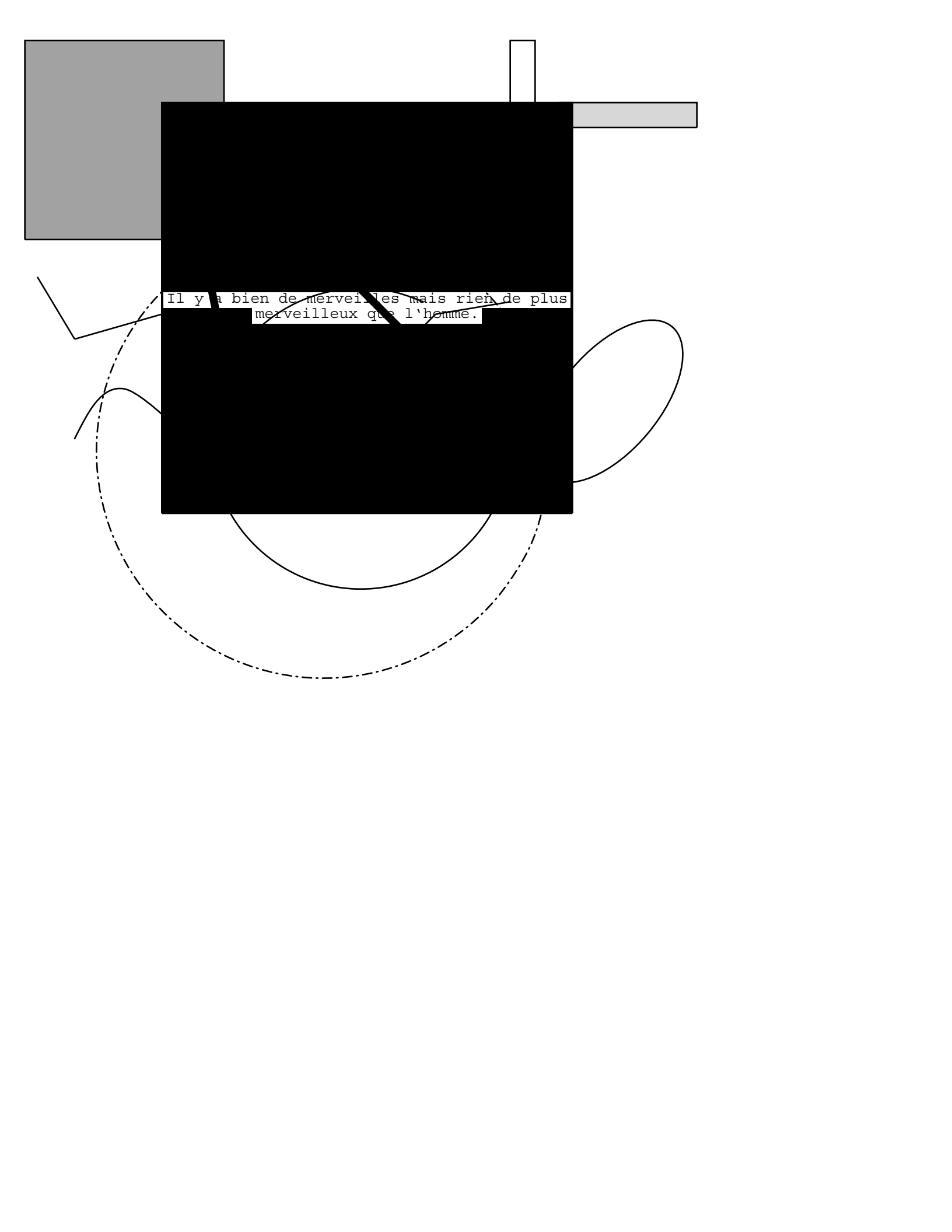








Il y a bien de merveilles mais rien de plus  
merveilleux que l'homme.



Il y a bien de merveilles mais rien de plus  
merveilleux que l'homme.

# XAIS Testing Directory Structure

## **Language: The Interlisp & Common Lisp languages, interpreters, compilers**

- Plans
- Results
- Auto
  - Test-Files {Eris}<Test>Language>Auto>
  - Aux-Files {Eris}<Test>Language>Auto-Aux>
- Hand
  - Scripts
  - Aux-Files

## **I/O: Streams, Windows**

- Plans
- Results
- Auto
  - Test Files
  - Aux Files
- Hand
  - Test Scripts
  - Aux Files

## **Prog Environment: Exec, Code Editing, Debugging, Application Toolkits**

- [subsection]
  - Plans
  - Results
  - Auto
    - Test Files
    - Aux Files
  - Hand
    - Test Scripts
    - Aux Files

## **Library: Individual Modules**

- [Module name]
  - Plans
  - Results
  - Auto
    - Test Files
    - Aux Files
  - Hand
    - Test Scripts
    - Aux Files

## **LispUsers: Repository for obsoleted library tests when modules are demoted**

- [Module name]
  - Plans
  - Results
  - Auto
    - Test Files
    - Aux Files
  - Hand
    - Test Scripts
    - Aux Files

## **Applications: "Customer Applications" for stress-testing system or subsystems**

- [Module name]
  - Plans
  - Results
  - Auto

- Test Files
- Aux Files
- Hand
  - Test Scripts
  - Aux Files

**[Subsystem, e.g. PCE]: One subdirectory per distinct product (PCE, Rooms, DEI, ...)**

- Plans
- Results
- Auto
  - Test Files
  - Aux Files
- Hand
  - Test Scripts
  - Aux Files

## File Naming

### —**.test**

A file of DO-TEST forms, suitable for running with DO-TEST-FILE or DO-ALL-TESTS. Within a directory, files are named x-y-z-w....test, with more general descriptions to the left, and more specific naming to the right. For example, there might be several tests of Lists: LIST-CREATION.TEST and LIST-MODIFICATION.TEST. It is poor form to include the words REGRESSION, TEST, or such like in the file name: Regression tests should be folded into the main test file, and all of these files are tests, so that's redundant.

### —**.u**

A script to be followed during hand testing. Within a directory, files are named x-y-z-w....u, with more general descriptions to the left, and more specific naming to the right. For example, the Exec might have several test files, e.g., Exec-PL.u, Exec-DIR.u, etc.

### —**.plan**

A plan for testing a section of the system, a new product like PCE, etc.

### —**m-d-y.log**

The results of running a test or series of tests.



This msg is stored on {eris}<lispcore>internal>library>do-test.tedit  
The tester is on {eris}<lispcore>internal>library>do-test.dcom.

### The main entries are the following:

(DO-TEST *name forms*)

A test succeeds if the final *form* returns a non-nil result. *Name* is just the name which can be an atom or string; strings are preferred. Forms are presumed to be read with the Common Lisp reader in package XCL-TEST, which uses LISP and XCL. If a test fails or an error occurs during evaluation, a message is printed to \*ERROR-OUTPUT\*.

(DO-TEST-GROUP *name&options forms*)

For associating a group of tests. For instance, a group of tests may all require the same setup and cleanup. If there are any options (see below) then the CAR of *name&options* is the name and the CDR is a keyword/value list. All *forms* must be DO-TEST forms.

(EXPECT-ERRORS *error-types forms*)

*Error-types* is a list of errors that may occur while executing the *forms*. If one of the listed errors occurs, EXPECT-ERRORS returns (values t error-that-occurred), otherwise NIL. Normal use of this form is:

```
(DO-TEST "a test"
 (EXPECT-ERRORS (type-of-error)
 (THIS-FORM 'SHOULD 'ERROR)))
```

(DO-TEST-FILE *filename*)

Reads and executes a file of tests. All forms in the file are read before any are executed. The file should be clear text (clearput in TEdit) and terminate with a STOP. The format for test names is Chap#[-sec#[-subsec#]]-comment.TEST

(CL-READFILE *filename*)

Reads all forms in *filename* and returns a list of them. This function is used by DO-TEST-FILE to read test files; test writers who want to see if their files are syntactically valid should first see if CL-READFILE will read them, then see if DO-TEST-FILE will execute them.

```
(DO-ALL-TESTS &key (results *test-batch-results*)
 (patterns *test-file-pattern*)
 (sysout-type nil)
 (resume nil))
```

Calls DO-TEST-FILE on each file that matches *patterns*, which is a list of directory patterns, and prints the results to a new version of a file named *results*. If *results* is T, results are printed to the window where DO-ALL-TESTS is running. The header of the results file is a message of the date and time the tests are being run and the MAKESYSDATE of the sysout; if *sysout-type* is supplied, a line for it goes out too. If *resume* is non-NIL, DO-ALL-TESTS attempts to resume an interrupted test sequence, appending the results onto the latest version of *results*.

TEST-SETQ, TEST-DEFUN, TEST-DEFMACRO

These work like SETQ, DEFUN, and DEFMACRO, except that if they are executed within a DO-TEST-GROUP, their effects are manually undone (old values are saved and then restored) upon leaving the DO-TEST-GROUP. Use these in :BEFORE forms that a whole group of DO-TESTs want to see. DON'T use TEST-SETQ on locally-bound variables or in loops.

### Relevant variables:

\*TEST-MODE\*

Default is :batch, which means to report test failures and errors on \*ERROR-OUTPUT\* (which is usually a file), and continue. Other values possible are:

:interactive which means to print a message before running each test, print another message for test failures, and produce a break window on errors.

:batch-verbose which means to generate all the messages of :interactive and do not break on errors.

\*TEST-BATCH-RESULTS\*

Defaults to "{eris}<lispcore>cml>test>test-results"

\*TEST-FILE-PATTERN\*

Defaults to ("{eris}<lispcore>cml>test>\*.test;" "{eris}<lispcore>cml>test>\*.x") which runs all the internal tests.

**\*TEST-COMPILE\***

If this switch is non-nil, DO-TEST compiles its forms before testing them. DO-ALL-TESTS will print a message in its header if this switch is on.

**\*ALL-FILES-REMAINING\***

While DO-ALL-TESTS is running, this variable contains a list of all the files remaining to be processed; files are removed from it AFTER they are read and executed. To restart a test run that somehow crashes the test driver, first clean up whatever blew up the run (if necessary, dump \*ALL-FILES-REMAINING\* to a file and get a new sysout), then do  
(DO-ALL-TESTS :RESUME T [:RESULTS "wherever"]).

**Options to do-test-group.**

:before        allows for a setup form for a group of tests.  
:after        allows a form to be run after the tests without affecting results.

The normal form of a DO-TEST-GROUP using all its features is:

```
(DO-TEST-GROUP
 ("a test group"
 :BEFORE (progn (before-form-1) (before-form-2) ...)
 :AFTER (progn (after-form-1) (after-form-2))
)
 (DO-TEST "first test")
 (DO-TEST "second test")
)
```

Add new abbreviation characters below, with a tab between. To test, expand the right-hand instance of each, and print the result.

|     |                  |
|-----|------------------|
| b   | b                |
| n   | n                |
| m   | m                |
| T   | fTf (this space) |
| d   | d                |
| D   | D                |
| s   | s                |
| ,   | ,                |
| ,   | ,                |
| "   | "                |
| ~   | ~                |
| 1/4 | 1/4              |
| 1/2 | 1/2              |
| 3/4 | 3/4              |
| 1/3 | 1/3              |
| 2/3 | 2/3              |
| c   | c                |
| c/o | c/o              |
| %   | %                |
| ->  | ->               |
|     |                  |
| ^   | ^                |
| <-  | <-               |

|          |                     |
|----------|---------------------|
| ̄DATE    | ̄DATE               |
| >>DATE<< | >>DATE<<            |
| L        | L (pound sterling)  |
| o        | o (degrees)         |
| Y        | Y (yen)             |
| +        | + (+/-)             |
| x        | x                   |
| /        | /                   |
| =        | = (goes both ways)  |
| p        | p (paragraph)       |
| r        | r (registered mark) |
| tm       | tm (trademark)      |

SADF ASDF ASD•F ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF ASDF

## Introduction

AGAST is an attempt to produce a program that can write intelligent stories. With an eclectic combination of ideas from the work of both computer scientists and writers, we have produced the flexible core of what could be a very intelligent story teller.

Work being done in cognitive science, natural language processing, and other areas that are closely based on human actions fits neatly into the project. Story-telling is not one isolated behavior, but a combination of many; work from text generation, decision-making, story-planning, character development and other areas is needed.

AGAST uses a formula similar to those developed by various professional writers to teach beginning authors how to write stories\*. This formula (described in detail in Part II) divides stories into five sections. AGAST attempts to model this formula by creating stories in the form of five inter-related sections.

AGAST is flexible because it "writes" stories in two stages. The first stage creates a story tree, where every action that happens is stored. As the tree is generated, the internal representation of the physical world (including locations, objects, and characters) is affected, which in turn affects the progress of the story.

A straight-forward depth-first traversal of the story tree produces a "chronological" account of the story. The second part, the text generator, walks the tree this way and thus tells the story. This part is extremely simple now; it just writes a sentence for each and every action, telling the story in excruciating detail. But since the story structure is unaffected by the telling, a different text

---

\* i.e. to capture what is essential in a story as opposed to a random collection of sentences, or some other form of prosaic writing such as a newspaper article or master's thesis

generator could easily be used before, after, or instead of the one used now. Sequences of events could be summarized to different levels as needed, events could be told in varying orders, or two stories could be meshed together.

The fact that the story exists as a tree after its generation means also that actions can be undone and the story can take a new direction in the retelling. AGAST uses this feature to handle stories that end in story-telling failure. Like a human writer, AGAST can "change its mind" and rewrite the story to end successfully. This also means that AGAST doesn't need to plan every detail of the plot ahead of time. It can randomly generate complications for the plot, handle them using any sort of decision making process, and know that if it paints itself into a corner, it can either undo the actions that got it into trouble, or change the situation so that the characters can successfully handle the problem.



## **Related Work**

### **A. James Meehan's "TALE-SPIN"**

"Tale-Spin" [Meehan 76] seems to be the grandparent of computer story writing--at least, everyone who does work in the field must refer to James Meehan's ground-breaking work.

Like Tale-Spin, AGAST has a physical world, with objects and locations. Both have actions, although AGAST's actions are arranged in a writerly (GIVE, PICK-UP) rather than a formal (PTRANS, MTRANS) fashion.

Tale-Spin's stories are generated by goal stacks--the original goal puts other goals on the stack, the achievement of all of which completes the original goal. AGAST's stories have this feature as well. However, like AGAST actions, AGAST goals are arranged in a writerly fashion, each broken down into story parts. In addition there is the possibility of having multiple goals, all being solved simultaneously.

One example of this is when a character is looking for two different objects. First one object is sought, and when it is found, the other is sought. However, if the second object is run across in the search for the first, it is picked up and the search for it is never initiated. Although it isn't yet implemented in AGAST, characters can have other character's goals as subgoals, thus helping friends achieve their major goals.

Also, in storing actions and their side-effects as they occur, AGAST allows story revision and backpatching, which aren't conceivable in Tale-Spin.

Tale-Spin does have some level of social interaction, which AGAST is at present totally missing.



## **B. Natalie Dehn's thesis**

Natalie Dehn [Dehn 81] makes the point that in writing a story, authors have a goal: to write an interesting story. Her project concentrates on author intentionality. AGAST attempts to emulate this goal with the plot formula that drives the story, and with backpatching that "saves" the story when it plots its way into a dead end.

However, there is also the point that characters must have goals. If they start without a specific goal, they are quickly given one, from simply staying alive to saving the universe. Dehn points this out (but not in terms of character goals) when she mentions justifying the situation a character finds him or herself in. A story goes wherever the author intends, but it won't be a very good story if the characters seem to be acting only on the author's whim. They should be following their own goals; their actions should make sense to them, not just to "The Story," of which characters generally aren't aware, anyway. AGAST attempts to combine the internal logic of goal-driven behavior of Tale-Spin with the author-intention-driven stories that Dehn promotes.

## **C. Michael Dyer's "BORIS"**

Michael Dyer's work [Dyer 81] is more on story understanding than on story generation. Dyer's BORIS attempts to understand stories not only by general semantic, grammatical and lexical knowledge but by discerning the context that the story creates. AGAST creates and stores its context, but so far makes only a limited use of it. One example in which AGAST uses the context of an event is when an accident occurs (a character is injured--they trip, or some

such accident). If the character is just travelling or exploring, they can cure themselves (but only if they're carrying a medikit). However, if they are fighting or escaping, they can't take the time to do anything about the injury.

While it would surely be interesting to have BORIS read in AGAST stories and answer questions about them, it would be more interesting to have a BORIS-like program enhance the context that AGAST builds. A memory of past events would allow characters to "learn" and would make social interaction easier to simulate. For instance, suppose Frank killed Libby's cat. When Libby next meets Frank, the past event might make her want to get revenge on Frank, and thus would influence what she did during the meeting. As in Tale-Spin, she would know Frank was not to be trusted--but she would conclude it rather than knowing it from the start. From the examples in Dyer's paper, it would seem possible to use such a system to determine characters' attitudes toward other characters and their current emotional states.

#### **D. Eduard H. Hovy's "PAULINE"**

The actual text of AGAST stories is generated very simply--every object in a story tree knows how to print a description of the action it represents. This produces very lengthy, boring text (see sample stories).

Eduard H. Hovy [Hovy 87] discusses a much better text generation model. His program, PAULINE, groups related actions together and summarizes them, specifically mentioning only the "high points" of the event. PAULINE interprets the actions, draws conclusions, and adds them to its knowledge of the event. PAULINE can also "shade" what it tells, adding evocative words that can slant the meaning of the text, although the event is still accurately portrayed.

These abilities would greatly enhance the "story-ness" of AGAST's stories.

Instead of:

**Libby swung her sword at the giant centipede, injuring its leg.**

**The giant centipede bit Libby, injuring her arm.**

...and so on, each exchanging many blows and ending with:

**Libby swung her sword at the giant centipede, injuring its head. Its head was severed and dropped to the floor. The giant centipede was killed.**

A program such as PAULINE might be able to produce more writerly text:

**Libby drew her sword as the giant centipede attacked. She slashed at the slaver creature as it bit at her. Howling with rage, the giant centipede sank its mandibles into Libby's left arm. Libby raised her sword and with a cry of desperation cut off the centipede's head.**

Since AGAST's actions are already grouped and catalogued (a series of "injure" actions that constitute a "fight" are stored in a slot of a "fight" event), summarizing events and choosing weighted words that fit the situation (desperate, rage) and the characters in it ("slaver", since giant centipedes are defined as non-intelligent animals) should be relatively easy to do.

## **E. Michael Lebowitz's "UNIVERSE"**

Michael Lebowitz's UNIVERSE program generates plots for soap-opera-like stories. This is different from most other work in the field in that stories in UNIVERSE are deliberately constructed not to end, but to have continuing characters moving from mishap to mishap. Below we examine two versions of UNIVERSE which have appeared in the literature.

[Lebowitz 83]

While AGAST is action-heavy, Lebowitz is primarily concerned with character consistency and development. Past events affect the personality (and thus actions taken) of characters. Like UNIVERSE, AGAST creates important

characters--the protagonist, the antagonist, anyone directly involved in the main goal--before the story starts. Unimportant characters--attacking monsters, for instance--are created on the fly.

AGAST reaches for this ideal with the Background section of the story providing the motivation for the story. However, UNIVERSE goes much further, with each character carrying around a changing history. Like Dyer's BORIS, a UNIVERSE-like program could significantly help the character development of AGAST's stories.

UNIVERSE also keeps track of character relationships in a more consistent way than AGAST. AGAST's characters can be related to one another (e.g. Libby is Frank's mother, and Frank is Libby's son), but if Frank is Stella's brother, all Libby knows is that she is somehow related to Stella. Relationships that change with time, such as marriage, exist, but are assumed permanent, with no history of past divorces or whatever.

[Lebowitz 87]

Using goal precedence and mutual-achievement criteria for goal selection, UNIVERSE nicely manages the interweaving of plots that is important for a complex, interesting story. AGAST currently has only one form of subplot implemented, the substory. Here the main goal is temporarily suspended while a subgoal (with a "subprotagonist") is "written" in a "meanwhile, back at the ranch..." type of story. The conclusion of the subgoal, usually with the subprotagonist and the protagonist joined as companions, then allows the completion of the major goal. The nesting of substories, however, can produce quite complex stories (Libby rescues Fred; they both rescue John; all three join with Natasha to continue the search for the Lost Ark of the Covenant...).

The "churning" of plots that UNIVERSE uses as one of its author goals is somewhat emulated by the introduction of obstacles and problems into the path of the protagonist. Plans have particular problems associated with them, and goals (which determine the type of story, such as the "quest") can also have special problems associated with them, especially at the climax of the story (which is a concept UNIVERSE doesn't have, since it writes "slice of life" narratives).

## **F. Schank and Abelson's scripts**

Schank and Abelson [Schank and Abelson 77] discuss many of the methods used in story writing programs. One important idea is that of scripts--an outline of how to behave in particular situations. They allow both understanding and generation of simple stories involving frequently done events, such as eating in restaurants or taking the bus.

Actions in the AGAST story tree are grouped and stored in PLAN objects. Many plans are similar to scripts in that they generate a restricted series of actions that constitute a type of event. For instance, if the event is a FIGHT between two parties: first, a character (randomly chosen from the first party) injures one (randomly chosen from the second party), then a return injury is done. These actions are repeated until one of the parties has no one left who can continue fighting.

ASDF ASDF ASDF ASDF ASDF ASDF  
QWER QWER QWER QWER QWER QWER

OKZXCV ZXCV ZXCV ZXCV ZXCV ZXCV.



## LESSON A7: FETCHING A NEW SYSOUT ON A DANDELION

B. Burwell December 7, 1987

Filed on : {Phylum}<ISLDOC>A07■DaybreakSysoutFetch

**Objective:** This document provides a recipe for the installation of a new copy of an Xerox Lisp "Sysout" (i.e., the main Xerox Lisp software), on a Daybreak which resides on a local network. You will be retrieving a new copy of the Xerox Lisp software from a network-based file. This document does not tell you how to load a new sysout from floppy disks.

### Discussion:

The Xerox Lisp software machine uses various resources, primary memory in particular, during its life cycle. Not all of these resources are relinquished and deallocated when no longer needed, and the machine tends to "fill up" after prolonged use. To avoid problems when running low on the available memory that Xerox Lisp will want to use, the software on the machine (i.e. sysout) should be periodically restored to its prior initial state. This is accomplished by loading into the computer a fresh copy of the software, which reflects a full complement of the resources that are supposed to be available. The frequency of such need varies according to use: busy programmers may require a new copy daily, while casual users using only the mail and editing facilities may survive several weeks or more. The need to load a new copy of the sysout may be heralded by seeing such error messages as "ARRAYS FULL". Also, when the system starts to get sluggish and it takes longer than usual for processes to complete this is generally an indication that a new sysout is needed.

Sometimes your machine may crash with an error that is unrecoverable until you get a new sysout. If you get a 0217, 1185 or some other error in the cursor, please refer to Lesson L5. If you are really stuck just get a new sysout.

The following assumes you have a working Dandelion that has an Xerox Lisp system installed and running. Additionally, this machine should be connected to an Ethernet and able to access the appropriate file server designated for your use. When you have completed the sequence of steps listed you should have a fresh copy of the Xerox Lisp sysout running and ready for service. All of your local disk files and permanently stored files on file servers will remain unchanged. Your personal additions may not change, unless you have previously modified your "user init file", which directs Xerox Lisp to customize your machine according to your individual specifications. (See Lesson H for more on Init files.)

In the following, comments are in *Italicized print*; you type the things that are underlined, and normal print represents statements issued by the computer.

## Procedure:

### 1. Saving your work and running the Lisp Installer

*IF THE MACHINE IS ALREADY ON and running Xerox Lisp, first save your work by doing the following: put any TEdit or Sketch files that you have made changes to, save any functions that you have defined, close any mail folders that you have open. Now logout by doing the following:*

(LOGOUT)

*IF THE MACHINE IS OFF, turn it on by moving the rocker switch on the main unit front panel toward the 1 position.*

*After a few seconds a number of icons will appear which will correspond with the function keys. Press the F1 key immediately followed by the 0 (zero) key. Do not hold both keys down at once.*

*This will look for and boot from a specific Mesa boot file on your local hard disk.*

*The Daybreak will come alive as a Mesa machine (i.e. running under the Pilot operating system).*

### 2. Installer Version 1.0

Copyright (C) 1984, 1985 by Xerox Corporation. All rights reserved.

Processor = 0AA007B74H = 25200075564B = 2■852■158■324

Memory size = 1536K bytes

Shall I try to find remote scripts? (Y / N): N <CR>

>Online

Drive Name: RD0 <CR>

*Pressing the carriage return at this point will bring the rigid disk (RD0) online.*

### 3. Installing a new sysout

*There are two choices for installing a new sysout. Since retrieving a new sysout from a file server on the Ethernet can take up to ten minutes, some people stash a fresh copy on*

*their LispFiles volume to save time. The sysout does take up space, though. If you already have a sysout on your LispFiles volume that you want to use, skip to step 5.*

*In this step you will actually fetch a new sysout onto the local disk. You will need to know ahead of time exactly which sysout you want and where it is stored. Refer to lesson M1 for a discussion about possible sysouts to load. The sysout is either on an NS file server or an IFS.*

*First you will open a connection to the file server and then fetch the sysout from that file server.*

### **For a sysout on an NS file server**

*If the sysout is on an NS file server (e.g. IE:PARC:Xerox) then you need to make sure you are logged in with your NS id (you were previously when connected to Starfile Public).*

> login  
User: your NS id (e.g. Joe Public:PARC:Xerox) <CR>  
Password: your NS password <CR>

> open  
connection to: your NS file server (e.g. IE:) <CR>

### **For a sysout on an IFS:**

*If the sysout is on an IFS (e.g. Phylum or qv) you will first need to login with your Grapevine name.*

> pup login<CR>  
User: your Grapevine id (e.g. Jones.pa) <CR>  
Password: your Grapevine password <CR>

> pup open  
connection to: your IFS (e.g. Phylum) <CR>

*Note: even if you give the name of an IFS that is running, it is possible to get an error returned. If you will get the message "No Route to host" simply execute the pup open command again until you get a connection. Alternatively, if you get the message "connection rejected by host" this means that the file server is very busy. The pup open command will keep trying to make a connection.*

4. *You now have a connection open to a file server. You want to ask the Installer to fetch the sysout onto a partition on the disk.*

> Lisp Sysout Fetch

Logical volume: volume name where you want sysout stored (e.g. Lisp) <CR>

Source: directory on file server where sysout is located (e.g. <Lisp>Lyric>Basics>Full.sysout) <CR>

Expand volume: (answer Y unless you are fetching the sysout to be a fresh Lisp sysout) Y <CR>

expanding. . . . (unless you said N above)

Shall I make this the physical volume: N <CR>

*Fetching a sysout usually takes 5 to 10 minutes.*

## 5. **Copying a fresh lisp sysout:**

*If you didn't fetch the sysout into a fresh lisp sysout then go onto step 6. You will now copy the fresh lisp sysout from the partition you fetched it into to the partition that you want to run Lisp in. After that you will expand the destination partition.*

> copy vmem

Source volume: your fresh lisp sysout volume (e.g. LispFiles) <CR>

Destination volume: your working lisp partition (e.g. Lisp) <CR>

copying.....done.

> expand volume

Logical volume: your working Lisp partition (e.g. Lisp) <CR>

expanding..... done.

6. **N.B. Unlike a Dandelion, the Daybreak requires that there be Lisp microcode in each volume that you want to run Lisp in.** You must have the right version of Lisp microcode which corresponds with the release of Lisp. For example, to run Lyric, you must use Lyric microcode. The Lisp microcode is available on Phylum, IE:PARC:Xerox and Starfile Public. In the following example, Lisp Lyric microcode will be retrieved from Phylum. Refer to Lesson A6 if Phylum isn't available. **If you are just getting a new sysout and are staying with the same release then continue on to step 7.**

> pup login<CR>

User: your Grapevine id (e.g. Jones.pa) <CR>

Password: your Grapevine password <CR>

> pup open  
connection to: Phylum<CR>

> Lisp Microcode Fetch  
Logical Volume Name: (the name of the volume e.g. Lisp) <CR>  
Source: <Lisp>Lyric>Basics>LispDove.db  
Shall I make this the physical volume? N <CR>

**7. Starting Lisp:** *You are now ready to go into Lisp.*

> Start Lisp  
Logical volume: your lisp volume (e.g. Lisp) <CR>  
Are you sure: Y <CR>

8. The screen will go dark and possibly display an odd pattern on the screen. The cursor will stop on 199 for a number of seconds and finally go to 1186. If the maintenance panel hangs on some other number refer to lesson L5.

*As Xerox Lisp looks for and activates different tools, the screen may change slightly. This process will be complete when the Xerox Lisp prompt "1>" returns to the 'TTY' window.*

*At this point you should now be in possession of a fully functional Xerox Lisp machine, and it is ready for you to resume your work.*

# Notes on {Medley}<internal>test>

## <test> subdirs:

admin, ARs, env, GC, IO , LANGUAGE, Library, loops, lyric, Maiko, tools

## <test>Top level files

**README.TEDIT** (this file)

**TEST-RESULTS** (contains log from running AUTO tests from 1988)

**DOT.read-me-first** (originally .read-me-first)

## <test>4045> Deleted

## <test>Maiko>

Subdir AUTO, OBSOLETE

Top level files moved to OBSOLETE

**STACKHAX** (has CHECKSTACKSPACE, seems to get tangled up in it's own stack)

**STACKTAKESI** (seems to cause a stack overflow on opurpose, which leaves stack clean enough that URAID hard-reset recovers from. Suspect timeouts aren't correct)

**BAD-XREF** (no compiled file)

**display.cl** (says "from Texas Instruments")

subdirs moved <ARs **optests** & .dfasl (**AuxHAND**

**ENDLESSPUSHES** **AR-TEST-CASES.Auto-log**

## <test>Maiko>AUTO>

**OPCODES.TEST**

**OPCODES.DFASL**

may need EXPORTS.ALL to compile

most tests succeed

test BITBLT-DIAGONALS and BITBLT-SLOPED-LINES fail

A little hard to debug because the inspector for 2D

Another cop y of **bbtests** and **optests.lisp**

## <test>Maiko>OBSOLETE>

Probably incorporated into AUTO>

**AREF-TESTER**

**ARRAY-TESTER.TEST**

**FLOAT-TESTER**

**MAIKO-UNWIND-TESTS**

**TESTER** (compiled OK)

**unwindtest**  
**xclopcodetests**

**<test>Library>**

4045xlpstream> RS232 deleted  
 CASH-FILE> only hand  
 GCHAX>Auto>  
 HASH-FILE> only HAND  
 MatMult>Auto>  
 TEdit>Hand-Aux> samples to TEdit  
 WHERE-IS> only

**<test>loops>**

**LOOPS-SETUP.TEDIT**  
**LOOPS-TESTER-....** files

**test>Lyric>**

(old-versions of **DO-TEST .dfasl** and **.tedit**)

**<test>Tools>**

**AUTOTEST.TEDIT** (originql AUTOTEST.TEDIT-orig)  
**AUTOTEST & .DFASL** framework for running tests

**DO-TEST & .DFASL** (copied newer version from Medley  
 internal/library)

**DO-TEST.TEDIT** on Writing Software Tests  
 (many other files not reviewed yet)

# XAIS Testing Directory Structure

## Special Files: For saving overall-testing information

|             |                                                                                            |
|-------------|--------------------------------------------------------------------------------------------|
| Information | {Eris}<Test>, files named <i>.name</i> , e.g. “.read-me-first” , this file.                |
| Procedures  | {Eris}<Test>Admin> for overall procedure files, rather than specific test plans or scripts |
| Results     | {Eris}<Test>ARs> for the results of AR test-case regression runs.                          |
| Tools       | {Eris}<Test>Tools> for general-purpose files like do-test or do-test-menu.                 |

## Language: The Interlisp & Common Lisp languages, interpreters, compilers

|            |                                |
|------------|--------------------------------|
| Plans      | {Eris}<Test>Language>Plans>    |
| Results    | {Eris}<Test>Language>Logs>     |
| Auto       |                                |
| Test-Files | {Eris}<Test>Language>Auto>     |
| Aux-Files  | {Eris}<Test>Language>Auto-Aux> |
| Hand       |                                |
| Scripts    | {Eris}<Test>Language>Hand>     |
| Aux-Files  | {Eris}<Test>Language>Hand-Aux> |

## I/O: Streams, Windows

|            |                          |
|------------|--------------------------|
| Plans      | {Eris}<Test>IO>Plans>    |
| Results    | {Eris}<Test>IO>Logs>     |
| Auto       |                          |
| Test-Files | {Eris}<Test>IO>Auto>     |
| Aux-Files  | {Eris}<Test>IO>Auto-Aux> |
| Hand       |                          |
| Scripts    | {Eris}<Test>IO>Hand>     |
| Aux-Files  | {Eris}<Test>IO>Hand-Aux> |

## Prog Environment: Exec, Code Editing, Debugging, Application Toolkits

|                                       |                                               |
|---------------------------------------|-----------------------------------------------|
| [subsection: Exec, Debugger, Editors] |                                               |
| Plans                                 | {Eris}<Test>Env> <i>subsection</i> >Plans>    |
| Results                               | {Eris}<Test>Env> <i>subsection</i> >Logs>     |
| Auto                                  |                                               |
| Test-Files                            | {Eris}<Test>Env> <i>subsection</i> >Auto>     |
| Aux-Files                             | {Eris}<Test>Env> <i>subsection</i> >Auto-Aux> |
| Hand                                  |                                               |
| Scripts                               | {Eris}<Test>Env> <i>subsection</i> >Hand>     |
| Aux-Files                             | {Eris}<Test>Env> <i>subsection</i> >Hand-Aux> |

## Library: Individual Modules

|               |                                               |
|---------------|-----------------------------------------------|
| [Module name] |                                               |
| Plans         | {Eris}<Test>Library> <i>module</i> >Plans>    |
| Results       | {Eris}<Test>Library> <i>module</i> >Logs>     |
| Auto          |                                               |
| Test-Files    | {Eris}<Test>Library> <i>module</i> >Auto>     |
| Aux-Files     | {Eris}<Test>Library> <i>module</i> >Auto-Aux> |
| Hand          |                                               |
| Scripts       | {Eris}<Test>Library> <i>module</i> >Hand>     |
| Aux-Files     | {Eris}<Test>Library> <i>module</i> >Hand-Aux> |

## LispUsers: Repository for obsoleted library tests when modules are demoted

|               |                                                 |
|---------------|-------------------------------------------------|
| [Module name] |                                                 |
| Plans         | {Eris}<Test>LispUsers> <i>module</i> >Plans>    |
| Results       | {Eris}<Test>LispUsers> <i>module</i> >Logs>     |
| Auto          |                                                 |
| Test-Files    | {Eris}<Test>LispUsers> <i>module</i> >Auto>     |
| Aux-Files     | {Eris}<Test>LispUsers> <i>module</i> >Auto-Aux> |
| Hand          |                                                 |
| Scripts       | {Eris}<Test>LispUsers> <i>module</i> >Hand>     |
| Aux-Files     | {Eris}<Test>LispUsers> <i>module</i> >Hand-Aux> |



## Applications: "Customer Applications" for stress-testing system or subsystems

```
[Application name]
Plans {Eris}<Test>Customer-Applications>application-name>Plans>
Results {Eris}<Test>Customer-Applications>application-name>Logs>
Auto
 Test-Files {Eris}<Test>Customer-Applications>application-name>Auto>
 Aux-Files {Eris}<Test>Customer-Applications>application-name>Auto-Aux>
Hand
 Scripts {Eris}<Test>Customer-Applications>application-name>Hand>
 Aux-Files {Eris}<Test>Customer-Applications>application-name>Hand-Aux>
```

## [Subsystem, e.g. PCE]: One subdirectory per distinct product (PCE, Rooms, DEI, ...)

```
Plans {Eris}<Test>subsystem>Plans>
Results {Eris}<Test>subsystem>Logs>
Auto
 Test-Files {Eris}<Test>subsystem>Auto>
 Aux-Files {Eris}<Test>subsystem>Auto-Aux>
Hand
 Scripts {Eris}<Test>subsystem>Hand>
 Aux-Files {Eris}<Test>subsystem>Hand-Aux>
```

# File Naming

### —.test

A file of DO-TEST forms, suitable for running with DO-TEST-FILE or DO-ALL-TESTS. Within a directory, files are named x-y-z-w....test, with more general descriptions to the left, and more specific naming to the right. For example, there might be several tests of Lists: LIST-CREATION.TEST and LIST-MODIFICATION.TEST. It is poor form to include the words REGRESSION, TEST, or such like in the file name: Regression tests should be folded into the main test file, and all of these files are tests, so that's redundant.

### —.u

A script to be followed during hand testing. Within a directory, files are named x-y-z-w....u, with more general descriptions to the left, and more specific naming to the right. For example, the Exec might have several test files, e.g., Exec-PL.u, Exec-DIR.u, etc.

### —.plan

A plan for testing a section of the system, a new product like PCE, etc.

### —m-d-y.log

The results of running a test or series of tests.

## AUTOMATED TEST HARNESS INTERFACES

This document specifies the interfaces to the automated tester harness. The harness is composed of two parts: the **top-level tester** and the **individual test handlers**. The name of the file to load for this is AUTOTEST.LCOM in the top level of the {MEDLEY}internal/test directory.

The top-level tester is set up similarly to the package FileBrowser. Items are selected in the same manner as FileBrowser, and are displayed similarly. The portions of the display are as follows (from top to bottom):

1. A prompt window for displaying messages and getting new input.
2. A command menu with the following commands:

TEST Tests sequentially each of the items selected in the test files window. Testing consists of loading the file containing the test suite, calling a function which has the same name as the NAME field of the filename (this function must return NIL iff the test suite is not successful), then undoing (as best as possible) the side-effects of loading and running the test suite. The function which is called is passed one argument: the name of the directory that the test suite came from (including the host name). If this item is selected with the middle button, then first it asks for the name of the file to direct output to (selecting this item with the left button will direct output to T, the process TTY display stream), before running the test suites. All output directed to NIL, the default output stream, will go to this file, including all error messages generated by the automated test harness and by TEST-MESSAGE (see below). It is assumed that no other activity is being performed while testing is in progress.

ABORT Aborts any tests in progress. Confirmation (via clicking the left mouse button) is required. New tests can be selected, tests can be re-run, etc. after an abort.

PAUSE Temporarily pauses any tests in progress. Any pause time does not count in the computation of timeouts (see below).

RESUME Resumes PAUSED testing.

DIRECTORY Does a directory of files (the directory pattern is prompted for in the prompt window) and puts them in the test files window in order to have a new set of test suites to select from.

PRINT Prints the results of testing the selected files. Selecting this item with the left button will print on the default printer. Selecting this item with the middle button will put up a menu asking whether to print to a printer or a file. If a printer is selected, then a menu asking for the printer to print to (gotten from DEFAULTPRINTINGHOST plus the selection "Other"; the latter will ask for the name of a new printer to print to) is up. Otherwise, if a file is selected, then the user will be prompted for the name of a file to print to (also, if the type of output is not obvious, i.e. PRESS or INTERPRESS, then the user will be prompted for the type of output). When the Hardcopy item of the right button menu is selected for this window, then this command is performed (except that selecting the main item does the default, while selecting either the printer or the file sub-item starts the sequence of questions at the intuitive place).

SUMMARIZE Similar to PRINT, except that it prints only those tests (out of the selected tests) which failed.

c. **QUIT** Quits testing, closing the window and throwing away all test results, test names, et stored in the window. If any tests are currently in progress, then confirmation (via clicking the left mouse button) is required in order to quit (in this case an ABORT is performed before quitting). When the tester window is closed, this command is performed.

3. A status window, which has the following fields:

**Suite** The name of the test suite currently running.  
**ID** The ID of the current test being performed by SINGLE-TEST.  
**Start** The time that the current test was started.  
**End** The time that the current test will time out at, or blank if none.

4. A summary window, which has the following fields:

**Files** The number of files in the test files window.  
**Selected** The number of files (test suites) selected in the test files window.  
**Completed** The number of test suites completed.  
**Successful** The number of test suites which were successful.

5. The directory pattern used to select the test suite files. Unless otherwise overridden, the directory pattern by default only selects the latest version of each test suite file. Also, unless otherwise overridden, the directory pattern by default only selects .LCOM files (if a source file is more recent than the corresponding compiled file, then an error message is displayed).

6. A heading line which identifies each column in the test files window.

7. The test files window which has a line for each test suite file which matches the directory pattern. The left button on an entry selects only that entry. The middle button on an unselected entry adds that entry to the selected entries. The middle button on a selected entry removes that entry from the selected entries. The right button in the left portion of an entry will extend the current entries to include this entry and all the entries inbetween (the mouse cursor will change to a right pointing arrow when this action is enabled). This window is also scrollable (both vertically and horizontally). When each test is completed, a line is drawn through the entry. This window has the following columns:

**Result:** The result of testing using the corresponding test file. The following can appear in this column:

? The test suite has not been completed or possibly even initiated, so no results are known.  
pass The test suite completed successfully.  
FAIL The test suite did not complete successfully. This could be because a test in the test suite returned bad results, a test in the test suite aborted, a test in the test suite ti out, etc..

med

**Name:** The NAME portion of the test suite file name.

**File:** The full name of the test suite file (except for the host name).

When the tester is loaded, a new entry is added to the background menu, labelled AutomatedTester. When this is selected, an automated tester process is started, which will prompt (in the system prompt window) for a directory pattern which is used to initialize the test files window.

The individual test handler is a function which is called by the top-level function of each test suite (the function which was called by the top-level tester). This function has the following interface (all arguments must be supplied):

Name: SINGLE-TEST (LAMBDA function).

Arguments:

IDENTIFIER The integer identifier of this test. Identifiers are assigned manually and are unique across all tests in all test suites. [We need to set up an index file for this purpose, in the standard test directory.]

ht EXPRESSION The expression to evaluate (e.g. (PLUS 2 3)). Note that in order to get the right results, this argument would normally be quoted with QUOTE (or ') or be an expression such as (QUOTE (fn)), where fn is a separately defined function (and is therefore compiled code, instead of interpreted code).

5)) PREDICATE The (one argument) predicate to check the result (e.g. (LAMBDA (X) (EQP X or NULL)). This must be NIL iff the result was not correct (non-NIL indicates that the result was correct). If more than one error can occur, then output identifying the specific error should be printed (to NIL). Note that this argument would normally be quoted with QUOTE (or ') or FUNCTION in order to get the right results.

he TIMEOUT The maximum elapsed (wall) time (in milliseconds) that the expression EXPRESSION should take to complete (NIL implies that no timeout is to be used). With the current Interlisp-D process mechanism, this will only work if the expression (or anything it calls) does a BLOCK, so that another process can check to see whether a timeout has occurred. Also, the timing is not exact, so the actual timeout used will be no less than the value supplied. Time elapsed while the test was PAUSED is not counted in checking for a timeout.

Result: NIL iff the test was not successful (due to PREDICATE returning NIL, a NOBIND being returned, a timeout occurring, or a deep exit (such as an abort) occurring). Non-NIL indicates success.

Description: This function evaluates the expression EXPRESSION and checks the result with the predicate PREDICATE, returning the result from calling PREDICATE. If NOBIND is returned from either EXPRESSION or PREDICATE, then an error message is printed (to NIL) and a NIL is returned from SINGLE-TEST. If the timeout is exceeded (and timeouts can be checked) then the evaluation of the expression is aborted and an error message is printed (to NIL) and a NIL is returned from SINGLE-TEST. If a deep exit occurred in either EXPRESSION or PREDICATE (e.g. from aborting of the expression), then an error message is printed (to NIL) and a NIL is returned from SINGLE-TEST.

Side Effects: A message can be printed (to NIL).

Assumptions: Deep exits completely out of EXPRESSION or PREDICATE are not part of the successful behaviour of either EXPRESSION or PREDICATE (any such exits must be caught internally within EXPRESSION or PREDICATE). Note that deep exits are caught via ERRORSET, so RETFROM, RETTO, RETEVAL, RESUME, etc. are not caught.

There is a function available which prints out an easily identifiable error message in a standard format to the standard output. This function has the following interface (all arguments must be supplied):

Name: TEST-MESSAGE (LAMBDA function).

Arguments:

IDENTIFIER The integer identifier of this test (as given to SINGLE-TEST).

TEXT The text of the error message.

INFO Information specific to this instance of this error.

Result: Not useful.

Description: The error message along with the test identifier and the specific information is printed to NIL in a standard, easy to notice format.

Side Effects: A message is printed (to NIL).

Assumptions: None.

Some side-effects of the automated test harness are:

1. The History List for the Programmer's Assistant is used, therefore old items are lost and a REDO, etc. immediately after testing will redo the last command that the automated test harness performed, not the last item printed in the top level typescript window.
2. The top level value and the value in the Programmer's Assistant of `HELPFLAG` are change for the duration of running a test suite.
3. Extra processes are run to perform the testing.

Known deficiencies with the implementation are:

1. ABORTing and PAUSEing can only be done between individual tests.
2. If a test is aborted between individual tests, but not between tests suites, then the effects LOADing and running that test suite are not UNDOne.
3. Some errors are not caught, and some side effects are not undone if errors occur.

Some possible extensions to this package are:

1. Utilities to help with testing for deliberate errors.
2. Utilities to help with automating input which would normally be manual.

Running DSKTEST  
The Disk-file-system test utility

1. Load the file DSKTEST.DCOM from whichever directory & server it is stored on.
2. Type  
(DSKTEST ' {DSK} <LISPFILES>

# # Medley

## ## What File to Load

Load the file {MEDLEY}internal>DO-TEST.DFASL

All the symbols mentioned in this document are in both the IL: and XCL-TEST: packages, unless otherwise noted.

## Main Testing Entry Points

(DEFTEST *name&options forms*)

[Definer]

This is the definer for tests, allowing them to be saved on file-managed files. The test succeeds if the final *form* returns a non-NIL result. If *name&options* isn't a list, then it's just the name which can be a symbol or string; symbols are preferred for DEFTEST tests. If you specify options, the CAR of *name&options* is the name. If you specify :COMPILED in *name&options*, the test will run only when it has been compiled. Since this test is stored as structure rather than as plain text, any symbols will be package-qualified appropriately. If a test fails or an error occurs during evaluation, a message is printed to '\*ERROR-OUTPUT\*'.<sup>1</sup>

Unless you have DFNFLG set to PROP, the act of defining a test also causes it to be run (so you'll see if your test fails right away).

Examples:

'''

```
(DEFTEST ISSUE-1000 ;use issue number in test name
 (= 3 (+ 1 2)))
```

```
(DEFTEST (+-OPT :COMPILED) ; A test of the compiler, only makes sense to run compiled.
 (= 3 (+ 1 1 1))) ; Checking that +'s optimizer does the right thing.
```

```
(DEFTEST (MS-TEST :INTERPRETED) ; A test of Masterscope, only makes sense interpreted.
 (TEST-DEFUN FN (X) (FOO X))
 (\. IS FOO CALLED BY FN))
```

'''

(DEFTESTGROUP *name&options forms*)

[Definer]

This is the definer for groups of tests, allowing them to be saved on file-managed files. For associating a group of tests. For instance, a group of tests may all require the same setup and cleanup. If there are any options (see below) then the CAR of *name&options* is the name and the CDR is a keyword/value list. All *forms* must be DEFTEST or DO-TEST forms.

Unless you have DFNFLG set to PROP, the act of defining a test group also causes it to be run (so you'll see if your tests fail right away).

:before allows for a setup form for a group of tests.

:after allows a form to be run after the tests without affecting results.

For example, a DEFTESTGROUP using all its features is:

```
(DEFTESTGROUP
 (UNWIND-OPCODE-TESTS
 :BEFORE (progn (before-form-1) (before-form-2) ...)
 :AFTER (progn (after-form-1) (after-form-2))
)
 (DEFTEST "first test")
 (DEFTEST "second test")
)
```

## ## Functions You'll Find Useful When Building Tests

(EXPECT-ERRORS (*error-types*) *forms*)

[Macro]

*Error-types* is a list of errors that may occur while executing the *forms*. If one of the listed errors occurs, EXPECT-ERRORS returns (values t error-that-occurred), otherwise NIL. If all you want to do is make sure that an error is signalled somewhere in the test, you can specify an *error-types* of T. Normal use of this form is:

```
(DEFTEST ERROR-CHECK
 (EXPECT-ERRORS (T)
 (THIS-FORM 'SHOULD 'ERROR)))

(DEFTEST (+-DETECTS-NILS :INTERPRETED)
 (EXPECT-ERRORS (XCL:TYPE-MISMATCH)
 (+ 3 NIL)))
```

(TEST-SETQ *Variable Value*)

(TEST-DEFUN *name (arglist) forms*)

(TEST-DEFMACRO *name (arglist) forms*)

[Macros]

These work like SETQ, DEFUN, and DEFMACRO, except that if they are executed within a DEFTEST or DEFTESTGROUP, their effects are manually undone (old values are saved and then restored) upon leaving the test. Use these in :BEFORE forms that a whole group of DEFTESTs want to see. **DON'T** use TEST-SETQ on locally-bound variables or in loops.

## Commands and Functions for Running Tests

run *Test-name*

[EXEC Command]

Once *Test-name* has been defined using DEFTEST or DEFTESTGROUP, you can run the test with the run command.

(DO-TEST-FILE *filename*)

Reads and executes a file of tests. All forms in the file are read before any are executed. The file should be clear text (clearput in TEdit) and terminate with a STOP. The format for test names is

'''

Chap#[-sec#[-subsec#]]-comment.TEST

```
(DO-ALL-TESTS &key (results *test-batch-results*)
 (patterns *test-file-pattern*)
 (sysout-type nil)
 (resume nil))
```

'''

Calls DO-TEST-FILE on each file that matches *patterns*, which is a list of directory patterns, and prints the results to a new version of a file named *results*. If *results* is T, results are printed to the window where DO-ALL-TESTS is running. The header of the results file is a message of the date and time the tests are being run and the MAKESYSDATE of the sysout; if *sysout-type* is supplied, a line for it goes out too. If *resume* is non-NIL, DO-ALL-TESTS attempts to resume an interrupted test sequence, appending the results onto the latest version of *results*.

' \*TEST-MODE\*'

[Variable]

Default is :batch, which means to report test failures and errors on \*ERROR-OUTPUT\* (which is usually a file), and continue. Other values possible are: :interactive which means to print a message before running each test, print another message for test failures, and produce a break window on errors. :batch-verbose which means to generate all the messages of :interactive and do not break on errors.

' \*TEST-BATCH-RESULTS\*'

[Variable]

Defaults to "{MEDLEY}tmp>test>test-results"

' \*TEST-FILE-PATTERN\*'

[Variable]

Defaults to "{MEDLEY}internal>test>\*.TEST "



'\*TEST-COMPILE\*'

[Variable]

If this switch is non-nil, DO-TEST compiles its forms before testing them. DO-ALL-TESTS will print a message in its header if this switch is on.

'\*ALL-FILES-REMAINING\*'

[Variable]

While DO-ALL-TESTS is running, this variable contains a list of all the files remaining to be processed; files are removed from it AFTER they are read and executed. To restart a test run that somehow crashes the test driver, first clean up whatever blew up the run (if necessary, dump \*ALL-FILES-REMAINING\* to a file and get a new sysout), then do

```
(DO-ALL-TESTS :RESUME T [:RESULTS "wherever"])
```

## ## internal Functions

(DO-TEST *name&options forms*)

[Macro]

This is the obsolete, plain-test-file testing macro; it is still around so that old tests work (and because DEFTEST uses it). A test succeeds if the final *form* returns a non-nil result. If *name&options* isn't a list, then it's just the name which can be an atom or string; strings are preferred. If you specify options, the CAR of *name&options* is the name. If you specify :COMPILED in *name&options*, the test will run only when it has been compiled. Forms are presumed to be read with the Common Lisp reader in package XCL-TEST, which uses LISP and XCL. If a test fails or an error occurs during evaluation, a message is printed to \*ERROR-OUTPUT\*.

(DO-TEST-GROUP *name&options forms*)

[Macro]

This is the obsolete, plain-test-file testing macro; it is still around so that old tests work (and because DEFTESTGROUP uses it). For associating a group of tests. For instance, a group of tests may all require the same setup and cleanup. If there are any options (see below) then the CAR of *name&options* is the name and the CDR is a keyword/value list. All *forms* must be DO-TEST forms.

:before allows for a setup form for a group of tests.

:after allows a form to be run after the tests without affecting results.

An example of a DO-TEST-GROUP using all its features is:

```
'''
(DO-TEST-GROUP
 ("a test group"
 :BEFORE (progn (before-form-1) (before-form-2) ...)
 :AFTER (progn (after-form-1) (after-form-2))
)
 (DO-TEST "first test")
 (DO-TEST "second test")
)
'''
```

(CL-READFILE *filename*)

Reads all forms in *filename* and returns a list of them. This function is used by DO-TEST-FILE to read test files; test writers who want to see if their files are syntactically valid should first see if CL-READFILE will read them, then see if DO-TEST-FILE will execute them.

(MUNG-TEST-FILES *filepattern* &key (*compiler* 'compile-file)  
(*startinglist* NIL))

Compiles test files so they can be run by just loading them. Compiles all files matching *filepattern* (which is fed to directory) using *compiler* and writes them out to the directory they came from with an extension appropriate to *compiler*. If you want to explicitly specify the list of files to compile, hand a list of pathnames to *startinglist*. Prints an error message for files that fail to compile. You have to use this function (instead of just compiling the test files) because it prefaces the test files with

'''

```
(in-package "XCL-TEST")
(setq *test-file-name* "NAME-OF-FILE")
, , ,
```

so the compiler will read them properly and the files will know their names for error reporting purposes. **NOTE:** tests that fail should not be compiled; the resulting compiled code may not be a valid test.

## THE INTERLISP-D TESTING SYSTEM

The Interlisp-D testing system is an integrated system built for creating, managing and using a large set of programmed tests for testing the correctness and the performance of the Interlisp-D programming environment.

The system is consisted of three parts : The test driver, the data base management system, and a graphic control tool. In addition, there are various tools for helping the test builders in the process of creating new tests.

All parts of the system assumes the structure of a TEST which is a data type consists of several fields, of which the most important are the expression which has to be evaluated, and a predicate which takes the results of this evaluation and determines whether the test was a success or a failure (i.e. whether the actual result is the same as the expected result).

The test driver is in principal a function which gets an object of type TEST, performs the test, and return either success or failure plus some additional information. It includes facilities for monitoring the test execution, tracing and recording the testing process to enable reproducing tests, Remote Eval protocols to enable performing tests with two machines and more.

The data base management system works in two levels. In the low level, the "test cluster" level, the system manages and organizes the tests in the file system, enable retrieving tests through a caching system, and allows concurrent access to test files using a simple locking scheme.

In the high level, the system enables each user to manipulate the database using its own VIEW of the system. This view is implemented through the CONCEPT SPACE which is a directed acyclic graph that will usually reflect the logic structure of the system as seen by the user.

The graphic control tool displays a concept space as a graph and allows the user to perform most of the Test system operations by selecting nodes from the graph.

The tools for building and manipulating tests include test inspector (and editor), a random generator which can generate random specified Lisp objects, Indirect reference to other tests in TEST fields for shrinking the space of the tests themselves and avoiding redundant work when creating tests which share some of their fields, and more .

In the next sections the different parts of the system will be described as well as the interaction between them.

## The TEST data type

The TEST is the data type of test objects. Its structure reflects the various properties that tests have. It includes the following fields:

TestID : The tests are identified by an integer.

Input : This field contains an expression that, when evaluated, will generate the list of arguments on which the tested expression will be applied. There are several tools which help in creating this entry. The random generator helps in generating random objects with specified restrictions. The SYSTEMATIC operation helps in generating systematically all the combinations over finite ranges.

Expression: It can contain a function name, a lambda expression or arbitrary sexpression. In the first two cases it will be applied on the input.

Success Predicate: This field contains a lambda expression with two arguments - the ACTUAL input for the test, and the result of the evaluation. It returns one of the two atoms: Success or Failure. When performing tests with random input, some tricks may have to be used as demonstrated in the examples in the end.

Timeout :This is a lambda expression which gets the ACTUAL input as an argument, and produces an upper limit to the estimated time of evaluation.

EvalBefore and EvalAfter: expressions to be evaluated before and after the test execution. usually, before the test we may want to set the appropriate environment for the test (like loading certain files), and after the test we may want to clean up the environment (like deleting files which the test created).

Pretests: Contains a list of links to other tests. These links may influence the order of an execution of a set of tests. Currently there are two types of links. A STRONG link to other test means that whenever the current test is going to be executed, the pretest must be executed first. An example for such pretest may be tests for the tests themselves. If a test generates a few thousands combinations of some arguments, it may be useful to test first if the test itself works correctly by executing a simplified test which works only on one set of arguments, and check that the test outcome is reasonable. A WEAK link to other test means that whenever a SET of tests is being executed, and both the test and pretest are in this set, the pretest will be executed before the test (thus it defines a partial order on any set of tests). This link may be used in cases where there is logical order on the execution of tests - for example, it is reasonable to test opening a file before testing writing to a file.

## The Test Driver

The test driver accept a test as its input and returns either success or failure. It will evaluate the input and the tested expression itself on a remote machine if requested, or on the local machine otherwise. All the process of the testing is recorded on a trace file, such that as much information as possible will be available if needed.

The driver evaluates the EVALBEFORE form, evaluates the input expression to generate the input for the tested expression, applies the tested expression on the generated input, applies the success predicate on the result and the generated input, and evaluates the EVALAFTER form. After some of the above stages the appropriate information is written on the trace file. The most important one is the input generated, especially in cases of random input.

All the evaluation done by the driver uses the Interlisp-D ERRORSET command, thus allowing evaluation that will not break under error condition. The error type may be used by the success predicate to determine if the result is a success or a failure. Thus one test for many arithmetic functions can be to supply them with non numeric arguments and to check that the error reported will be the right one.

The evaluation of the tested expression is done as a separated process, such that the driver will be able to try to interrupt it in case where the time of execution is larger than the value of TIMEOUT field of the test. This interrupt will work only if the test execution process will release voluntarily the cpu (when waiting on I/O for example) since Interlisp-D uses non preemptive scheme for process scheduling.

Remote evaluation will not benefit us much in this type of problems. If the remote machine is in infinite loop for example, it will not listen to interrupt attempts as well. The advantages of using remote machine are two: If a long sequence of tests are executed, and the machine "freezes", a remote test will freeze the remote machine and the local machine will be able to call for help and resume operation (as soon as the remote machine does not respond for more than some estimated limit of time, the local machine sends messages to a preset distribution list and asking for human help). A second benefit of remote evaluation is when we need to evaluate the tested expression in a different environment than the Testing system resides. We will want the testing system to work in considerably different environment (software release), while we are testing an experimental different environment.

### **The data base management system: the "test cluster" level.**

The user can retrieve a test by calling the GetTest function. The low level of the dbms is responsible for performing the appropriate operations to retrieve the requested test. If the test is not already loaded it will be loaded from its file. There is a limit on the number of the tests that are loaded, and if this number is exceeded a replacement will take place and a test will be removed. The replacement policy is LRU (least recently used) and is implemented by moving each test being referenced to the front of the list of the loaded tests. Thus the last test in the list will be the one to be removed. The limit on the number of loaded tests is dynamically modified according to the amount of the available memory.

The Interlisp-D testing system is designed to work with several users who use it concurrently. There are no problems if the users were only retrieving tests from the data base. Problems may occur if two users modify the same part of the data base in the same time.

For such cases a locking scheme was integrated into the system. There is a special designated file which is the "gate" for the data base. Users can obtain write LOCKS on tests. The file contains the list of users with their locked tests. The basic locking function is ObtainDatabaseWriteLock(testnumber) which checks the LOCK file, and registers the tests that are not already locked. The user has the option of

automatically generated messages that will be sent to the locking users, inform them that somebody is waiting for their locked tests and request them to release them as soon as they are not needed.

Thus, either automatically or manually, whenever a user edits a test, he will first obtain LOCK on tests. The locking scheme will work only if the users will follow the rules and will not try to access tests not through the testing system.

The basic operation - ObtainDatabaseWriteLock is "atomic" in the sense that the LOCK file is opened for read and write throughout the execution of this procedure, and thus no other user will be able to open it. The time interval in which the file is opened is very short.

Another problem that may arise from concurrent access to the data base is test numbering. As mentioned above, each test has a unique integer as an ID. Thus there is a file which contains the last ID issued, and the procedures for creating new tests will access and update this file.

### **The "Concept Space" level.**

What is the "thing" which is being tested by the test? it may be a specific low level system function, a library package, or a new representation scheme for integers. It is hard to find a common class to which all these entities belong. Thus the testing system assumes that it is some CONCEPT of the Interlisp-D system that is being tested.

While it is true that when a test is CREATED, its creator intent to test a specific concept, the test itself is not necessarily a test only for this concept. A test that was built for testing the READ function, may actually test also the NS communication protocols, the OPENFILE function etc.

For this reason the tested concept is not considered to be a part of the test itself. There is a separated knowledge space which is the way that the user views the test cluster. A concept space is an acyclic directed graph of CONCEPTS. Each node of the graph is of type CONCEPT which has four fields: The concept name, the tests that tests this concept, the subconcepts and the superconcepts.

The main purpose of the concept space is to enable the user to group tests in a logical way and to perform operations on these sets of tests. The semantic of a concept node is : "the tests which tests this concept are the tests of the concept itself plus the tests that tests its subconcepts (recursively)".

Such a definition allows us to build concept spaces which view the tests from different points of view. We may have a concept named "Arithmetic system" with subconcepts "Integer arithmetic", "Float arithmetic" and "Arithmetic functions". The "Arithmetic functions" will have as subconcepts, the concepts "IPLUS", "FPLUS", "PLUS" etc. "IPLUS" is also a subconcept of "Integer arithmetic", and "FPLUS" of "Float arithmetic". Thus, if a new representation for the integers was introduced to the system, we will test the "Integer Arithmetic" concept, while in other cases we may want to test the "Arithmetic Functions".

From the above example and from the more detailed example at the end, we can conclude that the organization of the test system should be very flexible, since there can be many parallel views into the same part of the system. We can also see why a tree structure would not be sufficient as a representation scheme. The Testing system supports the co-existence of several concept spaces, and thus each user can build and use his own concept space(s) to reflect his view of the system.

## THE CONCEPT SPACE BROWSER

Most of the operations of the Interlisp-D testing system are done through the "Concept Space Browser". The browser is a graphic tool which is applied on a concept space.

It has a few types of operations. Any operation that require a concept as an argument will get it by a selection from the displayed graph.

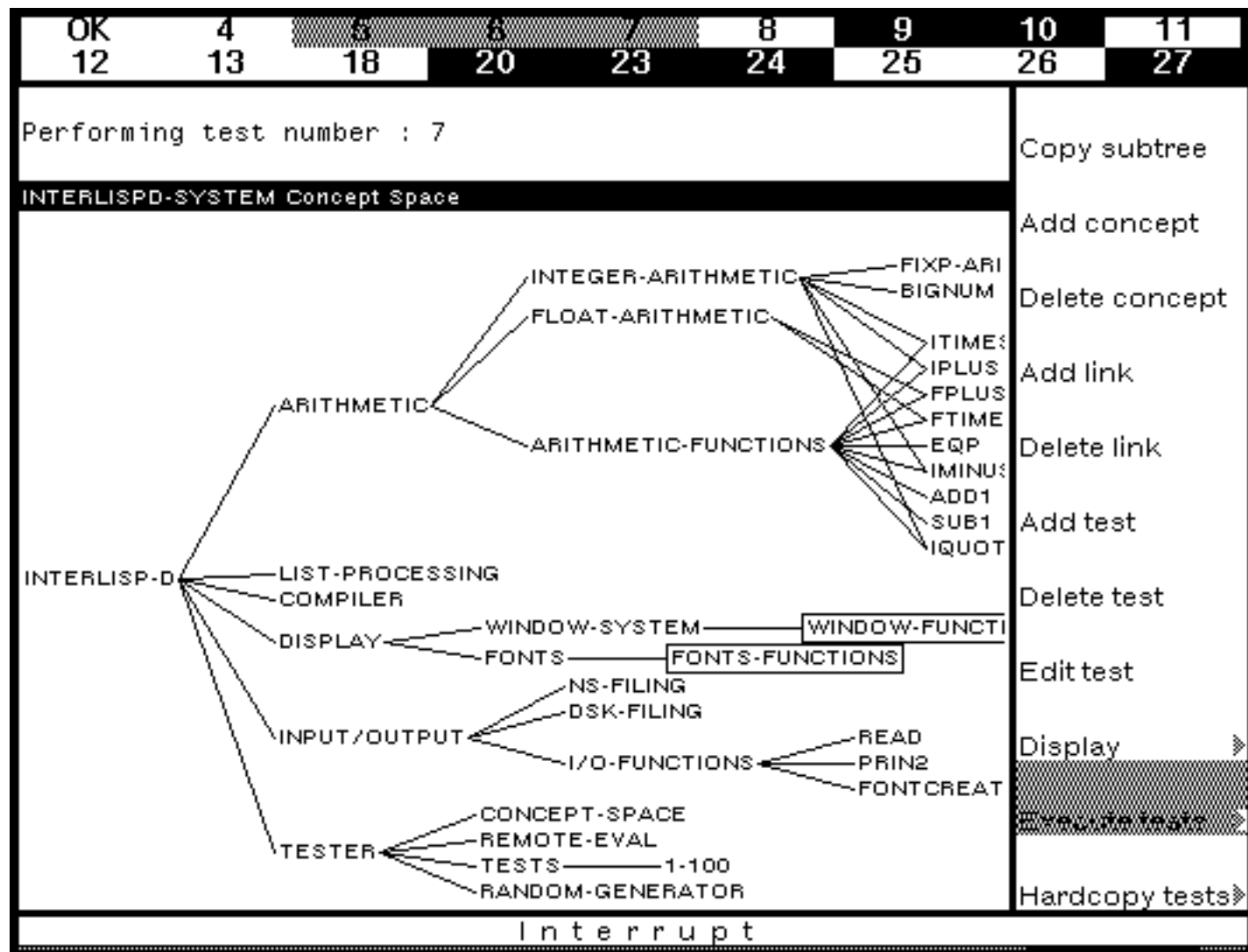
The first type of operations are operations for modifications of the concept space itself. There are commands to add new concept, to delete a concept, to add and delete a link and to add and delete a test to a concept.

Second type of operations are data base operations. The user can edit a test selected from specific concept, can hardcopy all (or part of) the tests of a selected concept, and can request to lock all (or part of) the tests of a concept.

A third type are display op can be specified, a browser of a subgraph can be created, and the tests can be dre are commands to execute all (or part of) the tests of a certain concept, with different modes of execution .

The browser also allows to copy subgraphs between two displayed concept spaces, and to get all the tests of a node by a copy selection so that functions that are not available in the browser can use the concept space as well.

.



### Test creation tools and misc tools

There are several tools that were created to help the tests builders in their task.

In each field of a test the user can write (& n) when n is a test number. This tells the system that to retrieve the value for this field it should refer to the same field in test n. This was done since many tests share values for some of their fields.

The test inspector is built on the top of the Interlisp-D inspector. The user can inspect and modify the various fields of a test. In addition he can call the inspector on indirect referenced tests.

There are functions which keep tracks of changes done to tests, and functions which stores modified tests.



The Random Generator is a very important tool for creating a test. It has many entries for different Lisp objects which he can generate randomly. These set of possible entries will grow constantly as test builders will need more types of random objects.

The random generator function gets as an argument an object type and a list of modifiers. The test builder can specify in the input field of a test a call like `(GenerateRandom 'LARGE-INTEGER)`, or `(GenerateRandom 'WINDOW)` or `(GenerateRandom '(LIST-OF-ITEMS WINDOW 50 100))` to get a random list of length between 50 and 100 of windows.

From the experiments done with the Testing system it was clear that random tests are an important part of any testing and can discover bugs that would be hard to find otherwise.

Another tool is the SYSTEMATIC input generator. Many times we want to test a function if it works right with all the possible combinations of values of its arguments, or to find out whether an library package works with all possible settings for its flags. For such cases the test builder can specify in the input field that he wants a systematic test, and supply the expressions that produces the ranges of values to combine.

## TEST EXAMPLES:

## EXAMPLE 1

```

Test number 18
PRETESTS :NIL
TESTCOMMENT :(* * Generates systematically all the pairs of all
 the "special" bignums
 (stored in TEST.BIGNUM-SPECIAL-NUMBERS in file
 TEST-ARITHMETIC-UTILS)
 apply IPLUS on them and the IDIFFERENCE
 and compare the results)
EVALAFTER :[LAMBDA (RES ARGS)
EVALBEFORE :(& 4)
TIMEOUT :NIL
TIMES :1
SUCCESSPREDICATE :[LAMBDA (RES ARGS)
 (if (EQP (IDIFFERENCE RES (CADR ARGS))
 (CAR ARGS))
 then (QUOTE SUCCESS)
 else (QUOTE FAILURE)]
INPUT : (SYSTEMATIC TEST.BIGNUM-SPECIAL-NUMBERS
 TEST.BIGNUM-SPECIAL-NUMBERS)
EVALEXPR :IPLUS
TESTID :18

```

## EXAMPLE 2

```

Test number 14
PRETESTS :((WEAK 10))
TESTCOMMENT :(* * Creates a random window, reshapes it to a
 sequence of random regions, and backwards to
 the original shape ; Checks if the screen
 bitmaps at the begining and end of operations
 are equal)
EVALAFTER :[LAMBDA (RES ARGS)
 (CLOSEW (CAR ARGS)]
EVALBEFORE :NIL
TIMEOUT :[LAMBDA (ARGS)
 1000]
TIMES :1
SUCCESSPREDICATE :[LAMBDA (RES ARGS)
 (if (TEST.COMPARE-BITMAPS RES (CADR ARGS))
 then (QUOTE SUCCESS)
 else (QUOTE FAILURE)]
INPUT :[LIST (TEST.GENERATE-RANDOM (QUOTE WINDOW))
 (BITMAPCOPY (SCREENBITMAP))
 (TEST.GENERATE-RANDOM (QUOTE (LIST-OF-ITEMS REGION
 10 50)]
EVALEXPR :[LAMBDA (WINDOW OLDScreen REGION-LIST)
 (PROG (OLD-WINDOW-REGION)
 (SETQ OLD-WINDOW-REGION (WINDOWPROP
 WINDOW
 (QUOTE REGION)))
 (for R in REGION-LIST
 do (SHAPEW WINDOW R))
 (for R in (APPEND (REVERSE REGION-LIST)
 (LIST OLD-WINDOW-REGION))
 do (SHAPEW WINDOW R))
 (RETURN (BITMAPCOPY (SCREENBITMAP)]

TESTID :14

```

## EXAMPLE 3

```

Test number 4
PRETESTS :((STRONG 11 23)
 (WEAK 7))
TESTCOMMENT :(* * Generates 1-30 BIGNUMS of the form 100...0,
 applies ITIMES on them,
 and checks that the result is of the form
 100....00 with the right number of ZER0s)
EVALAFTER :NIL
EVALBEFORE : (LOAD? (QUOTE TEST-ARITHMETIC-UTILS))
TIMEOUT : [LAMBDA (ARGS)
 (IPLUS 10000 (ITIMES (LENGTH ARGS)
 10000))]
TIMES :10
SUCCESSPREDICATE : [LAMBDA (RES ARGS)
 (PROG (SUM-OF-LENGTH UNPACKED-RESULT)
 [SETQ SUM-OF-LENGTH
 (for ARG in ARGS
 sum (SUB1 (LENGTH (UNPACK ARG)
 (SETQ UNPACKED-RESULT (UNPACK RES))
 (if
 [AND (EQP (ADD1 SUM-OF-LENGTH)
 (LENGTH UNPACKED-RESULT))
 (EQ (CAR UNPACKED-RESULT)
 (QUOTE 1))
 (for DIGIT
 in (CDR UNPACKED-RESULT)
 always (EQ DIGIT
 (QUOTE 0)
 then (RETURN (QUOTE SUCCESS))
 else (RETURN (QUOTE FAILURE]
 POSITIVE-POWEROF10-BIGNUM 1
 30))))
 (QUOTE 0)]
 (QUOTE 1))
 (RETURN (QUOTE FAILURE])
 (QUOTE (LIST-OF-ITEMS
 POSITIVE-POWEROF10-BIGNUM 1
 30))))
INPUT : (TEST.GENERATE-RANDOM (QUOTE (LIST-OF-ITEMS
 POSITIVE-POWEROF10-BIGNUM 1
 30)))
EVALEXPR :ITIMES
TESTID :4

```

## EXAMPLE 4

```

Test number 16
PRETESTS :NIL
TESTCOMMENT :(* * This test create systematically all
 combinations of arguments to FONTCREATE
 function,
 and tests whether the only error is
 "file not found")
EVALAFTER :[LAMBDA (RES ARGS)
 (if (NOT (TEST.ERRORP RES))
 then (APPLY (QUOTE SETFONTDESCRIPTOR)
 ARGS)]
EVALBEFORE :NIL
TIMEOUT :NIL
TIMES :1
SUCCESSPREDICATE :[LAMBDA (RES ARGS)
 (if (OR (NOT (TEST.ERRORP RES))
 (EQP (CADR RES)
 17))
 then (QUOTE SUCCESS)
 else (QUOTE FAILURE)]
INPUT : (SYSTEMATIC (QUOTE (GACHA NIL))
 (QUOTE (-1 12 NIL))
 [TEST.ALL-COMBINATIONS
 (QUOTE ((BOLD MEDIUM LIGHT)
 (ITALIC REGULAR)
 (REGULAR COMPRESSED EXPANDED))
 (QUOTE (0 90 NIL))
 (QUOTE (DISPLAY PRESS NIL))))
EVALEXPR :FONTCREATE
TESTID :16

```

## TEST APPRENTICE

This is the preliminary documentation for the first experimental version of the Test Apprentice. The purpose of this tool is to help with testing. It is eventually intended to generate and execute tests (it would be an AI application). In its current state it is just learning by watching what other testers do. But it is useful in this state because it can repeat exactly what other testers have done before.

The Test Apprentice is easy to use, you just type in the commands that you would use for doing the test the first time, with only an occasional extra command. To repeat a test, you only need to specify a special Test Apprentice function giving it the name of the test to re-perform.

The Test Apprentice groups test steps (commands) into tests, and groups tests into test suites. A test is a group of highly related test steps which are used to test a (small) portion of the system and which can be run independently from other tests (e.g. NS Filing directory enumeration). A test suite is a group of related tests which can be run together for convenience (e.g. NS Filing).

The Test Apprentice has the following limitations:

1. It can only record and repeat commands to the (standard) Lisp Executive, it does not record mouse operations or commands typed into other windows. It also cannot in general record input which is non-commands typed to the Lisp Executive (e.g. responses to the FILES? function's questions), but some commands record responses on the history list, so they are recorded (e.g. responses to the COMPILE function's questions). The rule of thumb to use here is if a REDO would re-perform those inputs, then the Test Apprentice has recorded them.
2. The number of test steps recorded for a particular test is limited to the size of the history list (usually 100 entries).

The following are the commands and functions available in the Test Apprentice:

|     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ST  | Start Test. A Lisp Executive command which starts the Test Apprentice recording the test steps of a test. An implicit ST is performed after an ET command or an ADD-TO-TEST-SUITE function. An ST can be performed at any time to restart recording of a test (and forget any recording in progress).                                                                                                                                                                                                                                                        |
| ET  | End Test. A Lisp Executive command which completes the recording of the test steps of a test. This must have the name of the test as its single parameter (e.g. "ET NSFiling-1"). Test names only have to be unique within a test suite.                                                                                                                                                                                                                                                                                                                     |
| ITS | Ignore Test Step. A Lisp Executive command which ignores test steps in the current test. Currently this will only ignore the previous step, but in the future it will ignore specified steps. This is useful for removing mistakes from a test or removing miscellaneous commands not directly related to the tests. If you need to delete anything other than the last test step, then you must edit the test suite manually or restart the test using ST (the former can only be done after an ET, while the latter can only be done before an ET).        |
| ITR | Ignore Test Result. A Lisp Executive command which ignores test results from test steps in the current test. Currently this will only ignore results from the previous step, but in the future it will ignore results from specified steps. This is useful for ignoring intermediate results. The Test Apprentice will check to see if the results from re-running tests are EQUAL to the results obtained when the test was recorded. For example, one test step may have returned a window as a result, then the next merely see if the type of the result |

is WINDOW. In this case two different instances of a window will not be EQUAL, so the result of the first step should be ignored. If this becomes a problem EQUALALL may be used in a future version of the Test Apprentice, but even this does not always work (e.g. it does not work on windows).

**ADD-TO-TEST-SUITE** An NLAMBDA function which adds a test to a test suite. Its one required argument is the name of the test suite. If the test suite did not exist before, then it is created. Test suites are just variables that contain a list in a specified format, they can be saved in files by the File Package command VARS (or UGLYVARS if data structures are used in test results or HORRIBLEVARS if there are circular pointers). This function must be called before any test steps are started. It sets up all future tests to be part of this test suite until changed by another ADD-TO-TEST-SUITE.

**EXECUTE-TEST-SUITE** A LAMBDA function which executes all tests in a test suite. Its one required argument is the test suite to execute. This returns a true (non-NIL) value iff all tests executed successfully. An error message is printed out for each test which fails.

**EXECUTE-TEST** A LAMBDA function which executes one specific test in a test suite. Its required arguments are the test suite and the name of the test in the test suite to execute. This returns a true (non-NIL) value iff the test executed successfully.

The following are recommendations on using the Test Apprentice effectively:

1. Don't use absolute command number references, use relative ones (e.g. use "(VALUEOF -2)" not "(VALUEOF 36)" if you are on command number 38). This is so the test will run correctly when it is re-run (if it were re-run and the VALUEOF was done on command number 67, then you would get some unexpected command's result instead of what you wanted).
2. Two different data structures (defined with DATATYPE, ARRAY, etc., such as windows) will never compare EQUAL, even if they contain the same values. This means these should never be used as (non-ignored) test step results. Use ways of looking at the contents of the data structures instead.

The following is an example of a normal session with the Test Apprentice for recording a test suite:

```

34_(ADD-TO-TEST-SUITE NSFiling)
NSFiling
35_ST
Start-of-test-block
36_<test 1 step 1>
<test 1 step 1 result>
37_<test 1 step 2>
<test 1 step 2 result>
...
40_ET NSFiling-1
End-of-test-block
41_ST
Start-of-test-block
42_<test 2 step 1>
<test 2 step 1 result>
43_<test 2 step 2>
<test 2 step 2 result>
...
65_ET NSFiling-2

```

```

End-of-test-block
66_(FILES?)
the variables: NSFiling...to be dumped.
want to say where the above go ? Yes
(variables)
NSFiling File name: NSFiling
NIL
66_(MAKEFILE '{Erinyes}<Test>Tests>OS>NSFiling)
{Erinyes}<Test>Tests>OS>NSFiling.;1

```

The following is an example of a normal session with the Test Apprentice for repeating a test suite:

```

19_(EXECUTE-TEST-SUITE NSFiling)
Executing NSFiling-1
Executing NSFiling-2
T

```



# THE INTERLISP-D TESTING SYSTEM - USER GUIDE

This document should be used as a guide for users of the testing system, and it assumes that the reading of "The InterlispD Testing System" document.

## LOADING THE TESTING SYSTEM

The testing system resides on the directory {Eris}<test>tools>. To load the testing system, LOAD the file TESTERLOADER. This file contains the correct loading sequence for the testing system files.

FILES:

TESTER.DCOM - Contains the main part of the tester program.

TESTERVARS - Contains most of the tester global variables.

TEST-REMOTE-EVAL.DCOM -Contains the functions for executing remote eval.

RANDOM-GENERATOR.DCOM - Contains functions and variables for random generation.

VARBROWSER.DCOM - A user package for manipulating the programs global variables.

THERMOMETER.DCOM - A user package for displaying the progress of a program in execution.

INTERLISPD-SYSTEM.CONCEPTSPACE - The file contains the global concept space of the system.

## EXECUTING TESTS

Usually execution of tests will be done through the Concept Space Browser. The basic function which is called by the browser is:

(TEST.PERFORM-TEST TEST TIMES LOCATION TRACE-FILE TRACE-MODE) [function]

Only the first argument is necessary. TEST must be of type TEST. TIMES is the number of times that the test will be performed and is defaulted to the value of the field TIMES of TEST. If this value is NIL it will be executed once. LOCATION can be either the atom "Local" or the atom "Remote" and is defaulted to the value of the global variable TEST.DEFAULT-LOCATION. TRACE-FILE is the name of the file on which the test execution process should be traced. The default name is the value of the global variable TEST.TRACE-FILE-NAME. TRACE-MODE can be the atom "On" or the atom "Off". It is defaulted to the value of the global variable TEST.DEFAULT-TRACE-MODE. When TRACE-MODE is "On" every testing step will be recorded on the trace file as soon as possible. The tracing will be done in a "careful" way - as soon as a new information is available (The input was generated, the result from the tested expression evaluation is returned, etc. ) , the trace file will be opened the information will be written and the file will be closed. If the trace mode is Off, the function will check at the end of each iteration of the function to see if the outcome was a failure and only in such a case it will write it down on the trace file.

There are two ways in which the PERFORM-TEST function can iterate. One way is if the TIMES argument is greater than one (or the times field of the test is greater than one). The other way is when the input expression is a list starts with the atom SYSTEMATIC. In such a case each of the elements of the CDR of the list will be evaluated. The tester expects them to produce sets (lists) which are the ranges of the arguments. It will then collect all the possible combinations of the elements in these finite ranges and perform the test on each of these combinations.

The function returns the name of the trace file.

(TEST.HARDCOPY-TRACE-FILE TRACE-FILE OUTPUT-FILE FAILURES-ONLY)

The default for TRACE-FILE is the value of TEST.TRACE-FILE-NAME. The default for OUTPUT-FILE is the value of TEST.DEFAULT-HARDCOPY-DEVICE (usually {LPT}). The trace file is written in a way that is hard for reading. This function prints the trace file in a "pretty" way. If FAILURES-ONLY is non-NIL only the trace of tests that failed will be printed out.

## Manipulating Concept Spaces

Concept spaces are stored in files, each concept space in its own file. Usually the name of the file can be XXX.CONCEPTSPACE where XXX is the name of the concept space. This name can be retrieved by calling

(TEST.CANONICAL-CONCEPT-SPACE-FILE-NAME CONCEPT-SPACE-NAME)

Returns XXX.CONCEPTSPACE where XXX is the value of CONCEPT-SPACE-NAME.

A concept space is of type CONCEPTSPACE which is a record with the fields CONCEPTSPACENAME, ROOTCONCEPT and CONCEPTLIST. ROOTCONCEPT is the name of the root concept. CONCEPTLIST is a list of concepts. A concept is an instance of the record CONCEPT which has the fields CONCEPTNAME, TESTS, SUBCONCEPTS and SUPERCONCEPTS. SUBCONCEPTS and SUPERCONCEPT are NAMES of concepts. TESTS is a list of tests ids. Usually the initial concept space will be created by the function

(TEST.CREATE-NEW-CONCEPT-SPACE CONCEPT-SPACE-NAME ROOT-CONCEPT-NAME)

which returns an instance of CONCEPTSPACE with one concept. The rest of the concept space will be most conveniently built using the Concept Space Browser.

The system maintains a global list of the concept spaces that are "known" to the system. It is convenient to work with concept spaces that appear in that list since this will enable the user to perform certain operations on the concept space using the background menu. The global list is stored in the global variable TEST.CONCEPT-SPACES. This variable can be manipulated directly or by calling the function

(TEST.ADD-CONCEPT-SPACE-TO-CONCEPT-SPACES CONCEPTSPACE)

If a concept space with the same name is already in the list, it will be removed and the new one will be added. The function

(TEST.GET-CONCEPT-SPACE NAME)

Returns the concept space with name NAME (if there is one in TEST.CONCEPT-SPACES).

Concept spaces that are on the TEST.CONCEPT-SPACES can be stored by calling the function

(TEST.STORE-CONCEPT-SPACE CONCEPTSPACE-NAME)

The function will prompt for a file name, suggesting the canonical name. If the concept space was loaded before through the TEST.LOAD-CONCEPT-SPACE function then the candidate file name will be the same as the one that it was loaded from (but with higher version). The user can also choose the subitem "Store Concept Space" from the background menu.

(TEST.LOAD-CONCEPT-SPACE CONCEPT-SPACE-FILE-NAME)

Loads the concept space from the specified file, adds it to TEST.CONCEPT-SPACES and keeps the file name in the property CONCEPTFILE of the concept space name.

## Building and manipulating tests

(TEST.CREATE-NEW-TEST )

[function]

Creates an instance of the record TEST with the default values which are obtained by calling the function TEST.GET-DEFAULT-FIELD-VALUE. The new instance is then added "officially" to the world by calling TEST.ADD-TEST . Then the test editor is called on the new test to allow the user insert values to its fields. It does NOT store the test in a file, and this should be done later by calling either TEST.STORE-TEST or TEST.STORE-CHANGED-TESTS. The test, after created is not assigned to any concept, and it is recommended to assign it as soon as possible to at list one concept.

(TEST.ADD-TEST TEST-RECORD)

[function]

Assigns a new ID to the test record instance by calling TEST.GET-AND-INCREASE-NEXT-TESTID and adds the test to the list of loaded tests.

( TEST.EDIT-TEST TEST)

[function]

TEST should be a test or a test number. If the global flag TEST.OBTAIN-LOCK-WHEN-EDIT is non NIL the function will try to obtain write lock on the test. If it fails to obtains such a lock it will exit without editing. The user can get the name of the locking users and automatically send release requests as described in the "accessing the database" section.

The test inspector will be called and the test will be marked as changed.

(TEST.GET-TEST TEST-NUMBER)

This function is the user interface to the database management system. The user calls this function and gets the test with the TESTID TEST-NUMBER. Actually the system will search for the test in the list of the loaded tests. If it will not be found a "test fault" will occur and the system will look for the file for this test and load the test from there. If by adding the test to the list of loaded tests an "overflow" occurs (the number of tests is more then the maximum allowed), the last test on the list (the least recently used will be deleted from the list.

(TEST.GET-FIELD-VALUE FIELD DATUM)

Gets the "Actual value" of a test field. If the field value is indirect reference to other test, the field value will be retrieved from there (using TEST.GET-FIELD-VALUE thus enable chaining), otherwise it will return the field value of DATUM.

( TEST.GET-DEFAULT-FIELD-VALUE FIELD-NAME)

FIELD-NAME is a TEST field name. The function will return the global default value for this field.

The following functions can be useful when building a test for use within the actual test fields:

(TEST.TEST-SINGLE-TIME TEST-NUMBER)

There are tests which have the TIMES field greater than one. Such tests will be executed more than one time. There are also tests which have as their INPUT field a list that starts with the atom SYSTEMATIC. Such tests will be applied on all the combinations of the elements of sets in their INPUT field. Such tests can run for hours only to discover that something in the test itself was not correct, and the trace file is full of garbage. To avoid such a case it is recommended to create for such tests a "Testing test" which will perform the tested test once and will only check that the outcome is meaningful (i.e. The result is either "success" or "failure" etc). Such a test can be built easily by having as its Expression field a call to this function with the tested test as TEST-NUMBER. It may be also useful to add a WEAK or STRONG link in the tested test PRETESTS field.

(TEST.ERRORP EXPR) [function]

If an error occurred during the evaluation of the evaluated expression, the returned result will be a list of two elements, the first is the atom ERROR!, and the second is the error number. This is a simple predicate that checks if an expression is of the form described above. It is useful for building the success predicate. For example, a test may have as the EVAEXPR the function ADD1, as its INPUT field some non-numeric atom, and as its success predicate, the expression (LAMBDA (RES ARGS) (IF (AND (TEST.ERRORP RES)(EQP (CADR RES) 10)) THEN 'SUCCESS ELSE 'FAILURE) . This test tests whether the function ADD1 breaks with non-numeric arg error.

(TEST.ALL-COMBINATIONS SET-OF-SETS) [function]

Produces the Cartesian product of the sets in the list SET-OF-SETS . (TEST.ALL-COMBINATIONS '((a b)(1 2 3))) will return the list ((a 1)(a 2)(a 3)(b 1)(b 2)(b 3)). This function is used by the tester when it encounters an INPUT field that starts with the atom SYSTEMATIC. It is useful in any case that the user wants to build tests that try all the combinations of possible values of a function arguments, or to have all the possible settings for flags and global variables for some subsystem or library package.

(TEST.LOCAL-EVAL-FORM FORM) [function]

Evaluates the form FORM using ERRORSET, thus the evaluation will not break even if an error condition occurs. If an error did not occur the function will return the result of the evaluation of form. If an error condition was entered, the function will return a list with the first element ERROR!, and the second element the number of the error (as described in Interlisp-D manual).

(TEST.PERFORM-TIMED-EVALUATION FORM TIMEOUT.ms) [function]

Evaluates FORM using TEST.LOCAL-EVAL-FORM (thus it will not break). If the evaluation will take time which is longer then the value of TIMEOUT.ms, the function will return the error expression (ERROR! TIMEEXPIRED). This function is used by the test driver if the TIMEOUT field of a test is non NIL, thus an evaluation that will take more time that the designated time will be considered as returning with error condition. The function create a seperate process to perform the evaluation, and set a timer for the designated time (plus some time for overhead). The user should note that since the Interlisp-D process scheduling algorithms are non-preemptive, the function is not guarantied to halt. An infinite loop may not release the CPU and then only keyboard interrupt will work.

(TEST.GET-NEXT-AVAILABLE-TESTID ) [function]

The test ids should be unique. That's why the system maintains a file which holds the next available test id. This function access this file and returns the id.

(TEST.GET-AND-INCREASE-NEXT-TESTID ) [function]

This functions returns the next available id as the one above, but also increase this number on the file. this function is called by the function TEST.ADD-TEST which adds a test "officially" to the world.

## The Random Generator

The random generator resides on the file RANDOM-GENERATOR.DCOM. The main function is

(TEST.GENERATE-RANDOM OBJECT-SPECIFICATION) [function]

This function is planned to be constantly expanded by the tests builders according to their needs. The OBJECT-SPECIFICATION can be an atom, which should be one of the objects known by the random generator. It can also be a list where the first element is an atom which is one of the known objects, and the rest of the list are modifiers according to the object type. The current list of known objects is : (INTEGER, SPECIAL-INTEGER, BOUND-INTEGER, LARGE-INTEGER, SMALL-INTEGER, BIGNUM, POSITIVE-BIGNUM, SPECIAL-BIGNUM, POSITIVE-POWEROF10-BIGNUM, WINDOW, REGION, SHORT-SIMPLE-LIST, SHORT-SIMPLE-NON-NULL-LIST, SHORT-LIST, LIST-OF-CHARACTERS, CHARACTER, LIST-OF-ITEMS and some more.

Some of the objects have modifiers, like short list which can have a maximum depth as a modifier. A very important object is LIST-OF-ITEMS which can have as its modifier another object specification, thus enable recursive use of the function. Thus you can write (TEST.GENERATE-RANDOOM '(LIST-OF-ITEMS REGION 100 200)) which will produced betwween 100 and 200 random regions.

Database access

(TEST.GET-LOCKING-USERS TEST-LIST) [function]

TEST-LIST is a list of test numbers or the atom DATABASE. Will return the list of all the users that kas locks to tests in TEST-LIST together with the tests in TEST-LIST that they are locking. If TEST-LIST is the atom DATABASE, the function will return the names of all the users that have locks to any test.

(TEST.OBTAIN-DATABASE-WRITE-LOCK TEST-LIST) [function]

TEST-LIST is as above. The function tries to obtain locks on the list of tests in TEST-LIST. If TEST-LIST is the atom DATABASE, it will try to obtain lock on the whole data base. A lock on a test can be obtained if there is no other user locking the test. A lock to the whole data-base can be obtained if there is no user that locks any test. The function returns the list of all tests that it was able to lock.

(TEST.RELEASE-DATABASE-WRITE-LOCK TEST-LIST) [function]

As above, but releases the locks to the tests in TEST-LIST that are locked by the user. Returns all the tests that it succeeded to release.

(TEST.SEND-RELEASE-REQUESTS TEST-LIST) [function]

TEST-LIST is as above. Sends automatic messages to all the users with locks to the tests in TEST-LIST to release their locks.

(TEST.MARK-AS-CHANGED TEST-NUMBER) [function]

The number of the tests that are being modified using the programs editor are added to the global list TEST.LIST-OF-MODIFIED-TESTS. This can be done by calling this function.

(TEST.UNMARK-AS-CHANGED TEST-NUMBER) [function]

As above, but remove the test from the list.

(TEST.STORE-CHANGED-TESTS) [function]

Stores all the tests in the list of modified tests.

(TEST.STORE-TEST TEST-NUMBER) [function]

Stores the test TEST-NUMBER in the file with the name returned by the function TEST.TEST-NUMBER-TO-FILE-NAME, and removes it from the list of changed tests.

(TEST.TEST-NUMBER-TO-FILE-NAME TEST-NUMBER) [function]

Returns a file name on which the test TEST-NUMBER is stored. The directory is the value of the global variable TEST.TEST-DATA-BASE-DIRECTORY. The the root name for test number 45 will be TEST00045.

## The Concept Space Browser

To browse a concept space you can either select the submenu "Browse Concept Space", or by calling the function

(TEST.DISPLAY-CONCEPT-SPACE-BROWSER CONCEPT-SPACE REGION/POSITION DEPTH INCLUDE-TESTS) [function]

CONCEPT-SPACE must be of type CONCEPTSPACE. REGION/POSITION can be either a region or a position for the browser window. DEPTH is the depth of the lattice that will be displayed. If INCLUDE-TESTS is non NIL, the tests will be included as part of the displayed graph. Only the first argument is necessary.

All the operations on concepts are done in PREFIX form - first you select the operation and then the argument (like all Lisp operations). Copy selection from a node will push the list of test numbers belong to this node into the current tty stream. The operations that are available using the concept space browser are:

**Copy subtree** : Allow you to copy a subtree from one displayed concept space to another one. Will prompt for selection of the new parent node and the root of the subtree.

**Add Concept** : Prompts for the parent of the new concept and for the name of the new concept.

**Delete concept** : Deletes the concept selected, and all its children which have the deleted concept as they only parent (and so on recursively).

**Add Link** : prompts for the superconcept and the subconcept and creates a link between them.

**Delete Link** : prompts for the superconcept and the subconcept and deletes the link between them.

**Add test** : Adds a test to a concept. Prompts for selection of the concept and for a test number. The test number can be a list of numbers, thus you can copy select tests from any node on any browser window.

**Delete test** : Will ask you to select a node and will add a menu with all the tests of the node so that you can select those you want to delete.

**Edit test** : Will ask you for selection of a node and will pop up a menu of all the tests in the selected concept. Will apply the test editor on the selected test.

**Display - Display tests on/off** : will switch the mode of display. You can either display the graph with the tests as part of it or only with the concepts.

**Display - Browse subtree** : Will ask you to select a concept and will apply the concept space browser on the subgraph for which the selected node is the root.

**Display - Change depth** : Will pop up a menu of integers. You can select the depth of the graph being displayed.

**Display - Update** : Recomputes the graph of the concept space and redisplay it.

**Execute tests** : Will execute all of the tests or part of the tests of the selected concept. Will pop up a menu to set the execution modes.

**Hardcopy tests** ; Sends a pretty printed hardcopy of all the tests (or the selected tests) of the selected concept.

**data base - obtain lock** : Tries to obtain lock on all the tests of the selected concept.

**data-base - Release lock** : Releases all the locks that the user has to tests belongs to the selected concept



## Test Utilities

The following are a useful set of utilities for the testing group. They all reside in the TestUtils file on the Tools subdirectory of the standard test directory.

**PRINT-TEST-ARS**      A lambda function which prints out the AR number and subject fields from an AR query. This function has a window and a file as arguments. The window is the main window of an AR query form (labelled "AR Query Window"). The file is the file to print output on (NIL goes to standard output). The query must already have been performed to select those items which are to be printed and the query should have been sorted by status (as the first sort item in the sort list). In order for this to work correctly, you must scroll the "AR Query Browser" window through all the items before using this function. As an example of use, say that you had previously specified a query list of "(System: IS Operating% System) (Subsystem: IS Generic% File% Operations)" and a sort list of "Status:" in an AR query window (and performed the query and scrolled through all the items), then you could point the cursor to the window labelled "AR Query Window" and do "(PRINT-TEST-ARS (WHICHW) '{DSK}<LISPFILES>TEMP)". This would print a list of the ARs in the file {DSK}<LISPFILES>TEMP.