

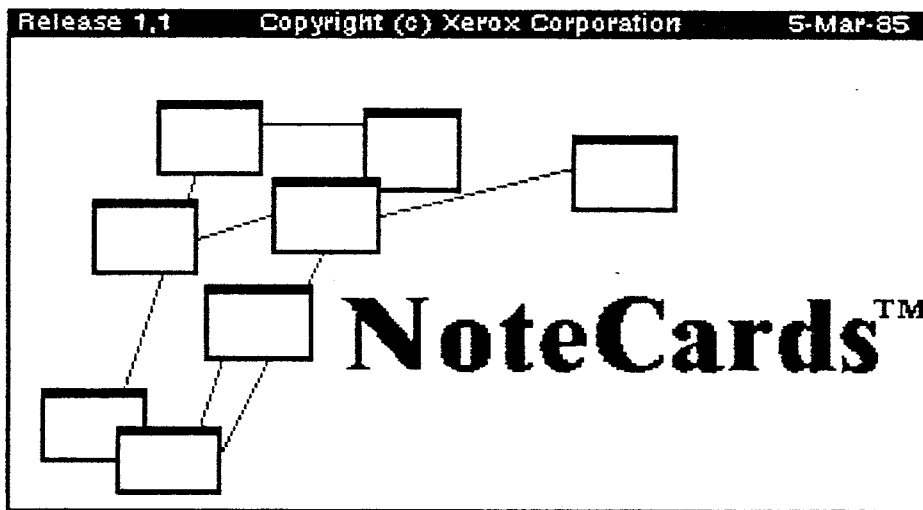
1.2

NoteCards™ Release ~~1.1~~ Reference Manual

Xerox Special Information Systems

~~5 March 1985~~

3 APRIL 1986



Copyright (c) 1985 Xerox Corporation

All rights reserved.

This publication may not be reproduced or transmitted in any form by any means, electronic, microfilm, xerography, or otherwise, or incorporated into any information retrieval system, without the written permission of Xerox Corporation.

Preface

The Release 1.2k Reference Manual is an extended version of its predecessor, the Release 1.1 Reference Manual. Due to the frequency of NoteCards releases, the changes to the original manual are presented as 1.2k release notes (Section 13), and are not integrated into the text of the 1.1 manual. The table of contents and the index have been modified to reflect the additions, so if a feature appears to be described in both places, please refer to the 1.2k release notes (Section 13) as well as to earlier sections of the manual.

Most of the figures and examples used in the NoteCards Release 1.2k Reference Manual are based on a NoteFile called *Demo.NoteFile* which is provided with the standard NoteCards release package. Use of this NoteFile while reading sections of the manual may prove to be a significant aid to understanding the contents of this document. For example, most of the sample NoteCard types shown in Section 2 are in the Demo.NoteFile filebox *Examples of NoteCard Types* filed in the NoteFile's *Table of Contents*. Likewise, the filebox *NoteCards Functionality* contains explanations of several important features discussed in this manual; these explanations will step you through their use.

NoteCards is a research prototype that is part of an ongoing project in the Intelligent Systems Laboratory at the Xerox Palo Alto Research Center. This project investigates "idea structuring" tasks. Because NoteCards is a vehicle for current research and is still undergoing development, you may encounter occasional bugs or be frustrated by seeming deficiencies. This manual has been written with these considerations in mind. User feedback has helped drive the development of NoteCards through many iterations, so your comments on the documentation and the demo NoteFile as well as on NoteCards itself are encouraged.

Acknowledgments

NoteCards was designed and implemented at the Xerox Palo Alto Research Center (PARC) by Frank Halasz, Thomas Moran, and Randy Trigg. Other contributors to NoteCards include Ken Allen, Cathy Marshall, Melissa Monty, Kurt VanLehn, and Terri Wanke. NoteCards was influenced by the AnnoLand system, developed by Richard Burton and John Seely Brown at PARC. NoteCards is built on several Interlisp-D packages including TEdit (John Sybalsky), Sketch (Richard Burton), and Grapher (Ron Kaplan, Kurt VanLehn, *et al.*).

This Reference Manual was written by Cathy Marshall, based on partial documentation by Richard Burton, Frank Halasz, John Sybalsky, Randy Trigg, and Terri Wanke. The NoteCards release package was assembled at the Xerox Special Information System's Vista Lab by Ken Feuerman, Cathy Marshall, and John Todd.

Correspondence

Comments on or corrections to this manual, questions about the distribution of NoteCards, and bug reports for the NoteCards system should be sent to:

NoteCards Support
MS 1120
Xerox Special Information Systems
250 N. Halstead St.
Pasadena, CA 91107

Correspondence concerning research issues should be addressed to:

Thomas Moran
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

Contents

0.	Using This Manual	0
1.	Overview & Concepts	1
2.	NoteCards	4
3.	FileBoxes	16
4.	Links	20
5.	Main Menu	25
6.	Screen Management	30
7.	Text Editor	32
8.	Graph Editor	40
9.	Sketch Editor	42
10.	Bitmap Editor	58
11.	Programmer's Interface to NoteCards	61
12.	Recovering from System Bugs	64
13.	NoteCards Release 1.2k Description	70
<hr/>		
	Appendix A: Programmer's Interface Functions	84
	Appendix B: NoteCards Types Mechanism	100
	Appendix C: NoteFile Concepts	114
	Appendix D: Inspecting and Repairing NoteFiles	117
<hr/>		
	Index	128

0. Using This Manual

This manual was originally written as the reference manual for Release 1.1 of the NoteCards system. Since the system is changing rapidly and is undergoing continuous development, the Release 1.2 manual has retained the structure it had for the previous release, and is supplemented by Section 13, a description of NoteCards Release 1.2k. All of the appendices have been updated with material pertaining to Release 1.2k. Therefore, if material in the Release 1.1 portion of the manual (Sections 1 through 12) appears not to pertain to the current functionality of the system, please refer to Section 13 to find a more up-to-date account of various features.

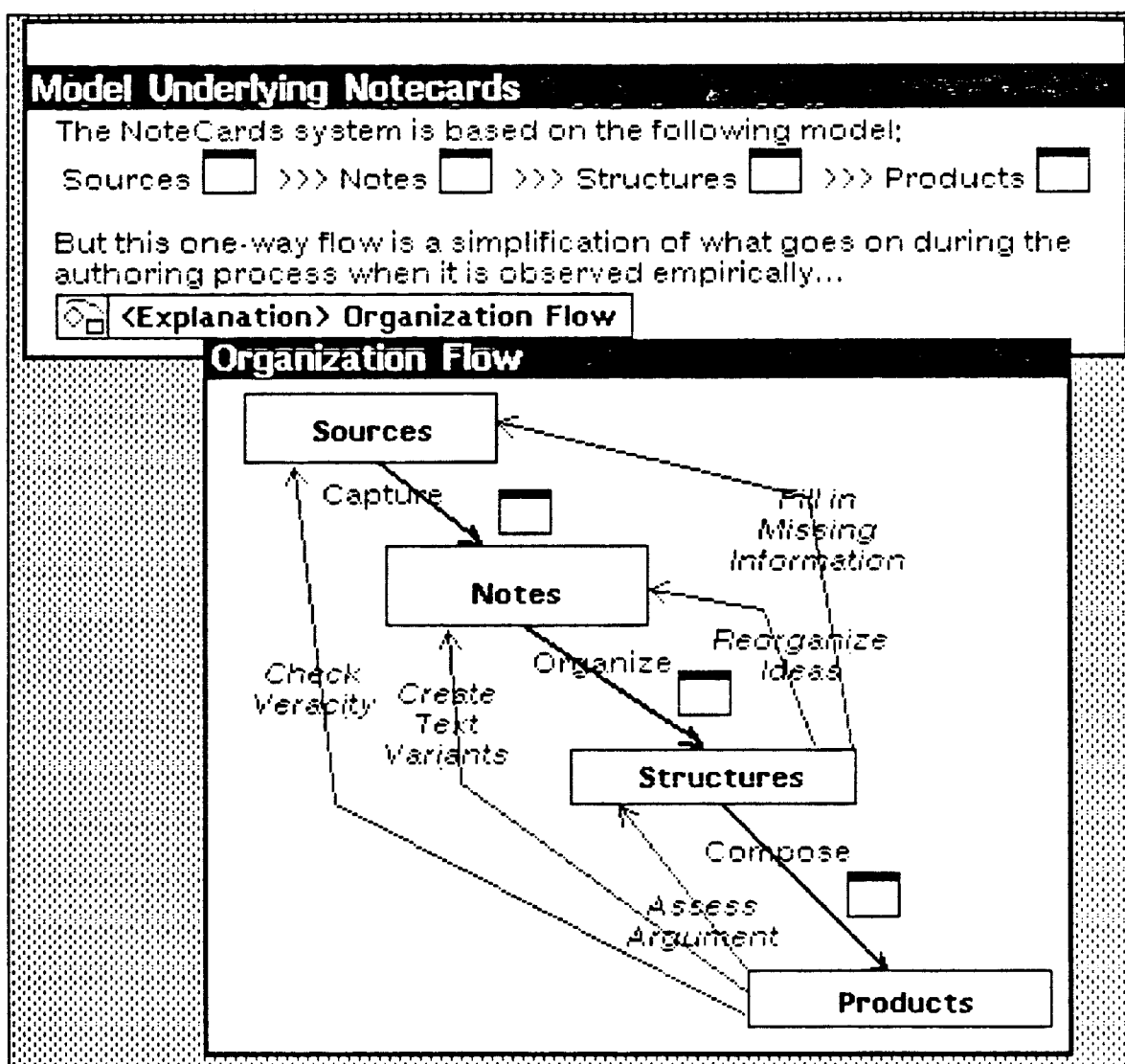
Changes of particular note include modifications to the browser card, originally described in Section 2.1; Section 13.3 has instructions on how to use this extended facility. Many of the NoteFile Ops described in Section 5 have also undergone major changes; they are documented in Section 13.1 (including new and changed maintenance and recovery operations). The NoteCards Parameters Menu, also described in Section 5, has been updated to include the parameters documented in Section 13.2. Some general aspects of the user interface have changed; in particular, the stylesheets discussed in Section 13.2 have replaced sequences of menus. For example, changing link display mode (Section 4.2) used to require a sequence of menu choices starting from the Pointer Ops menu. Link icon display mode is now controlled by a stylesheet, also shown in Section 13.2 (Refer to Figure 13.2-3).

Sometimes menu items will have changed in name or menu level, but not in functionality. For example, the Collect Children option (see Section 3.2) on the FileBox menu is now called Put Cards Here. The functionality associated with this menu item remains unchanged. An example of menu level change is the elevation of Show Links to a separate item on a card's left button title bar menu. In Release 1.1, this same function was available as a submenu option of the title bar menu item Show/Edit Properties (see Section 2.3).

If you are an experienced Release 1.1 NoteCards user, it is suggested that you scan Section 13, just to learn what new functionality is available. If you are a new user, much of the Release 1.1 portions of the manual are still valid; however, it may be necessary to read Section 13 to understand the current state of the system.

1. Overview & Concepts

NoteCards is a system designed to aid in the collection, structuring, and analysis of textual and graphical information. The basic objects in the NoteCards system are **NoteCards** (or simply cards). Each card contains a small, idea-sized unit of text or graphics. Individual cards can be linked together to form networks that reflect the interconnections among the information (ideas) contained in each card. These networks among cards rely on the notion of typed links. **Links** are identified by link icons, objects located in the text or graphics of one card (the originating card) that reference some other card (the destination card). The user can retrieve and display the destination card of any link by using the mouse to select the iconic representation of that link in the originating card. The user can also display a listing of all of the links that are directed into and out of a card. The screen image below shows two cards that discuss the model underlying the system.

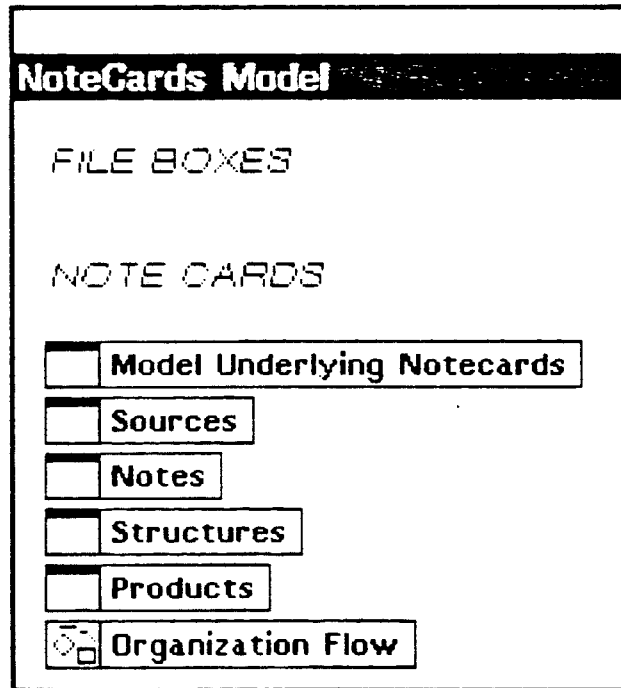


The type of each link specifies the nature of the relation between the originating and destination cards. For example, in the screen image above, the card titled "Model Underlying Notecards" has links to other cards that explain various components of the model. One of those links goes to the card

"Organization Flow", which is displayed beneath it. The link icon in "Model Underlying Notecards" has been expanded to show its type, Explanation, and its destination, "Organization Flow".

Cards are stored in NoteFiles. The user may have several NoteFiles, corresponding, for example, to various topics or subject domains. However, only one NoteFile can be active at a given time. More importantly, cards in one NoteFile cannot be linked to cards in another NoteFile. Therefore, a NoteFile is a self-contained collection of interconnected cards.

It is easy for users to get lost in a large NoteFile of interconnected cards. For this reason, the NoteCards system provides two organizations: the network of interconnections among cards and the FileBox hierarchy. A FileBox is an object that contains (links to) cards and/or other FileBoxes. Cards can be filed in one or more FileBoxes. Every FileBox (except the "root" FileBox) is contained in at least one other FileBox. Whereas cards may be linked together to form an arbitrary network, the set of FileBoxes forms a strict hierarchy (i.e., the set of FileBoxes contained in FileBoxes is restricted to form a directed lattice structure). FileBoxes are meant to hold all cards relating to some given topic. A FileBox typically contains both subFileBoxes, containing any cards relevant to the subtopics of the main topic, and cards containing information relevant to the main topic. For example, the screen image below shows the FileBox containing both cards shown in the previous screen image in addition to other cards describing aspects of the NoteCards model



The FileBox structure provides a way of keeping track of sets of cards on a common topic. In contrast, the links between individual cards allow the user to represent the interconnections between various ideas or pieces of information, independent of any categorization into topic areas.

The NoteCards system provides a number of tools to help users collect, manipulate, and organize the cards in a NoteFile. The **Browser mechanism** displays a graphical representation of a given set of

cards and the links between them. The **Search mechanism** builds a list (link icons) of all cards in the NoteFile that have a given word or character string in their descriptive title. The system provides a mechanism that helps track the source of the information in any card. This **Source mechanism** allows users to specify for each card a link to the source of the information in that card.

The system also maintains a special **property list** containing information about the author and time of every change made to the card. In addition to this update information, users can add arbitrary attribute-value pairs. For example, a user might want to attach a **Certainty Value property** (and value) to every card. The value of this property would indicate the degree to which the information contained on the card is believed to be true. At the present time, the property list of a card can only be inspected and changed by the user. However, future plans call for mechanisms that automatically inspect and manipulate these property lists, with the goal of making machine-generated inferences about the information contained in a card.

The **Link Index mechanism** builds a sorted list of cards linked by a specified link type or set of link types throughout an entire NoteFile. This mechanism allows users to examine a relationship in a NoteFile by looking at all instances of this relationship. The list generated by this process may be useful in clarifying the user-defined relationship taxonomy. Finally, the **Documentation mechanism** provides a tool to pull together the textual information stored in a specified group of cards into one card.

It should also be noted that the NoteCards system includes tools for manipulating whole NoteFiles. For example, NoteFiles can be backed up onto floppy diskettes and damaged NoteFiles can be repaired.

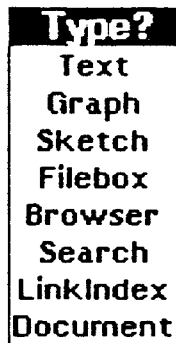
Sections 2 through 4 of this manual describe the objects in the NoteCards system, i.e., NoteCards, FileBoxes, and Links. These sections include descriptions of the Browser, Documentation, and Search mechanisms. Sections 5 and 6 explain how to manipulate these objects within the NoteCards environment. Sections 7 through 10 give operating instructions for the text, sketch, graph, and bitmap editors used to manipulate the text and graphics contained in NoteCards. Section 11 discusses the Programmer's Interface and how card types may be created through this interface. Section 12 outlines strategies for coping with system bugs. Appendices A and B provide a more detailed look at the Programmer's Interface and the NoteCards Types Mechanism.

2. NoteCards

Three types of cards, **Text**, **Sketch**, and **Graph**, can be used to store various types of information in the NoteFile. Text, graphics and combinations of the two may be entered into a card with the Text and Sketch editors. In addition, the Graph editor is used to construct a layout of user-defined, labelled nodes.

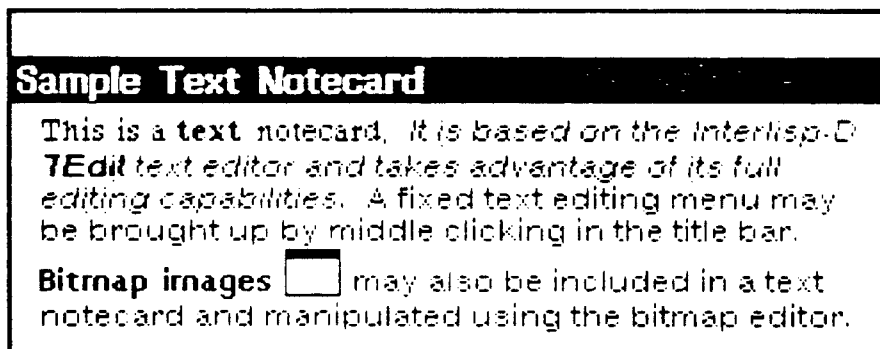
2.1. NoteCard Types

Cards are created by selecting **Create** from the **Main Menu** with the *middle* mouse button (use both buttons for *middle* if using a two button mouse) and choosing a card type from the Type? menu.

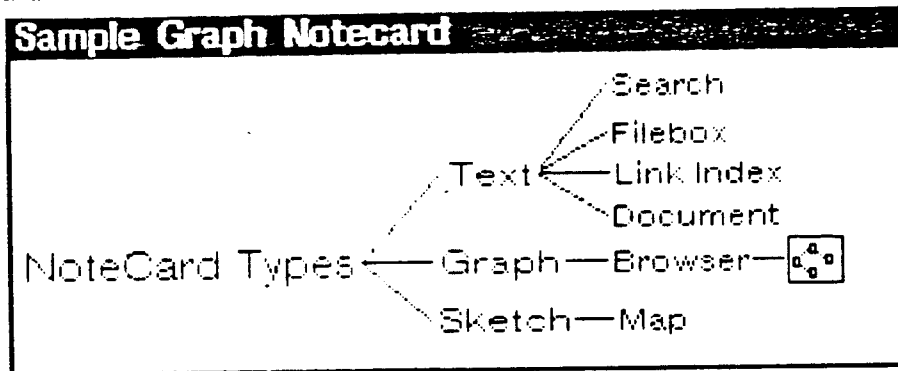


Selecting **Create** with the *left* mouse button immediately creates a specific type of card. This specific type has been previously designated as the default type through the **Change Parameters** command from the **Main Menu**. (See Section 5. Main Menu: *Change Parameters* and *Create*). Cards are viewed through windows manipulated by a window menu (see Section 5. Main Menu: *Create* and Section 6.1. Window Menu). Approximately 20 cards and/or boxes may be displayed on the screen at one time with the windows overlapping if desired. After making a selection, a border of a card will pop up next to the cursor. This border can be moved to any location on the screen by moving the cursor. Plant the border by depressing any mouse button, and a card then replaces the border. The system-defined card types are:

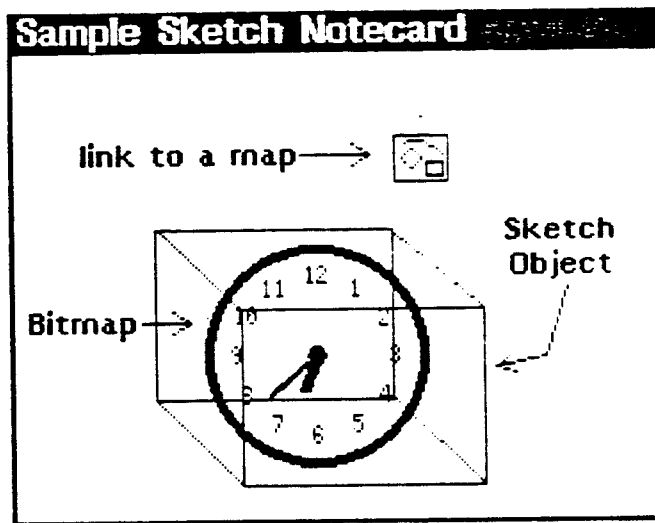
Text — a card for entering text. (See Section 7. Text Editor for editing instructions.) An example of a text card is shown below.



Graph — a card for creating and editing node links and graph structures. An example of a graph card is shown below. (See Section 8. Graph Editor for editing instructions.)



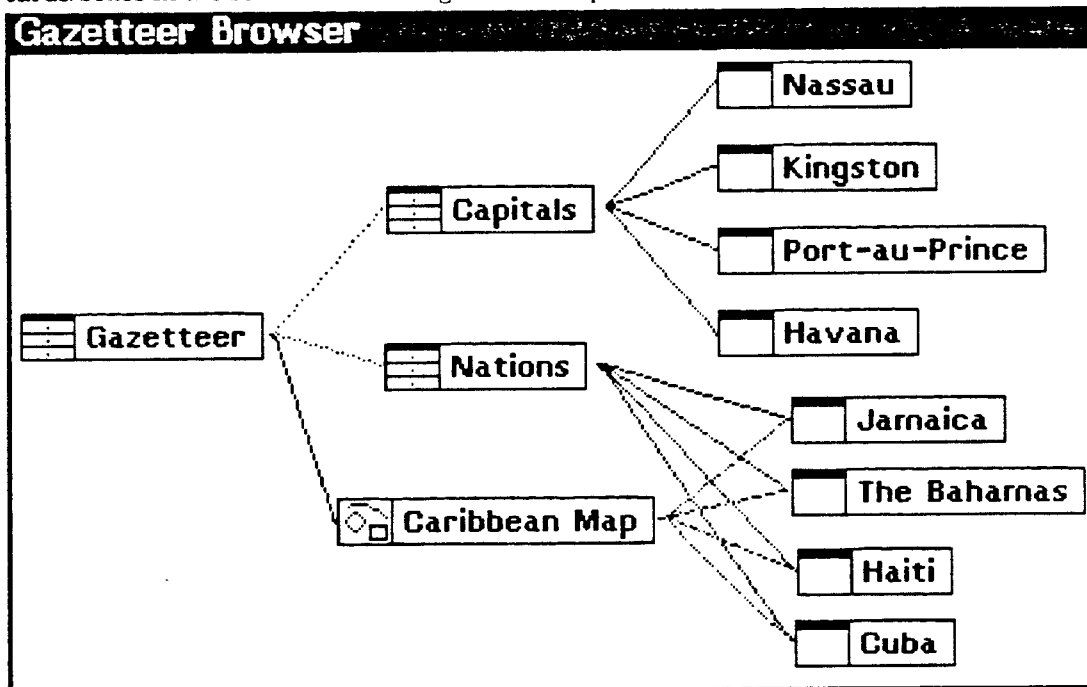
Sketch — a card for creating and editing sketches, e.g. line drawings, that can optionally include maps and bitmaps. Text may also be entered into this card. *Note:* creating a map requires substantial machine time. (See Section 9. Sketch Editor for editing instructions.) An example of a sketch card with text, a link to another card and an embedded bitmap is shown below.



FileBox — a card containing a list of link icons representing sub-FileBoxes (FileBoxes nested within other FileBoxes) and NoteCards. The labels *FILEBOXES* and *NOTECARDS* help differentiate what kinds of cards are filed in the FileBox. FileBoxes are used to support hierarchical organization of information according to conceptual groupings. Section 3. FileBoxes provides an explanation of how FileBoxes may be used for structuring information in the system.

Browser — a card that presents a view of cards/boxes and the links between cards/boxes in a specified portion of the current NoteFile. Given a starting card or box and a set of link types, Browser will create a graph structure showing all NoteCards and FileBoxes linked to and/or from the initial card/box by recursively following all links of the types specified.

This graph structure displays both the specified set of cards/boxes, and the links between the cards/boxes in the set. The following is an example of a Browser card:



Before selecting Browser, the starting card/box or a link icon referring to the card/box must be visible on the screen to allow selection with the mouse. After creating a blank browser card, a prompt window will be displayed above the card asking the user to designate a starting card/box. The user may select the initial card/box or abort the browser by selecting **Cancel**. In the above example, the box chosen was **Gazetteer**.

After designating the starting card/box, a list of Link Types will be displayed from which the user selects the types of links the browser should follow. Both links to cards/boxes and links from cards/boxes may be selected for tracking. The user may select the types and **Done** or abort the browse by selecting **Cancel**. In the above example, the link types chosen were **FiledCard**, **SubBox**, and **Comment**. Once the link types have been determined, the browser compiles the set of cards/boxes and displays it in the Browser card.

The links can be drawn in one of two ways, depending on a parameter value set in the **Change Parameters** command from the **Main Menu**. (See Section 5. **Main Menu: Change Parameters: LinkDashingInBrowsers**.) The first method shows the links as solid lines; the second method uses dashed lines. With the second method, each link included in the Browser is assigned a different style of dashing; therefore, dashing for

links after the sixth type will be of the same style. The Browser card shown above was created with the LinkDashingInBrowsers parameter set to Yes.

A legend is attached to the upper right-hand border of every Browser. This legend displays each type of link present, and, if the links are dashed, the legend displays the particular pattern of dashing. For the Browser card shown above, the legend appears as follows:

Links	
FiledCard	_____
SubBox
Comment	-----

The characteristics of browser cards and graph cards are identical except for the single-item menu that is displayed by pointing to the title bar with the cursor and depressing the middle mouse button. (See Section 8.1. Graph Editing Menu.) The command in this menu follows:

Recompute Browser — modifies the browser card to reflect the current NoteFile starting from the same card or box and following the same link types as originally specified.

Warning: nodes and links added or deleted through the **Graph Editing Menu** commands **Add Node**, **Add Link**, or **Delete Link** will be removed when **Recompute Browser** is executed. (See Section 8.1. Graph Editing Menu.)

Search — searches through all boxes and cards looking for titles containing a specified string of characters. A new card is created containing links to these cards. This search card also contains the day and time the list was compiled. Search is case sensitive so the string must be typed exactly as it appears in the title. For example, the Search card below was created by answering the prompt for a search string with Creat.

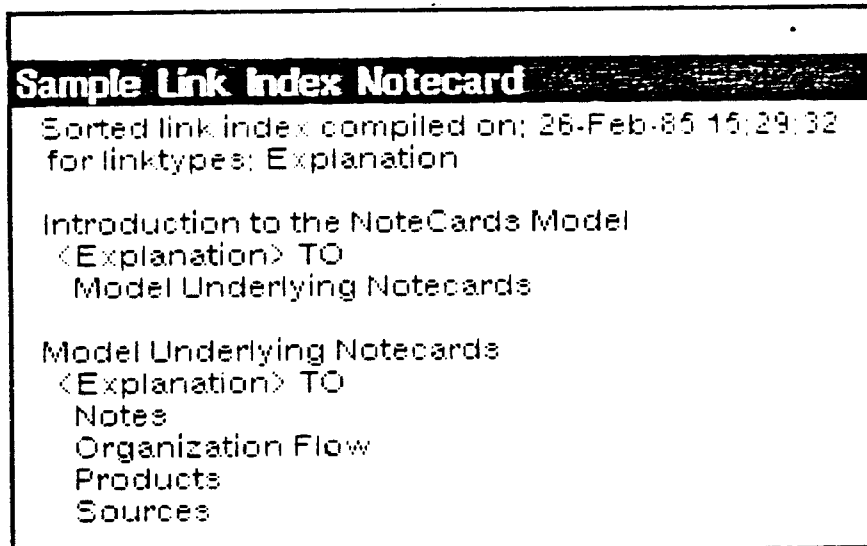
Cards with "Creat" in title

List compiled on: 1-Mar-85
13:34:31

- Creating Notecards
- Creating a Document Notecard
- Creating a Browser Notecard
- Creating a Text Notecard
- Creating a Link Index Notecard
- Creating a FileBox
- Creating Links
- Creating a Search Notecard

Note: this list is *not* updated as new cards are entered into the NoteFile. The date and time the list was compiled can be used to help determine if the list in the card is out-of-date.

LinkIndex — creates a card that contains a sorted list of all of the cards in the NoteFile connected by the specified link type(s). A Link Index card helps clarify the link-based relationship taxonomy which has been created within a NoteFile. The following is an example of a Link Index card.



After creating a link index card, a list of Link Types will be displayed from which the user selects the link type with which the index will be built. Both links to cards/boxes and links from cards/boxes may be selected for tracking. The user may select the type(s) and **Done**. In the example above, Explanation was selected, followed by **Done**.

Once the link type have been determined, a prompt window is displayed above the card asking the user whether or not back links should be built to the cards or boxes connected by the specified links. Back links are useful if the link index is being used to collect cards for later modification of link types. The example shown above was compiled with the back links option off. After the prompt is answered, link index compiles the set of cards/boxes linked by the chosen link types and displays the list in the Link Index card, sorted alphabetically. This new card also contains the day and time the list was compiled.

Note: this list is *not* updated as new cards are entered into the NoteFile. The date and time the list was compiled can be used to help determine if the list in the card is out-of-date.

Document — creates a card that contains textual information collected from all descendant cards and/or boxes from a specified FileBox or NoteCard. Once Document is selected, the user is prompted for a starting box or card. After this selection, a sub-menu of Make Document parameters is displayed.

MakeDocument Parameters	
HeadingsFromFileboxes	NumberedBold
TitlesFromNoteCards	Bold
BuildBackLinks	ToCards
CopyEmbeddedLinks	ALL
ExpandEmbeddedLinks	(Capital)
--DONE--	--CANCEL--

These parameters are set to default values displayed to the right of each parameter. It is possible to change a value by placing the cursor over the parameter and depressing the *left* mouse button. This pops up a menu of values. Select a value from this menu. The parameters and their choice of values are described below:

HeadingsFromFileBoxes — creates section headings and subheadings for the document according to the option selected.

NumberedBold — includes numbered headings, in bold, of FileBox descendants of the originating FileBox. Each heading is numbered in standard section heading form. For example, "1.3" would be assigned to the third sub-filebox of the first sub-filebox of the originating FileBox.

UnnumberedBold — includes unnumbered headings, in bold, of FileBox descendants of the originating FileBox.

NONE — excludes headings of FileBoxes including the originating FileBox.

TitlesFromNoteCards — enables the document card to be generated with paragraph-level labeling according to the option selected.

Bold — includes titles, in bold, of all card descendants of the originating card.

NotBold — includes titles, in regular type, of all card descendants of the originating card.

NONE — excludes all titles of cards including the originating card (if starting from a NoteCard).

BuildBacklinks — builds back links according to the option selected. Back links to cards allow the user to easily retrieve a card for text revision; back links to boxes facilitate structural revision of a document. In the sample MakeDocument Parameters menu shown above, the *BuildBacklinks* parameter is set to *ToCards* to facilitate the text editing process. When a final version of the document card has been generated, the user may want to set this parameter to *NONE* to omit the back links from a hardcopy.

ToCardsBoxes — builds links to each NoteCard and FileBox descendant of the originating card or box. The link icon, displayed as a small notecard, is inserted next to the title of the card/box the link refers to.

ToCards — includes link icons only to NoteCard descendants of the originating card or box. The link icon, displayed as a small notecard, is inserted next to the title of the card the link refers to.

ToBoxes — includes link icons only to FileBox descendants of the originating or box. The link icon, displayed as a small notecard, is inserted next to the title of the box the link refers to.

NONE — back links are not built for the Document card.

CopyEmbeddedLinks — allows the user to retain or omit embedded links in descendant cards.

ALL — includes any links appearing in descendant NoteCards.

NONE — omits any links appearing in descendant NoteCards.

Select — enables the user to select link types to be included in the Document card. When select is designated, a list of Link Types pops up from which the user selects the link types to include.

ExpandEmbeddedLinks — allows the user to replace embedded links in descendant cards with their text. Note that cards that appear multiple places in the structure defined by the descendant cards are only expanded once, thus avoiding unnecessary duplication of text. In the sample MakeDocument Parameters menu shown above, the *Select* option was chosen, then the *Capital* link type was selected off of the menu listing link types. Therefore, any *Capital* links encountered while building the document will cause the text of the cards at the other end of the link to be embedded in the document at the link's location.

ALL — for any link appearing in the text of a descendant card, the link is replaced by the text of the card linked to.

NONE — leaves embedded links alone.

Select — expands embedded links selectively, depending on the link type.

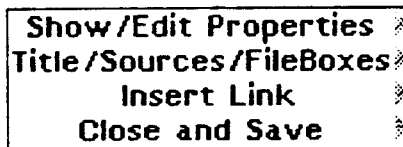
2.2. Selecting Cards and Boxes

Often, the system requires users to select one or more cards. Selections can be made in one of two ways:

1. Place the cursor in the title bar of the card or box and depress the *left* mouse button.
2. Place the cursor over any link icon that refers to the card or box and depress the left mouse button. (See Section 4. Links.)

2.3. NoteCard Menu

The NoteCard Menu contains a list of commands used to manipulate a NoteCard. Display the menu by placing the cursor in the title bar of the card and depressing the *left* mouse button. The following menu appears:



Select the desired command from the menu before releasing the button. Commands followed by a gray triangle pointing to the right have a sub-command menu. This menu is displayed by holding the button down and sliding the cursor to the right through the triangle. Select the sub-command before releasing the button.

The commands and their sub-commands are as follows:

Show/Edit Properties — displays a list of attribute value properties above the card or box. Shown below is an example of a text notecard with its property list displayed:

Edit Property List

```

Certainty      { .5 }
Keywords      {(Property Attribute PI)}
Updates        [(MARSHALL 1-Mar-85 14:18:09)]
Updates        [(MARSHALL.PASA 22-Feb-85 16:44:43)]
ID             [NC00088]
    
```

Modifying the Property List (PI)

This card has had an attribute-value pair assigned to it to express uncertainty and a list of keywords to index its contents.

To add a certainty property from the Programmer's Interface, type the following in the Lisp Exec:

(NCP.CardProp 'NC00088 'Certainty '0.5)

Included in this list are system-maintained properties such as the ID number (in this case, NC00088) and a list of the author, date and time of all updates to the card or box. These system-maintained properties may not be edited. In addition, the user can add or delete

user-defined properties such as keywords or certainty values for the information contained in the card. The advantage of adding these user-defined properties is that they are machine readable and will be used in the future to support machine inferences.

A menu to edit user-defined properties is displayed by placing the cursor in the title bar of the attached property list and depressing the *left* mouse button. Select the command from this menu before releasing the button.

Add New Property — adds user-defined properties to the property list of this card.

Type in the property name and attribute value. Then select a property already in the list before which this new property will be inserted. Properties are displayed in bold type and attribute values are displayed between brackets in regular type.

Delete Selected Property — deletes a user-defined property from the property list of this card. Select the property to be deleted. Properties are displayed in bold type.

Quit w/o Saving Changes — closes the display without saving any of the current changes made with the use of **Add New Property** or **Delete Selected Property**.

Quit - Saving Changes — closes the display saving all current changes made with the use of **Add New Property** or **Delete Selected Property**.

Edit Property List — same as **Show/Edit Properties** above.

Show Links — displays above the card or box a list of all links to and from other cards and boxes. Links are represented in the list by link icons. (See Section 4. *Links*.) Selecting an icon with the *left* mouse button will display the card/box referenced by that link icon.

To close this display, place the cursor in the title bar of the List of Links, depress the left button, and select **Quit** from the single item menu.

Title/Sources/Fileboxes — pops up a prompt window above the card asking the user to follow through with the following: type in a title to the card, designate the sources from which the contents originated, and file the card in one or more FileBoxes.

These operations may be selected individually from among the following sub-menu commands:

Assign Title — pops up a prompt window above the card asking for the card's title. Type in a new or revised title followed by a carriage return.

Designate Sources — pops up a prompt window above the card asking the user to select one or more cards that contain the source of the information in the current card. The source card is sometimes the source material itself or it may be a bibliographic reference to a source that is not in electronic form. Select **Done** from the menu above the prompt window after selecting the source card(s). Alternatively, select **No Source** from the menu to indicate that the contents of the current card do not have a source of information within the NoteFile.

A link icon for the source card is *not* inserted in the body of the current card. The source for any card can only be displayed using the **Show Links** or **Delete Source** commands.

File in FileBoxes — pops up a prompt window above the card asking the user to file the card in one or more FileBoxes.

The initial execution of this command displays a two-item menu above the prompt window. Select **Done** from this menu after designating the FileBoxes to file the card in. Alternatively, select **No Box** to place the card in the "To Be Filed" FileBox (see Section 3.3. Special FileBoxes).

Once a card has been filed in a FileBox, selection of this command offers the choice of filing the card in another box and selecting **Done** from the menu or selecting **Cancel** to abort the command.

Unfile from FileBoxes — unfiles the card from selected FileBoxes. Displays above the card a list (link icons) of all FileBoxes this card is filed in. Unfile the card by selecting the link icon(s) from this list or by selecting the FileBox itself.

Delete Source — deletes a source link. Displays above the card a list of source links emanating from this card. Delete the source assignment by selecting the link icon(s) from this list or by selecting the source card itself.

Insert Link — inserts a user-specified link to another card inside the body of the current card. The link is represented by a link icon. This type of link is used to indicate a relation between cards (see Section 2. NoteCards and Section 4. Links).

Before execution of this command, select a point in the body of the card at which the link icon is to appear.

After selection of the command, a menu pops up displaying a list of user-specified Link Types currently available in the NoteFile. Specify the type of link by selecting a type from this list or select "New Type" to create a new link type. This new type of link is added to the NoteFile and becomes a choice in the list of Link Types. It is not possible to assign a system-reserved type of link to a user-specified link. **Cancel** may also be selected from the list to abort this command.

After a type has been designated, a prompt window and menu pops up above the card asking the user to choose the destination of the link. The user has the option of selecting an existing card/box or of creating a new card/box as the destination by selecting **New Card** from the menu with the *left* button. If this option is chosen, a menu of card types pops up from which the user selects the desired type of card. (See Section 5. Main Menu: *Create*.) Alternatively, select **Cancel** from the menu to abort this command.

The link icon will be inserted at the place in the body of the card indicated by the flashing caret.

Close and Save — saves the NoteCard or FileBox in the NoteFile before closing the card or box. Closing requires that a NoteCard have a title; therefore, the system prompts the user if this has not been satisfied. The user may also be required to file the card in a FileBox unless the **ForceFiling** parameter has been changed to "No". (See *Title/Sources/FileBoxes* above and Section 5. Main Menu: *Change Parameters*.)

Close and Save — same as **Close and Save** above.

Close w/o Saving — closes the NoteCard without saving, in the NoteFile, any changes made since the last save. This is useful if text is mistakenly lost or scrambled during editing. This command asks for confirmation. Type a *carriage return* for yes or type "N" and a *carriage return* for no. (See Section 12.2. Display/Editor Bugs and Fixes for less drastic recovery measures.)

Save in NoteFile — saves, in the NoteFile, all changes made to the NoteCard or FileBox without closing the card or box.

Delete Card — *permanently* deletes the NoteCard from the NoteFile and all its links to and from other cards and boxes. Because this deletion is irreversible, the user is asked to confirm before the delete command is executed. Type a *carriage return* for yes or type "N" and a *carriage return* for no.

2.4. Scrolling

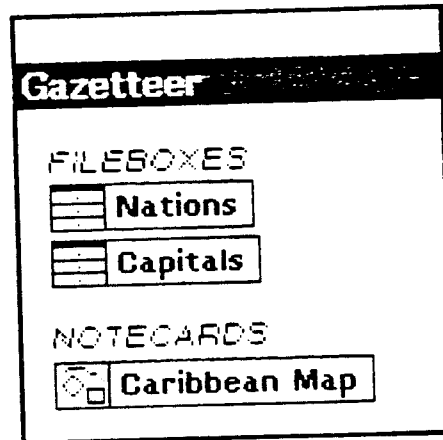
A NoteCard or FileBox may contain more information than is currently visible on the screen. To view more of the contents, a card or box may be scrolled up or down. When the cursor is moved slowly from inside the left edge of the card to the outside a scroll bar appears. The height of the bar represents the entire contents of the card and the grayed area represents the portion currently viewed on the screen. While the cursor is in the scroll bar it becomes a double arrow pointing up and down. When the *left* mouse button is depressed, the contents of the card move up, making the contents toward the end of the card visible. When the *right* mouse button is depressed, the contents of the card move down, making the contents toward the beginning of the card visible. Depressing the *middle* mouse button (depress both buttons for the *middle* if using a two-button mouse) scrolls to a point in the card relative

to the beginning and end of the contents. For example, pressing the *middle* button while the cursor is at the bottom of the scroll bar will bring the contents at the end of the card into view.

Graph or Sketch/Map cards may also be scrolled right and left. The scroll bar is viewed by moving the cursor slowly from inside the lower edge of the card to the outside. The *left* button moves the contents to the left; the *right* button moves the contents to the right; and the *middle* button moves to a relative point in the card.

3. FileBoxes

A FileBox contains a list of link icons to sub-FileBoxes (FileBoxes listed within other FileBoxes) and NoteCards. FileBoxes are organized hierarchically to represent a topic and its sub-topics. The contents of a box may include subtopics represented by sub-FileBoxes, information filed under that topic represented in the various types of NoteCards, or both subtopics and NoteCards. Section 2 describes a non-hierarchical way of organizing cards. An example of a FileBox, *Gazetteer* is shown below. It contains link icons pointing to two sub-FileBoxes, *Nations* and *Capitals*, and a Sketch card, *Caribbean Map*.



3.1. Creating FileBoxes

FileBoxes can be created by selecting **Create** from the **Main Menu** with the *middle* mouse button. Boxes are viewed through windows manipulated by the window menu (see Sections 5. Main Menu: *Create* and 6.1. Window Menu). Approximately 20 boxes and/or cards may be displayed on the screen at one time, with the windows overlapping if desired.

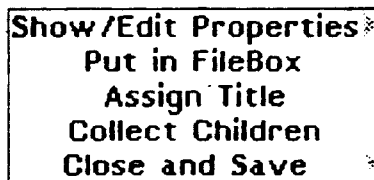
It is necessary to have an existing FileBox or a link icon directed to a box displayed on the screen before creating a new FileBox.

When **FileBox** is selected from the **Create** sub-menu, a border of a box pops up next to the cursor. This border can be moved to any location on the screen by moving the cursor. Plant the border by depressing any mouse button. A FileBox will then replace the border.

After creating a FileBox, it should be placed in a parent FileBox by using its title bar menu command **Put in FileBox** (see Section 3.2. FileBox Menu). To keep within the boundaries of the NoteCard's hierarchical design, certain rules must be followed when placing a FileBox in a parent FileBox. The box can not be put in itself, in one of its sub-boxes, in any sub boxes within those sub-boxes, etc. Finally, a NoteCard can not act as a parent to the FileBox.

3.2. FileBox Menu

The FileBox Menu contains a list of commands for manipulating a FileBox. Display the menu by placing the cursor in the title bar of the box and depressing the *left* mouse button. The following menu will appear by the cursor:



Select the desired command before releasing the button. Commands followed by a gray triangle pointing to the right have a sub-command menu that can be displayed by sliding the cursor to the right through the triangle. Refer to Section 2.3. NoteCard Menu for commands not defined below.

Show/Edit Properties — see Section 2.3. NoteCard Menu.

Edit Property List — see Section 2.3. NoteCard Menu.

Show Links — see Section 2.3. NoteCard Menu.

Put in FileBox — files the current box in some other FileBox. Select one or more FileBoxes to put the current box in. The rules discussed in Section 3.1. for placing a FileBox in another FileBox must be followed. Select **Done** after indicating the parent FileBoxes or select **Cancel** to abort the command.

There is no command for removing a FileBox from another box; however, it is possible to remove a box by using the text editor to *delete* the link icon. Section 7.2. Text Editing Operations: *Deleting Text* discusses this process.

Assign Title — a prompt window pops up above the box asking for a title to the FileBox. Type a new or revised title followed by a carriage return.

Collect Children — allows selection of several cards and boxes to be filed in the current FileBox. Select the cards and boxes in the order in which they are to appear in the FileBox. This command eliminates the need to individually file each card/box into the current FileBox. Section 2.3. NoteCard Menu: *File in FileBoxes* discusses how cards may be filed individually.

Close and Save — see Section 2.3. NoteCard Menu.

Close and Save — see Section 2.3. NoteCard Menu.

Save in NoteFile — see Section 2.3. NoteCard Menu.

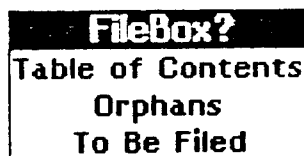
Delete FileBox — *permanently* deletes the FileBox from the NoteFile and all links to the box from other cards and boxes. All cards and boxes that are only filed in this

FileBox are transferred to the *Orphans* box (see Section 3.3. Special FileBoxes). Because this deletion is irreversible, the user is asked to confirm before the delete command is executed. Type a *carriage return* for yes or type "N" followed by a *carriage return* for no

3.3. Special FileBoxes

All Notefiles have three predefined FileBoxes that are necessary to the operation of the system. These boxes cannot be created or deleted by the user, but may be renamed and otherwise treated as regular FileBoxes.

Select the **Show Box** command from the **Main Menu** with the *middle* mouse button (use both buttons for *middle* if using a two button mouse) and choose one of the FileBoxes from the sub-menu illustrated below.



Pressing the **Show Box** with the *left* mouse button automatically brings up the *Table of Contents* FileBox. This command is discussed in more detail in Section 5. Main Menu: *Show Box*

Table of Contents — the top-level FileBox of each NoteFile designed for storage of links to FileBoxes and NoteCards at the highest level of the hierarchy.

Orphans — a backup FileBox for FileBoxes and NoteCards whose last link from another card has been removed.

To permanently delete a card or box, use the **Delete Card** command on the NoteCard title bar menu (see Section 2.3. NoteCard Menu: *Delete Card*) or use the **Close/Delete** option on the Main Menu (see Section 5. Main Menu: *Close/Delete*). Removing a link icon from the Orphans box with the text editor severs the card or box from the FileBox hierarchy but does not delete it from the NoteFile. Generally, *Search* is the only means of accessing a severed card or box.

To Be Filed — a temporary FileBox for NoteCards that are not filed in any FileBoxes. A card may be filed in this box in one of four ways:

1. Choose the **No Box** option offered when first using the **File in FileBox** command.
2. Choose the **No Box** option offered when first using the **Close and Save** command without having previously filed the card. This method applies only if the **ForceFiling** parameter is set to "Yes." (See Section 5. Main Menu: ChangeParameters: *ForceFiling*.)

3. Select the *To Be Filed* box when using the *File in FileBox* command.
4. Use the *Collect Children* command from the **FileBox Menu**. (See Section 3.2. FileBox Menu: *Collect Children*.)

When a card is later retrieved and filed in a specific FileBox, it is *not removed* from the *To Be Filed* box. To unfile a card from a FileBox use the *Unfile from FileBox* command from the NoteCard Menu or manually delete the link icon with the text editor. (See Sections 2.3. NoteCard Menu: *Unfile from FileBox* and 7.2. Text Editing Operations: *Deleting Text*.)

3.4 Suggested FileBoxes and NoteCards

The user may find it helpful to create the following sorts of general FileBoxes. The titles used here are only suggestions; the user may prefer other names for these boxes.

Bibliography — a FileBox for the collection of sources used in the NoteFile.

Index — a FileBox listing keywords from the NoteFile. May be helpful when using *Search*. (See Section 5. Main Menu: *Create*.)

Read Me — a NoteCard in the top level FileBox giving global information about the NoteFile for first time browsers.

Active Cards — a FileBox kept in a top level of the FileBox hierarchy containing FileBoxes and NoteCards that represent work in progress and are thus frequently accessed. A sketch NoteCard containing links to these FileBoxes and NoteCards is another method of organizing active cards, using spatial cues as one way of representing organization.

4. Links

Links provide connections between NoteCards and/or FileBoxes. Two groups of link types exist in the NoteFile system; one group is system reserved and the other is user-specified. Some of the system reserved links support the FileBox/SubBox/FileCard hierarchy the system helps the user to create. The user-specified links add a relational dimension to information structuring. Links are directed - they begin at an originating card and end at a destination card.

4.1. Link Types

Each link between two cards is of a specific type. Some of these types are reserved by the system and others are user-specified.

System Reserved —

SubBox: a link from a FileBox to a SubBox.

FiledCard: a link from a box to a filed card.

Source: a link from a card to a source card.

ListContents: a link pointing to a card or box from a Search card, enabling a user to bring up any of the cards or boxes found during the search process.

BrowserContents: a link pointing to a card or box from a Browser card. This type of link enables a user to bring up any of the cards or boxes represented in a Browser card.

DocBackPtr: a link that is a back pointer to a card or box from a Document card. This type of link is built when the user specifies that a Document card should have back pointers to the cards and boxes used to make it (See Section 2.1. NoteCard Types: *Document*).

LinkIndexBackPtr: a link pointing to a card or box from a Link Index card. This is the type of link built when the user specifies that a Link Index card should have back pointers to the cards and boxes being indexed (See Section 2.1. NoteCard Types: *LinkIndex*).

User Specified — This type of link signifies a user-specified relationship between two NoteCards. The user selects the appropriate link type from a list of types currently in the NoteFile or creates a new type that is then added to the NoteFile (see Section 2.3. NoteCard Menu: *Insert Links*).

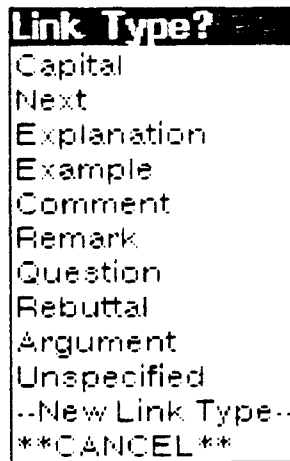
4.2. Link Menu

The Link Menu contains a list of commands used to manage a link icon. Select the *center* of the icon with the *middle* mouse button to display the following commands:

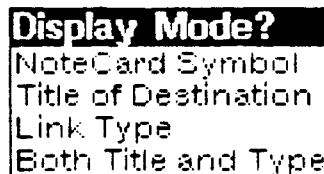


Bring Up Card/Box — displays the card or box referenced by this icon. Note: the card or box may be automatically displayed by selecting the *center* of the icon with the *left* mouse button.

Change Link Type — pops up a menu of link types enabling the user to change the type of link. Only user-specified link types may be changed. An example of this type of pop-up menu is shown below. The options of choosing to name a new link type and cancelling the operation are added to the list of user-specified link types.

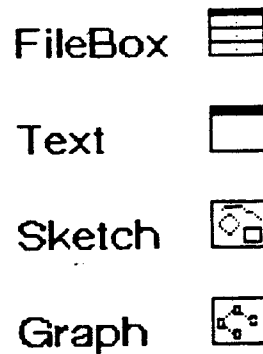


Change Display — pops up the following menu for changing the appearance of the link icon.

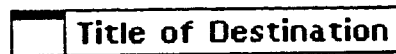


The following options are available.

NoteCard Symbol — gives the link icon a notecard appearance. There are several different link icons depicting primitive NoteCard types. These icons provide a visual cue as to the type of the destination NoteCard. The list shows the correspondence between link type and symbolic representation:



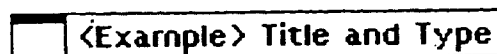
Title of Destination — displays the title of the destination card or box in a rectangle in addition to the NoteCard symbol if the global parameter *AttachBitmapsToLinkIcons* is set (see Section 5. Main Menu: *Set Parameters*). Note that this is the usual display mode for links to filed cards and subBoxes in a FileBox. It may also be a good way to display user-specified links in a card containing many links. For example, a display of a link to a card named "Title of Destination" looks like this:



Link Type — displays the link type (between angle brackets) in a rectangle in addition to the NoteCard symbol if the global parameter *AttachBitmapsToLinkIcons* is set (see Section 5. Main Menu: *Set Parameters*). For example, a display of an user-defined link type "Example" to a card looks like this:



Both Title and Type — displays both the link type (enclosed in angle brackets) and title of the destination card or box in a rectangle in addition to the NoteCard symbol if the global parameter *AttachBitmapsToLinkIcons* is set (see Section 5. Main Menu: *Set Parameters*). For example, a display of an user-defined link type "Example" to a card named "Title and Type" looks like this:



4.3. Creating Links

To create a link between two cards, select *Insert Link* from the left button menu in a card's title bar (See Section 2.3. NoteCard Menu: *Insert Link*). Before selecting this command, be sure to select a place in the text of the card to indicate where the link icon is to appear. First, a menu pops up to prompt you for **link type** (see the discussion of the *Link Type?* menu earlier in this section). Then you're asked to select a card to be at the other end of the link, the card pointed to. Either a new card or any card represented on the screen can be chosen as the destination, or the process may be cancelled.

4.3. Managing Links

Once a NoteFile begins to get large, the tools provided in the system for examining links and link-based structures become very important. Some of the system features to help manage links are the Browser and Link Index cards discussed in Section 2.1. NoteCard Types, the link display operations discussed in this section, and the *Show/Edit Properties: Show Links* option provided in the NoteCard Title Bar Menu discussed in Section 2.3. NoteCard Menu: *Show Links*). An example of how the Show Links and Change Display Modes options may be used to look at the links contained in a card is shown below.

List of Links

Explanation TO	{ <input type="checkbox"/> Structures }
Explanation TO	{ <input type="checkbox"/> Creating a FileBox }
Explanation TO	{ <input type="checkbox"/> Creating Links }

General Notes on Structures

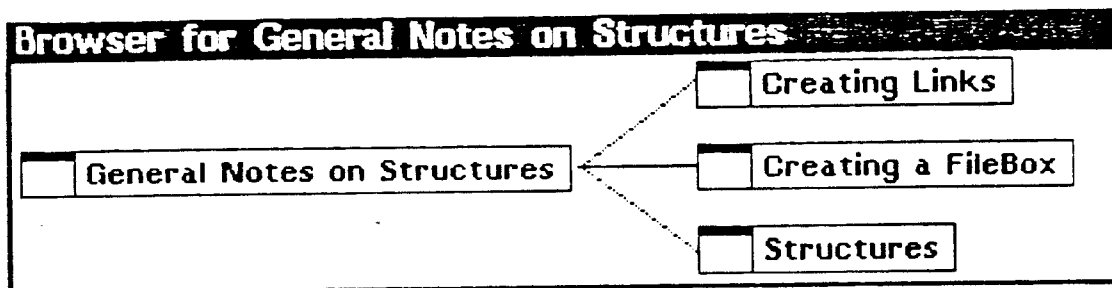
From the discussion of Structures in the Notecards model,
 <Explanation> Structures ,
 you can see that there are several different kinds of structure building which you may use to organize your NoteFile.

To build hierarchical structures of FileBoxes and FiledCards, see
 <Explanation> Creating a FileBox .

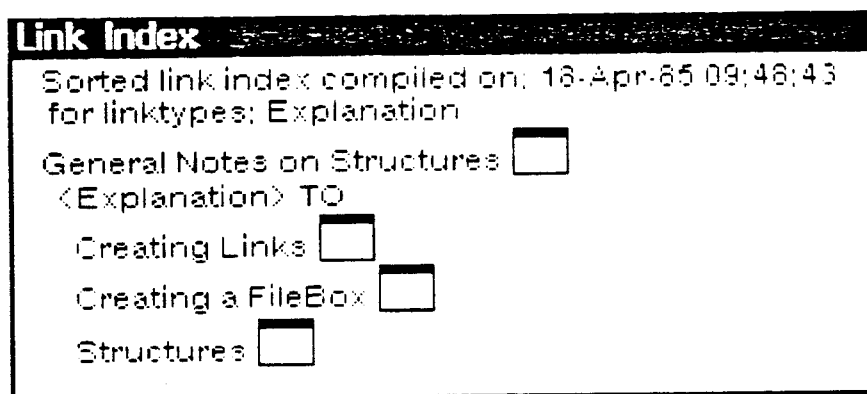
To build relational structures with link types you define, see
 <Explanation> Creating Links .

In the above example, the text card *General Notes on Structures* has three user-specified local-to-global embedded links, all of type *Explanation*. Explanation links have been created to link *General Notes on Structures* to the cards *Structures*, *Creating a FileBox*, and *Creating Links*. The display mode of all three embedded links has been changed to *Both Title and Type*. The *Show/Edit Properties: Show Links* option has been selected from the card's title bar menu. If the *List of Links* were scrolled, a FiledCard FROM link would be visible to show the link from the FileBox, *Building Structures*, to this card.

If the user wished to look at these same Explanation links from *General Notes on Structures* using the Browser mechanism, the following Browser card may be created:



These Explanation links from General Notes on Structures may also be examined using the Link Index facility. The following Link Index card may be created by selecting the Explanation link type as a basis for the index and by opting to build backpointers to the cards linked by this link type to make them easily accessible.



5. Main Menu

Once the NoteCards environment has been initialized, the **Main Menu** is displayed on the screen:

No Open NoteFile	
Close/Delete	NoteFile Ops
Create	Show Box

The **Main Menu** provides access to the range of NoteCards functionality, such as opening and closing your **NoteFile** database (**NoteFile Ops**), walking through your database (**Show Box**), creating new cards (**Create**), and closing or deleting sets of cards (**Close/Delete**). The four commands on the **Main Menu** and their sub-menus are described below:

NoteFile Ops — select with the *left* mouse button to display a list of sub-commands that can be used to manage a **NoteFile**. Select the desired command from the menu below using the *left* mouse button.

Operation?
Change Parameters
Open NoteFile
Close NoteFile
List NoteFiles
Create New NoteFile
Compact NoteFile
Repair NoteFile
Delete NoteFile
Restore From Floppy
Backup To Floppy

Many of these commands will prompt with the name of the most recently accessed **NoteFile** in the black prompt window. The command will act on this **NoteFile** if a *carriage return* is typed. If a different **NoteFile** is specified, the name must be typed into the prompt window and followed by a *carriage return*. **NoteFile** names must end with ".**NoteFile**"; however, if the user omits this suffix, the system will append it.

Several commands may create more than one version of a **NoteFile**. When using **Create New NoteFile**, it is possible to assign an already existing name to the new **NoteFile**, thereby creating more than one version. When a **NoteFile** is compacted, a new version is created from the old, and both versions are saved under the same name. Finally, if a **NoteFile** that already exists on the disk is being restored from a floppy, an additional version will be added to the disk. It is necessary to use the **Delete NoteFile** command to erase any unwanted versions of **NoteFiles** from the **NoteCard** system.

During execution of each sub-command, the system processes the information currently in the NoteFile, and displays information on its progress in the black prompt window (e.g. "Processing Card Number 20 of 365"). Information is not provided for the commands *Restore From Floppy* and *Backup To Floppy*. With the exception of *Change Parameters* and *Close NoteFile*, execution of these sub-commands requires that there be no open NoteFile.

Change Parameters — pops up the border of a box containing a list of parameters. Plant this border by moving it into view and clicking the *left* mouse button. The following parameter/value list will appear:

NoteCards Parameters	
DefaultCardType	Text
FixedTopLevelMenu	No
ShortWindowMenus	Yes
ForceSources	No
ForceFiling	Yes
ForceTitles	Yes
MarkersInFileBoxes	Yes
AttachBitmapsToLinkIcons	Yes
LinkDashingInBrowsers	No
SpecialBrowserSpecs	No
AnnoAccessible	No

Each parameter is preset to a default value. Placing the cursor over the name of the parameter and depressing the *left* mouse button will toggle the value listed to the right. The various parameters follow:

DefaultCardType — when *Create* is selected with the *left* mouse button from the Main Menu, a card of the default type is immediately created and displayed. This parameter allows the user to change this default card type to a preferred choice. Select the new default card type from the list of choices provided.

FixedTopLevelMenu — a value of "No" makes it possible to move the Main Menu about the screen through use of the Window Menu (see Section 6.1. Window Menu: *Move*). The "Yes" value fixes the menu in the center of the upper part of the screen.

ShortWindowMenus — a value of "Yes" sets the Window Menu to a shortened version. (See Section 6.1. Window Menu.) To expand the menu to include *Snap*, *Paint* and *Clear*, change this parameter's value to "No".

ForceSources — the Source mechanism allows the user to specify for each NoteCard a link to the source of the information in that card. If this

parameter is set to "Yes," the user will be required to indicate a source for a NoteCard before that card can be closed.

ForceFiling — if set to "Yes", this parameter requires the user to file a NoteCard into a FileBox before the card can be closed. Setting the value to "No" eliminates this forced filing, but the system will file the card in To Be Filed if it has no parents at when the card is closed. When a card is not filed in a FileBox, it may be retrieved if a link to the card has been inserted into another card. The card may also be retrieved through the use of a Search card. (See Section 2.1. NoteCard Types.)

ForceTitles — if set to "Yes", this parameter requires the user to title a NoteCard before the card can be closed. Setting the value to "No" eliminates this forced titling. When a card is not titled, it is named "Untitled" by default; this may cause confusion if several cards are left untitled. The system has a unique identifier for each card, however, and will continue to regard them as unambiguous entities.

LinkDashingInBrowser — if set to "Yes", this parameter builds a Browser with dashed links. Each link in a Browser refers to the type of link being followed. With link dashing, each link included in the Browser is assigned a different style of dashing (up to six styles). A "No" setting means links will be presented as solid lines in a Browser. A Browser with link dashing will take a little longer to create than one with solid lines.

SpecialBrowserSpecs — if set to "Yes", this parameter causes a sequence of five prompts to appear above the Browser NoteCard during the Browser creation process. These prompts mainly concern the node-link graph properties documented in Section 8. Graph Editor. For example, this option may be used to create a vertical browser instead of the default horizontal presentation. Note that a carriage return given as a response to any of the prompts will cause the SpecialBrowserSpec to retain its default value.

AnnoAccessible — This parameter is not intended for general use.

Open NoteFile — opens an existing NoteFile or, if the name typed in does not exist, asks the user if a new NoteFile by that name should be created. After a NoteFile has been opened, the main menu's title bar changes to reflect the name of the open NoteFile. For example, if a NoteFile named DEMO.NOTEFILE were the currently active open NoteFile, the main menu would appear as follows:

NoteFile: DEMO	
Close/Delete	NoteFile Ops
Create	Show Box

Close NoteFile — closes the currently open NoteFile, and if the user confirms, closes all cards and boxes currently displayed on the screen.

List NoteFiles — displays, in the black prompt window, a list of NoteFiles stored on the local disk or the fileserver. If the list contains more names than the window can accommodate, the window will invert to a white background. A carriage return must be typed to scroll the remainder of the list into view.

Create New NoteFile — creates a new empty NoteFile. It is necessary to select **Open NoteFile** to open a newly created NoteFile.

Compact NoteFile — compacts a NoteFile by rejecting extraneous information. As NoteCards and FileBoxes are revised, old extraneous information from the cards and boxes remains in the NoteFile. Compacting creates a new, compressed version of the NoteFile by copying into the new version only current information from the old version. The old version of the NoteFile remains on the disk until the user deletes it with the **Delete NoteFile** command.

Compacting can take 5 to 20 minutes, depending on the size of the NoteFile.

Repair NoteFile — this command may be helpful in repairing the NoteFile when there are inconsistencies or problem areas in the actual NoteFile (see Section 10 Recovering from System Bugs).

The NoteFile must be closed before executing this command. It is usually best to compact the file after a repair - though this is not necessary.

Delete NoteFile — deletes the oldest version of the NoteFile from the local disk.

The system will ask the user to confirm the deletion by typing a *carriage return* for no or a "Y" and a *carriage return* for yes. If yes, confirmation will be required a second time to insure against unintentional deletion.

Restore From Floppy — retrieves a stored copy of a NoteFile that has been backed up on a floppy. The floppy must be in the drive before execution of this command.

Restoring from a floppy can take 5 to 30 minutes depending on the size of the NoteFile. An illuminated red light on the disk drive indicates that the floppy is either reading or writing a file.

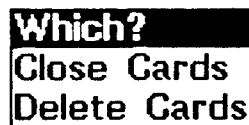
Backup To Floppy — stores a backup copy of a specified NoteFile onto a floppy. This copy can be used as an archive or as a backup to a NoteFile that is somehow destroyed on the local disk. A floppy must be in the drive before execution of this command.

Copying to a floppy can take 5 to 30 minutes depending on the size of the NoteFile. An illuminated red light on the disk drive indicates that the floppy is either reading or writing a file.

Show Box — used to retrieve and display Special FileBoxes. Select this command with the *middle* mouse button to display a sub-menu for the retrieval of the special "Table of Contents," "Orphans," and "To Be Filed" FileBoxes. (See Section 3.3. Special FileBoxes.) Select **Show Box** with the *left* mouse button to immediately display the "Table of Contents" FileBox.

Create — used to create and display new FileBoxes and NoteCards. Select this command with the *left* mouse button to create a new NoteCard of the default type specified by the NoteCards Parameters menu. (See Section 5 Change Parameters: *DefaultCardType*.) Select **Create** with the *middle* mouse button to display a sub-menu for the creation of FileBoxes or any other available NoteCard type. (See Section 2.1. NoteCard Types.)

Close/Delete — displays the following sub-menu.



Close Cards — enables the user to close any number of currently open cards/boxes by simply selecting each one. (See Section 2.2. Selecting Cards and Boxes.) After the choices are specified, select **Done** from the menu which will pop up when **Close Cards** is selected. **Cancel** may be selected to abort this command.

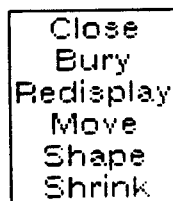
Delete Cards — enables the user to delete any number of currently open cards/boxes by simply selecting each one. After the choices are specified, select **Done** from the menu which will pop up when **Close Cards** is selected. **Cancel** may be selected to abort this command. These deletions are irreversible once executed.

6. Screen Management

All contents of the NoteFile are viewed through windows. Windows are standard Interlisp-D facilities and are manipulated with individual window menus. Windows may be overlapped and can be moved about the screen with the menu or mouse.

6.1. Window Menu

Display the following menu by placing the cursor in the title bar of the card or box and depressing the *right* mouse button. Select the relevant command before releasing the button.



This menu can be expanded to include *Snap*, *Paint* and *Clear* (not defined here) with the **Change Parameters** command from the Main Menu. (See Section 5 Main Menu: *Change Parameters*.) The commands of this menu are as follows:

Close — similar to **Close and Save** without access to the sub-menu (See Sections 2.3. NoteCard Menu: *Close and Save* and 6.2. Mouse Functions.)

Bury — buries the current card or box at the bottom of a stack making it possible to view previously hidden cards or boxes.

Redisplay — redisplay the contents of the card or box. This is useful when the contents appear garbled.

Move — moves the card or box to a new location on the screen. When this command is selected, the cursor becomes a square and jumps to the upper right corner of the card. Attached to the square is an outline similar in size to the card. Move this outline by moving the mouse to a new location. Press the left button and the card will jump to its new position. (See Section 6.2. Mouse Functions for an alternative method.)

Shape — allows the user to specify a new shape and/or location for the existing card or box. After selecting this command a small outline pops up next to the cursor. There are two options for shaping:

Reshape — when the *middle* button is depressed, the cursor will take on the shape of the nearest corner of the card or box with the outline expanding to meet the opposite corner. Keeping the button down and moving the cursor expands or shrinks the outline to the desired shape. Releasing the button pops the card into


its new shape. This method is useful for making small adjustments to a card or box that is already positioned correctly.

Relocate and *Reshape* — move the outline to a new location, depress the *left* button, shape the outline by moving the square and release the button. The card will jump to its new location in its new shape.

Shrink — shrinks the card or box down to its title bar. The card or box can be restored by selecting *Expand* from the window menu or by selecting a pointer icon pointing to the card or box.

Expand — restores the card or box associated with the title bar.

6.2. Mouse Functions

When the cursor is outside all cards and boxes, depressing the *left* button will transform the cursor to a square: . This square can be used to either move or close a card or box

Move — keeping the button down, move the square into the card/box near a corner. Then move the square out of the card/box passing the square through the same corner. The square will pick up an outline of the card/box. Move this outline to a new location on the screen by moving the square. Release the button and the card or box will jump to the new position. (See Section 6.1. Window Menu: *Move* for an alternative method.)

Close — keeping the button down, pass the square several times through the middle of a border of the card or box. This movement will close the card or box. This **Close** is similar to **Close and Save** without access to the sub-menu. (See Sections 2.2 NoteCard Menu: *Close and Save* and 6.1. Window Menu: *Close*.)

7. Text Editor

Text NoteCards are designed for the collection of characters typed in by the user. When a text card is created, a blank card is displayed on the screen with a flashing caret indicating where to begin typing. Text may then be edited using the standard Interlisp-D text editor, TEdit.

This text editor is also used as the editor for all FileBoxes.

TEdit operates on "selected" text. The editor uses a combination of standard keys and special keys from the Dandelion keyboard to aid the user when editing a card or box. Because these special keys are not obviously labeled on the keyboard a description of location is necessary.

BS — the backspace key. This key is the last key to the right in the top row of keys in the main key pad.

Ctrl — the control key. This key is located at the bottom in the right column of the left hand key pad.

Esc — the escape key. This key is the first key to the left in the top row of keys in the main key pad.

Lock — the shift-lock key. This key is the fifth key from the left in the very top row of keys.

Tab — the tab key. This key is the first key to the left in the second row of keys in the main key pad and is just under the **Esc** key.

Undo — the undo key. This key is the lower key in the middle column of the right-hand key pad.

Unlock — the shift-unlock key. This key is the sixth key from the left in the very top row of keys.

"↑" — to type an upward pointing arrow, use the last key to the right in the second row from the bottom of the main keyboard in conjunction with the shift key.

7.1. Selecting Text

Selected text is highlighted in some way. Insertions begin at the current caret position; deletion and other operations are applied to the currently selected text. Refer to the section above describing the Dandelion keyboard for any unfamiliar keys.

Text is selected using the mouse. There are two regions within a text card: The area containing text, and a "line bar" just inside the left edge of the card. While the cursor is inside the text region, it appears as the normal up-and-left-pointing arrow. When the cursor moves into the line bar, it changes to an up-and-right-pointing arrow.

The *left* mouse button always selects the smallest things. In the text region, it selects the character the cursor is pointing to; in the line bar, it selects the single line the cursor is pointing to.

The *middle* mouse button selects larger things (select with both buttons if using a two-button mouse). In the text region, it selects the word the cursor is over, and in the line bar it selects the paragraph the cursor is next to.

The *right* button always extends a selection. The current selection is extended to include the character/word/line/paragraph you are now pointing at. For example, if the existing selection was a whole-word selection, the extended selection will also consist of whole words.

There are special ways of selecting text which carry an implicit command with them:

Ctrl-selection — Holding the *ctrl* key down while selecting text specifies a "delete" selection. The text will be shown white-on-black. When you release the *ctrl* key, the selected text will be deleted. To abort a *ctrl*-selection, release the *ctrl* key before releasing the mouse button. Reselect with the mouse, if necessary, to achieve this.

Shift-selection — Holding the *shift* key down while selecting text is a "copy-source" selection. A copy source is marked with a dashed underline. Whatever is selected as a copy source when the *shift* key is released will be copied to the current caret position. This also works when copying text from one text card to another. To abort a copy, release the *shift* key before releasing the mouse button.

Ctrl-shift-selection — Holding the *ctrl* and *shift* keys down while selecting text is a "move" selection. A move is marked by white-on-black. Whatever is selected as a "move" source when the *ctrl* and *shift* keys are released will be moved to the current caret position. This also works when moving text from one text card to another. To abort a move, release the *ctrl* and *shift* keys before releasing the mouse button.

7.2. Text Editing Operations

Inserting text — Except for command characters, whatever is typed on the keyboard is inserted at the current caret position. The *BS* key and *ctrl-A* both act as a backspace, deleting the character just before the caret. *Ctrl-W* is the backspace-word command.

Deleting text — Hitting the *delete* key causes the currently-selected text to be deleted. The *ctrl*-selection method described above may also be used.

Copying text — Use *shift*-selection, as described above. This operation also works when copying text from one text card to another.

Moving text — Use *ctrl-shift*-selection, as described above. This operation also works when moving text from one text card to another.

Undoing an edit operation — The lower key of the middle column of the right-hand key pad is the Undo key. It will undo the most recent edit command. Undo is itself undo-able, so you can never back up more than a single command.

Redoing an edit operation — The *esc* key is the Redo key. It will redo the most recent edit command on the current selection. For example, if you insert some text, then select elsewhere, hitting ESC will insert a copy of the text in the new place also. If the last command was a delete, Redo will delete the currently-selected text; if it was a font change, the same change will be applied to the current selection.

7.3. Text Editor Menu

The Text Editor Menu contains a list of commands used to edit text in a FileBox or Text NoteCard. Display the following menu by placing the cursor in the title bar of the box or card and depressing the *middle* mouse button.

<p style="text-align: center;">Restart Editor Advanced Editing Menu Change Font</p>
--

Restart Editor — restarts the text editor. Use this command when edits are incorrectly displayed on the screen.

Advanced Editing Menu — displays above the text card or box a list of commands used for editing. The Advanced Editing Menu contains selectable menu buttons and places to type text (e.g., what to search for when you do a *Find*). The menu can be edited, so the usual editing operations work--with one change. Some parts of the menu can't be selected or operated on: they're protected. The places you can select are: menu buttons, the margin ruler (see below), and between pairs of curly braces, so: {}.

Menu buttons appear in bold; every menu button which needs to ask for text has a pair of {} associated with it, e.g., the line

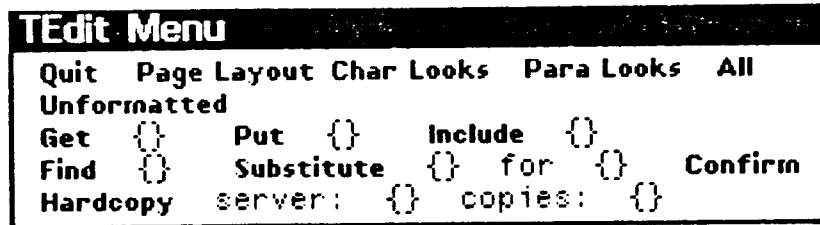
Quit Hardcopy server: {} copies: {}

has two buttons on it. The **Quit** button needs no further arguments, while the **Hardcopy** button can take two arguments: the name of the server to print to and the number of copies to print. Fill the text in before hitting the menu button.

Scroll the display to see all possible commands (see Section 2.4. Scrolling). Select the desired command with the *left* mouse button.

Note: because this text editor is cumbersome the user may want to use it only when special formatting is required.

The TEdit operation menu looks like this:



Quit — similar to *Close and Save* without access to the sub-menu (see Sections 2.3. NoteCard Menu: *Close and Save*).

CloseMenu — closes the display of text editing commands.

All — selects all characters within the card or box.

Get — replaces the text in the card with the contents of the file whose name is specified between the brackets {}.

Put — puts the text in the card into a file with the name specified between the brackets {}.

Include — includes the contents of the file whose name is specified between the brackets {} at the position designated by the caret.

Find — hunts for a match to a string of characters supplied by the user.

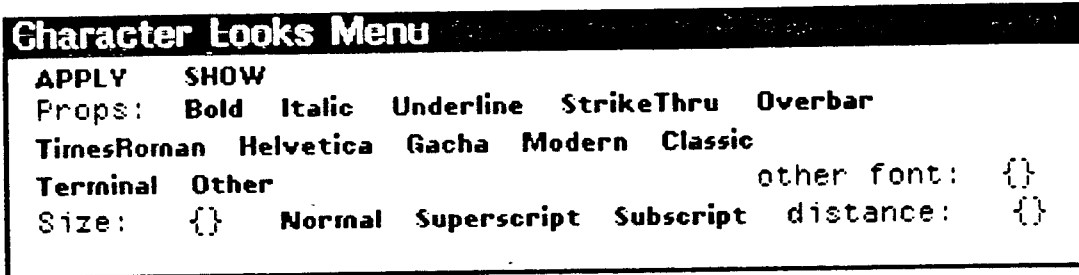
Type a string of characters between the brackets {}, then select **Find**. The system then hunts from the caret toward the end of document for a match. Selects the first match found; if there is none, nothing happens.

Substitute — substitutes, within the current selection, all instances of a search string of characters, supplied by the user, with a replacement string of characters, supplied by the user.

Type a search string and a replacement string between the brackets {}, then select **Substitute**.

Hardcopy — takes two optional text arguments. If you specify a server name, the hardcopy will be sent there. You may also specify how many copies of the document you want; if you don't put anything in the "copies" field, you get one copy.

The Character Looks Menu changes the character looks of the selected characters: the font, character size, and face (bold, italic, etc.). The menu looks like this:



Generally speaking, you select the text you want to change, set the entries in this menu up as you want the text to appear, then make the change by hitting the **APPLY** button.

If you have a piece of text whose looks you want to copy, select the text and hit the **SHOW** button. The menu will be filled in to match that text's looks. You can then **APPLY** it elsewhere, perhaps after modifying things slightly.

The second line of the menu is a list of character properties which can be modified independently. Each of the menu buttons has three states: If the button appears white-on-black, that property will be turned on; If the button appears with a diagonal line, that property will be turned off; If the button appears black-on-white, that property will be left alone.

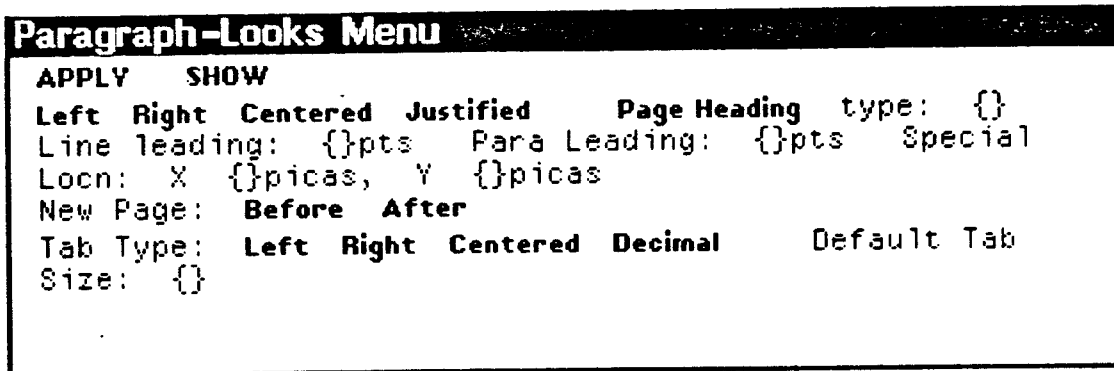
Why is it useful to leave a property alone? Suppose you have a paragraph in Times Roman with some bold and some italic in it. If you want to change the font to Helvetica without changing the boldness or italicness, you can do so.

The third line of the menu is a list of font-family names. You can select among them: selecting one family deselects any others. You can also select no family by mouse buttoning between two of the families. If you **APPLY** with no font family selected, the text will be left in whatever font family it was.

The last line of the menu lets you set the font's size, and specify any superscripting or subscripting. Fill in the "Size:" field with a number, and **APPLY**ing will change all the selected characters to that size. Leave it empty, and the characters will retain their existing sizes.

For character offsets, you have three choices: Normal characters lie on the baseline; Superscript characters lie above the baseline by the distance you specify (2 points by default); Subscript characters lie below the baseline by the distance you specify (2 points by default). As with font family names, you may mouse in the space between options to neutralize the choice. **APPLY**ing with a neutral choice leaves characters with the super- and subscripting they had, if any.

The Paragraph Looks Menu changes the looks of the selected paragraph. Included is a ruler to indicate paragraph margins. The menu looks like this:



Below this menu, if you scroll it up far enough, is a solid black rectangle, used for setting indentations, and a ruler, used for setting tab stops.

As with the Character Looks Menu, you select the text you want to change, set the entries in this menu up as you want the text to appear, then make the change by hitting the APPLY button.

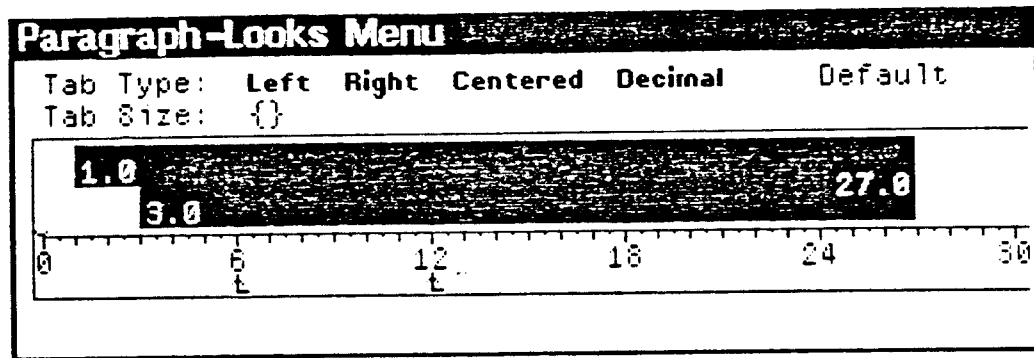
If you have a paragraph whose looks you want to copy, select the text and hit the SHOW button. The menu will be filled in to match that text's looks. You can then APPLY it elsewhere, perhaps after modifying things slightly.

The second line of the menu is for specifying how the paragraph margins are to be justified. A Left justified paragraph has a ragged right margin, but is justified flush with the left margin. A Right justified paragraph has a ragged left margin, but is justified flush on the right. A Centered paragraph is centered between the two margins. A Justified paragraph is set flush with both the left and right margins.

The Page Heading property refers to the headings defined by using the Page Layout menu. For example, the Page Layout menu might define a heading called "Chapter" to be included at a specified location on every left page. The Page Heading button on the Paragraph Looks menu may be used to indicate the actual text which will be the heading for this chapter.

The space between lines in a paragraph is called "line leading". You can specify it, in units of printer's points. You can also leave space in front of a paragraph (without using extra carriage returns) by specifying "paragraph leading," also in units of printer's points.

You set paragraph margins using the margin ruler on the bottom of the Paragraph-Looks Menu. The Paragraph-Looks Menu scrolled to show the margin ruler and tab stops appears below.



There are three margin values: The left margin for the paragraph's first line, the left margin for the rest of the paragraph, and the paragraph's right margin. For example, on the menu shown above, the left margin for the paragraph's first line is set at 1.0, the left margin for the rest of the paragraph is set at 3.0, and the paragraph's right margin is set at 27.0. The margin ruler has three sensitive areas, one for each margin value. Margins are measured in printer's picas (6 to the inch), with a grain of 1/2 pica. There are 12 points to the pica. Plans exist for allowing different units (and granularity) in the ruler.

The first-line left margin is controlled by the top half of the ruler, left end. To move it, push a mouse button near the left edge, and hold it. Moving the mouse pulls the margin along with it: the margin ruler always shows the current values of the margins. If you push the RIGHT mouse button over the margin, it becomes neutral: i.e., APPLYing the paragraph menu won't change the first-line left margins of any paragraphs.

The rest-of-paragraph left margin is controlled by the bottom half of the ruler, left end. You move it (and neutralize it) the same way.

Likewise for the right margin, which is controlled by the right end of the margin ruler. There are a couple of differences here. First, you can set the right margin to 0, which will create a "floating" right margin (one that follows the right edge of the edit window or of the printed page). This is signalled by a margin ruler that is as wide as the window, but shows a value of 0 at its right end.

Since the editing window may be narrower than the document, you can also set the right margin beyond the edge of the window, by pulling it with the mouse, and pulling past the window edge. A right margin you can't see is represented by a double wavy line at the right edge.

You can also set tab stops using the margin ruler. The space below the ruler markings is sensitive to all three mouse buttons, and is used to represent tab stops. In the Paragraph-Looks Menu shown above, there are two tab stops set, one at 6 and another at 12.

To set a tab, you first need to choose what kind of tab you want, using the line starting with "Tab Type:." Make your choice of tab type the same way you'd choose a font family. Left tabs are regular typewriter type tabs; Right tabs take the succeeding text and push it so it is flush-right against the tab stop location; Centered tabs cause the succeeding text to be centered about the tab stop; Decimal tabs (not implemented) cause the succeeding text to have its decimal point lined up on the tab stop. Tab stops are shown in the margin ruler as small arrows with suggestive tails.

To create a new tab stop, use the middle mouse button. In the region below the ruler markings (and the numbers!), point to where you want the tab to be, and press the middle mouse button. The tab should appear; as long as you hold the button down, the tab will follow the mouse around, so you can adjust its location. To move a tab stop, point at it and press the left mouse button. As long as you hold it down, the tab stop will follow the mouse. To delete a tab stop, point at it and press the right mouse button.

Change Font — pops up three separate menus allowing the user to change the font of selected text: font type, style and size. Select the characters that are to be changed before selecting this command.

8. Graph Editor

The "Graph" NoteCard is designed to allow the user to construct a layout of user-defined words or phrases, called nodes, which may be linked together with lines to indicate some structure. Each node may be easily moved about the card without losing its links.

8.1. Graph Editing Menu

The Graph Editing menu is a standard feature of Interlisp-D. Display the following menu by depressing the *right* mouse button in the body of the card. Select the desired command before releasing the button.

```

Move Node
Add Node
Delete Node
Add Link
Delete Link
label smaller
label larger
↔ Directed
↔ Sides
↔ Border
↔ Shade
STOP

```

Prompts for several of the following commands are viewed in the black window at the top of the screen. To close the menu before making a selection, move the cursor outside the menu before releasing the button.

Move Node — select the node to be moved with the *left* mouse button, move the node to its new position, and release the button.

Add Node — pops up a window prompting the user to type in a node label name followed with a carriage return. The label will appear next to the cursor within the graph card. Move the label, by moving the cursor, to its desired location. Plant the label by clicking any mouse button.

Delete Node — select the node to be deleted with the *left* mouse button. The black prompt window will expect a "Y" for yes or an "N" for no followed by a carriage return to confirm deletion.

Add Link — select the "from" node and the "to" node with the *left* mouse button. If a another link is made between the same two nodes in the opposite direction, the lines representing the links will not be visible.

Delete Link — select the "from" node and the "to" node with the *left* mouse button to delete the link.

label smaller — decreases the font size of the selected node. Repeat this command as necessary to achieve the desired font size. This command does not work on pointer icon labels.

label larger — increases the font size of the selected node. Repeat this command as necessary to achieve the desired font size. This command does not work on pointer icon labels.

<-> Directed — moves all links from top to bottom of node.

<-> Sides — moves all links from left and right sides to top and bottom sides of node.

<-> Border — draws a rectangular border around the node selected. Select the node to have a border drawn around it with the *left* mouse button. To remove a border, re-apply this option.

<-> Shade — inverts the shade of the rectangular around the node selected. For example, a black label on a white background becomes a white label on a black rectangular background. Select the node to be inverted with the *left* mouse button. To change the shade back, re-apply this option.

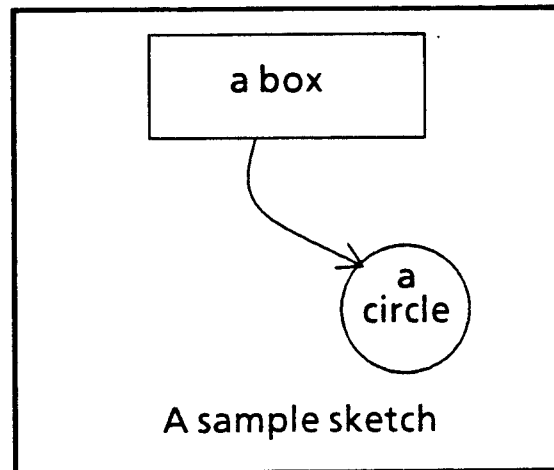
STOP — similar to *Close and Save* without access to the sub-menu (see Sections 2.3. NoteCard Menu: *Close and Save*).

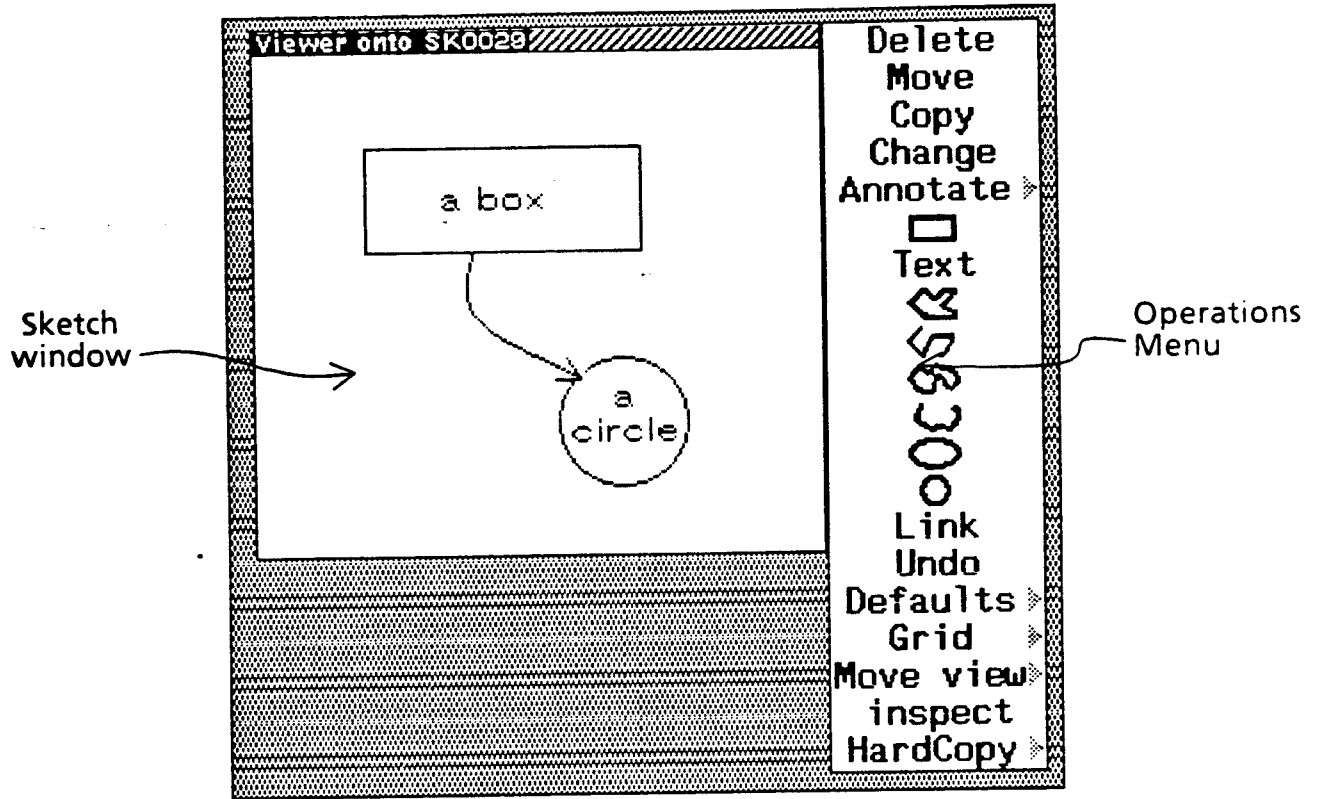
9. Sketch Editor

9.1. Introduction

A Sketch NoteCard implements a drawing program modelled after a whiteboard in which the user is able to place text and graphics to achieve desired images. The figures can be included (via copy-select) in Text NoteCards to allow a mixture of text and graphics in the same document.

A sketch consists of "sketch elements" in a floating point coordinate space. Sketch elements include text, lines, curves, boxes, circles and ellipses. Each element has one or more positions (called control points) that determine its location, and a set of properties such as brush. A sketch is viewed in one or more sketch cards that provides a region and scale. Within a card, the sketch can be edited by adding or deleting elements, or by changing the locations of control points or the values of properties.

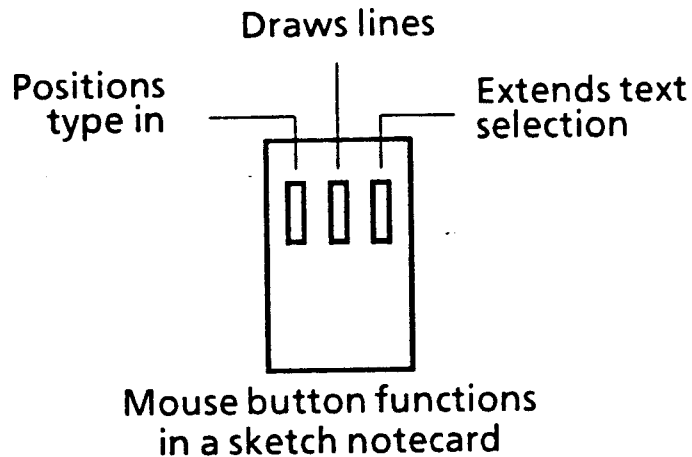




A sketch window with its operation menu

9.2. Sketch Mouse Functions

Within the Sketch NoteCard, the three mouse buttons provide quick access to text editing and line drawing. The *left* button locates typed characters. If this is within existing text, a vertical bar will appear between the existing characters. If this position is not within existing text, a caret shape will appear and a new piece of text will be centered around that position when typing occurs. The *right* button is used to extend a selection within an existing piece of text and any text so selected will be deleted when the next character is typed.

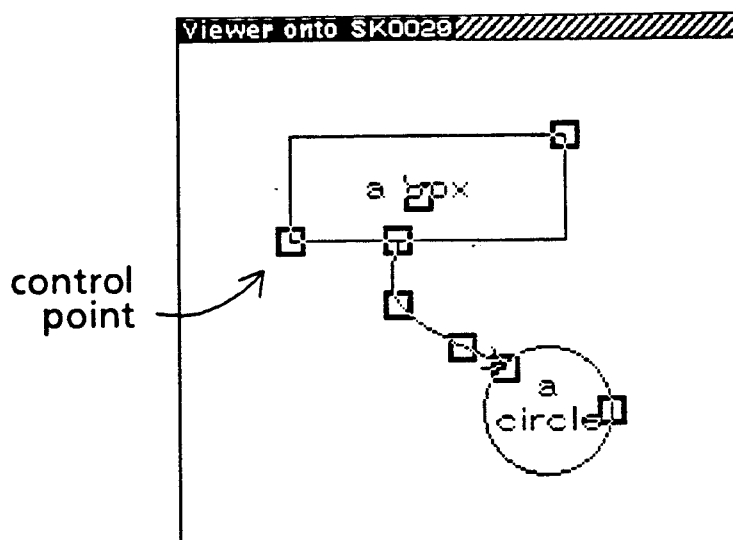


The *middle* button is used to construct straight lines. Pressing the *middle* button will cause an arrow to track the cursor (though only on grid points - described in section below on "grid"). Releasing the button marks the beginning of the line. Pressing the *middle* button again will stretch a line from the beginning point to the cursor. This line will move until the *middle* button is released again. More lines can be added by pressing the *middle* button. Each line starts where the previous one ended. To start a new series of lines, press any mouse button outside the window. The drawing of any line segment can be stopped by moving the mouse outside the window. This does not start a new series of lines; moving back into the window and pressing the *middle* button will pick up where the last line ended.

9.3. Sketch Editing Menus

Sketch has three menus retrieved by placing the cursor in the title bar of the window and depressing the appropriate mouse button. The *left* mouse button retrieves the standard NoteCards menu. *Middle* retrieves the Sketch menu (select with both buttons if using a two-button mouse), and *right* the standard window menu.

The Sketch menu (see the first figure) is designed to aid the user in creating sketches. Retrieve the menu by depressing the *middle* mouse button in the title bar and select the command before releasing the button. If more than one command is going to be executed, it is convenient to keep the menu visible. This is achieved by selecting **Fix Menu** from the Sketch Menu (see **Fix Menu** below). Select commands with the *left* or *middle* mouse button. Make all selections of graphic elements within the sketch window with the *left* button.



A sketch notecard in selection mode
(control points visible)

9.4. Sketch Element Selection

Several of the commands ('delete', 'copy', 'change' and 'move') apply to existing elements in the sketch. These commands are used by first selecting the command from the command menu, then specifying the control point, element or elements to apply the command to. The elements are selected by the following protocol. The 'move' command is the only command that accepts points. For the 'move' command, a point can be selected by placing the cursor within the square surrounding that point and clicking the *left* button. A second immediate click (double clicking) of the left button will select the element of which the point is a part. A third click will select the entire sketch. For commands other than 'move', an element can be selected by placing the cursor within the square surrounding any of its points and clicking the *left* button. A group of elements can be selected by moving the cursor to one corner of the group, pressing the *left* button, then while holding the *left* button down, sweeping to the opposite corner. This will select all elements in the swept out area. The entire sketch can be selected by clicking the *left* button twice in the same square. The elements selected will be marked by having their control points blackened. A selected element can be deselected when the *left* button is held down by also holding down the *right* button.

9.5. Sketch Menu Commands

The menu has the following commands:

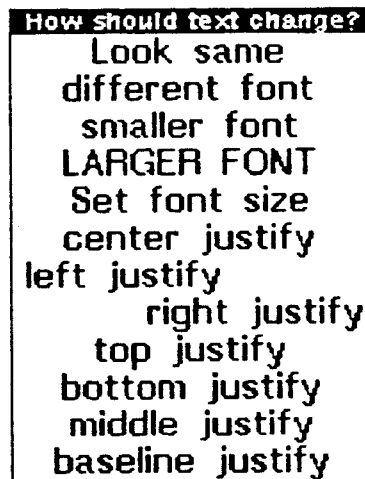
Delete — deletes an element or group of elements. Note that delete cancels the entire graphic object. All graphic objects are identified when **Delete** is chosen, and the user must select the object to be deleted. To cancel this selection before deleting, place the cursor outside the sketch card and click a mouse button.

Move — moves a selected control point, an element or a group of elements. If a single point is selected, the cursor will change to \oplus , a new location for that point should be selected by pressing the left button. To stop the move, select a point outside the sketch window.

If an element or a group of elements is selected, an image of the selected elements will be shown on the screen and will follow the cursor until the *left* button is pressed and released. When the button is released, the selected elements will be moved to the shown locations. To stop the move, press the *left* button when the entire image is out of the window. (Note: An image containing curves or ellipses may be clipped at the edges. The complete image will be moved.)

Copy — similar to **Move** except that it makes a copy of the selected elements in the new position.

Change — enables the user to change the looks of text, a line or a curve. All graphic elements are identified when **Change** is chosen, and the user must select the elements to be changed. The **Change** commands vary for different types of elements because each has different properties. If in doubt, select the object you want to change and see what choices are offered. Selecting outside of the offered menu will not change anything. If the selection contains elements of different types, the offered change will be for the type that is first selected and the change will be made to any other selected elements to which it is applicable



Menu offered for changing text elements

Text — enables the user to change the font, size and justification of the text string relative to its position. The following sub-menu:

Look same — enables the user to make all of a collection of text look alike. It makes all of the selected text items be the same font size, and alignment as the first selected item. This works on the selected text items only. It provides a way of making text that was entered at different scales to look the same.

different font — enables the user to change the type of the font. The user will be offered a menu of the previously entered font families and the item **other**. If the item **other** is selected, you will be prompted for the name of a font family. If the selected or named font is not available in the size of the first selected text element, an error message is printed and nothing is changed. If an appropriately sized font is found, any of the selected text elements that have the same size as the first selected text element will be changed to the new font. The search for fonts will encompass any directories on DISPLAYFONTDIRECTORIES and may take a few minutes. For PRESS fonts, there is no way of determining what sizes of a given font exist. Sketch will try to create sizes 8, 10 and 12 in new fonts. If it doesn't find a font that you believe exists, you can create that font by making it the default font (both family and size) and retrying the change.

Font size control: *Smaller font, Larger font, Set font size*

Horizontal control: *Center Justify, Right Justify, Left Justify*

Vertical control: *Top Justify, Middle Justify, Baseline Justify
Bottom Justify*

Closed Curves, Circles and Ellipses — enables the user to change the brush size and shape with the following sub-menus:

Shape — choose from the following sub-menu. Note: the printers only support **Round** brushes.

Round, Square, Vertical, Horizontal, Diagonal

Size — Choose from the following sub-menu:

smaller line — makes the curve one thinner.

larger line — makes the curve one thicker.

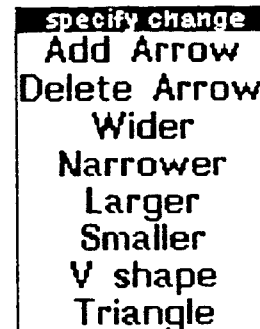
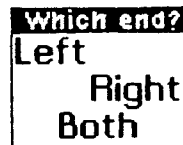
Set line size — prompts for a number and sets the line to this thickness.

Open Curves — enables the user to change the brush size and shape or add and change arrowheads at either end of the curve.



First menu for changing open curves

You will first be asked to specify which aspect of the curve you wish to change. The aspects are Shape, Size and Arrowheads. Depending upon your choice, you will be prompted with different other menus.



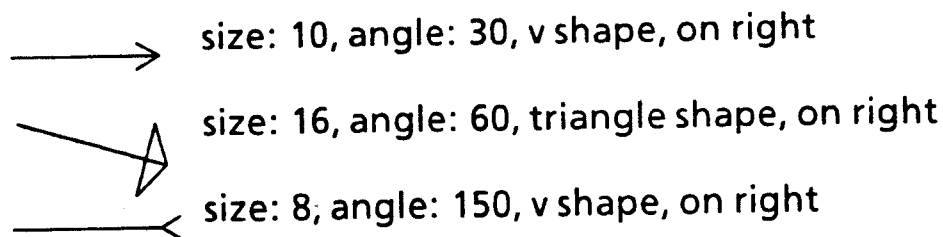
Menus for changing arrowheads

Arrowheads — You will be given two menus. The first one specifies which end of the curve this change is to affect. The second one specifies the change to make. The choices in the first menu are:

- Left** — change arrowhead at the leftmost end of the curve (topmost if ends have the same X position.)
- Right** — change the arrowhead at the rightmost end of the curve (bottommost if ends have the same X position.)
- Both** — change applies to the arrowheads at both ends of the curve.

The choices in the second menu are:

- Add Arrow** — adds an arrowhead of the default specifications at the previously specified end or ends. If that end already has an arrowhead, nothing happens.
- Delete Arrow** — deletes an arrowhead at the previously specified end or ends. If that end doesn't have an arrowhead, nothing happens.
- Wider** — increases the angle of the arrowhead with respect to the line.
- Narrower** — decreases the angle of the arrowhead with respect to the line.
- Larger** — increases the length of the side of the arrowhead.
- Smaller** — decreases the length of the side of the arrowhead.
- V shape** — changes the shape of the arrowhead to be two sides.
- Triangle** — changes the shape of the arrowhead to be two sides and a base.



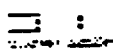
Sample arrowheads

Shape — same as for **Curves**.


Size — same as for **Curves**.


Lines — change enables the user to change the arrowhead characteristics and the size in the same manner as **Open Curves**.


Boxes — change enables the user to change the line thickness in the same manner as changing the brush size of **Curves**.


 — to sketch boxes. Prompts for two points which become the corners of the box.


Text — pops up a window prompting the user to type in a single line of text (terminated by a carriage return). Place the cursor in the sketch card. After pressing return, the text string will appear next to the cursor. The text can be located by moving the cursor with the mouse and depressing any mouse button to plant the text at the desired location. The command can be cancelled by placing the text outside of the sketch window. (See 8.1. **Mouse Functions** for an alternative way of entering text.)

 — to sketch closed lines. A line is defined by a number of special points called knots. These knots are placed in the window by depressing any mouse button while moving the cursor in a desired pattern. To draw a line through these knots, move the cursor outside the window and click any mouse button. To cancel this command, click the button with the cursor outside the window before entering any knots.

 — to sketch open lines. See closed lines procedure.

 — to sketch closed curves. See closed lines procedure.

 — to sketch open curves. See closed lines procedure.

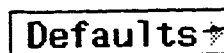
 — to sketch an ellipse. Indicate the center of the ellipse by depressing the *left* mouse button in the window. Indicate the semi-major axis by depressing the *left* mouse button in the

window. Indicate the semi-minor axis by depressing the *left* mouse button in the window. This command can be canceled by clicking outside the window.

O — to sketch a circle. Indicate the center of the circle by depressing the *left* mouse button in the window. Indicate the point of the circumference of the circle by depressing the *left* mouse button in the window. This command can be canceled by clicking outside the window.

Undo — to undo a previous command. Select the event from the undo window. The most recent event is placed at the top of the list. ("Wire" refers to lines.) If the same graphical element has been changed more than once, the changes to it must be undone in the reverse order in which they were done. To cancel this command, click the button with the cursor outside the card before selecting an event.

Defaults — enables the user to change the default brush, text and arrowhead characteristics. The default characteristics are the properties given to a newly added line, text or arrowhead. (The properties of an existing object can be changed by using the **Change** command above.)



Submenu indicator

To access submenu:
press button and roll
out the right side.

'Default' menu item showing
submenu indicator

This command is followed by a gray triangle pointing to the right indicating a sub-menu (see figure). To retrieve this sub-menu, press down the *left* button, slide the cursor to the right through the triangle, and select the desired sub-command before releasing the button. Continue holding the button and slide through the desired sub-menu item to obtain the sub-sub-menus. When the desired item is found, release the button while over it. The sub-menu allows the user to change the brush or the arrowhead characteristics.

Brush — enables the user to change the default brush size. A pad of numbers is displayed from which the user chooses the brush size and clicks OK to exit. If the brush size is not to change, select OK when the size is 0 to cancel the command.

A sub-menu allows the user to change the default brush shape or size.

Shape — enables the user to change the brush shape. The most recent change becomes the default for all new elements created. Choose from the following sub-menu:

Round, Square, Vertical, Horizontal, Diagonal

Size — same as **Brush** above.

Arrowhead — enables the user to change the default arrowhead size. A pad of numbers is displayed from which the user chooses the arrowhead size and clicks OK to exit. If the

arrowhead size is not to change, select OK when the size is 0 to cancel the command. The arrowhead size is given in screen points. When the arrowhead is added, the screen point size is scaled to get its real sketch size. This size is then scaled as the line or curve scales.

A sub-menu allows the user to change the default arrowhead size, angle and type.

Size — same as **Arrowhead** above.

Angle — enables the user to change the angle of the default arrowhead. A pad of numbers is displayed from which the user chooses the angle in degrees. If the arrowhead angle is not to change, select OK when the size is 0 to cancel the command.

Type — enables the user to change the type of the default arrowhead. A menu of type is displayed from which the user chooses the new default. Currently there are two types: **V-shape** — the arrowhead consists of two lines from the head and **Triangle** — the arrowhead consists of a triangle (two lines from the head and a line connecting their endpoints.)

Text — enables the user to change the size of new text. A menu of the known font sizes is displayed from which the user chooses the size. Note: not all fonts are available in all sizes. The sub-menu allows the user to change the size and alignment properties of new text.

Size — same as **Text** above.

Font — enables the user to change the font family property of newly entered text. The user specifies a font family using the same method as when the font property of text is changed (described under the 'change' command). If the specified font can not be found in the current default size, an error message is printed and no change is made.

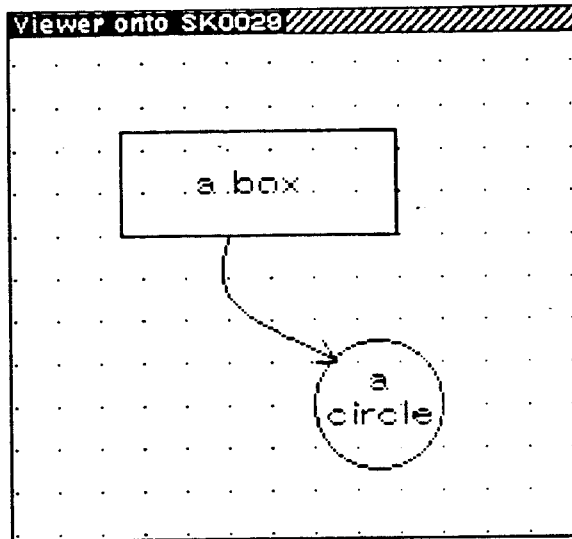
Horizontal alignment — enables the user to change the horizontal alignment property of newly entered text. The choices are:

Center, Left, Right

Vertical alignment — enables the user to change the vertical alignment property of newly entered text. A menu of alignments is displayed from which the user chooses the new default. The choices are:

Top, Center, Baseline, Bottom

Grid — displays and changes the grid which determines where points can be positioned on the display. When the grid is on, only points on the corners of the grid can be specified for any sketch operations. This makes it much easier to align parts of the sketch. The default is to have a grid turned on with a default grid size of about half centimeter. If the grid is off, any point can be selected. The main menu item flips back and forth between using and not using the grid. The grid is always on a power of two in the sketch coordinate space. This means that same points are accessible as the sketch scale changes.



Sketch window with default grid displayed

A submenu of commands that manipulate the grid can be obtained by rolling out the right side of this menu item. The submenu commands are:

Turn grid ON

Turn grid OFF

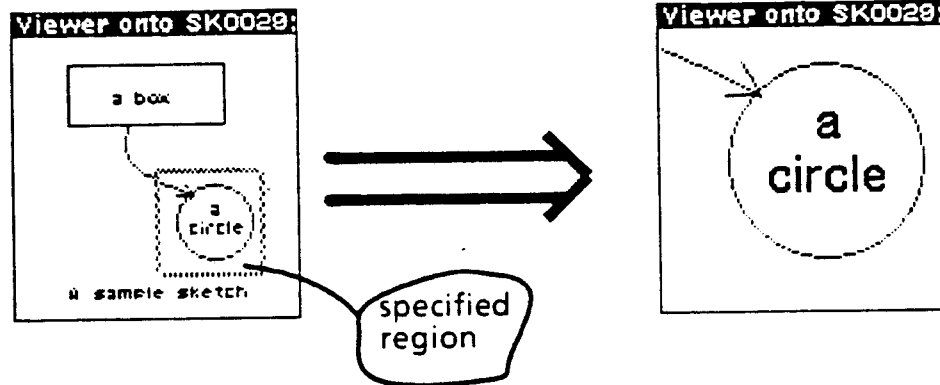
LARGER grid doubles the space between the grid points

smaller grid halves the space between the grid points.

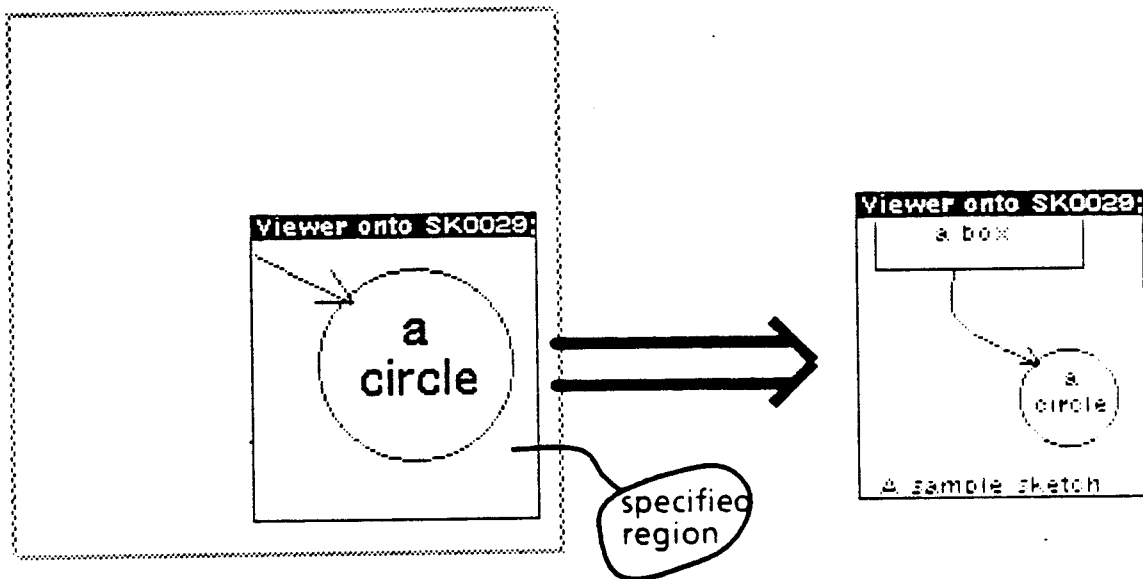
Display grid will display a point at each point on the grid. If the grid points are closer than 3 points together, they are not shown.

Remove grid display will remove the grid display (by redisplaying the window contents). (A fast way of erasing the grid points is by selecting **redisplay** from the menu obtained by *right* buttoning in the title of the sketch window.)

Move view — The user specifies a portion of the sketch that will appear in the window by depressing any mouse button at one corner and sweeping the cursor to the other corner. The specified region will be outlined in gray. When the button is released, the portion of the sketch within the gray box will be scaled to fill the sketch window. The box will maintain the same aspect ratio as the window. The section may be smaller or larger than the current window size. This command can be canceled by typing control-E. Use the subitem **Home** to return the sketch to its original form.



Zooming in with "Move view"



Zooming out with "Move view"

This command is followed by a gray triangle pointing to the right indicating a sub-menu. To retrieve this sub-menu, slide the cursor to the right through the triangle, and select the desired sub-command before releasing the button. The sub-menu allows the user to change other aspect of the way the sketch is viewed.

Zoom — does the same action as **Move view**.

AutoZoom — changes the scale around a selected point. Holding the *left* button down will zoom in toward the cursor. Pressing the *middle* button will zoom out from the cursor. The amount of each scale jump is determined by the global variable AUTOZOOM.FACTOR and is initially .8. Holding either button down will zoom

continuously in the appropriate direction. Use **Home** to return the sketch to its original form. Press any button outside the window to stop autozoom mode.

Home — returns the user to a view at scale of 1.0 which has the origin in the lower left corner. This is useful if you lose the sketch after scrolling or using **Zoom** or **AutoZoom**.

Locator — puts up a small window that displays the cursor's location in sketch coordinates whenever the cursor is inside the sketch window.

New window — opens another window onto the same sketch. The second window can be scrolled or zoomed without affecting the view in the original window. This allows one window to act as an overview while the other can be used to blow up an intricate part of the sketch and make detailed changes. Any changes made to either of the windows are reflected in both windows.

Hardcopy — sends the image in the window to the DEFAULTPRINTINGHOST. **Hardcopy** has a submenu that can change the display to hardcopy mode or makes a printer file of the current view of the sketch in either **Press** or **Interpress**, and, optionally, sends it to the printer. Each of the subcommands have subitems that allow the choice of **Press** or **Interpress**. The subcommands are:

File Only Prompts for a file name and makes an image on it.

Print Only Prints the image but doesn't save the file.

File & Print

Hardcopy Display Displays the text in the window as close as possible to the way it will look when printed.

Normal Display Displays the text in a way more easily read and edited on the screen.

Fix Menu — (appears in the pop up menu only) fixes menu to the right edge of the sketch window.

9.6. Editing Text in a sketch window

Characters within a text string may only be deleted or replaced through the following procedures:

Delete — select after the character to be deleted and use the **bs** (backspace) key.

Replace — select the first character to be replaced with the *left* mouse button and extend the selection with the *middle* button (both buttons if using a two-button mouse). The selected string is shown as white on black. Begin typing the new text. The old text is deleted as the new text is typed.

Characters or words within a string may not be moved or copied. A text string may be moved similarly to other graphic objects by using the **Move** command (see **Sketch Editing Menu**.)

9.7. Putting sketches into text notecards

Sketches can be included in text notecards by copy-selection. Select the location in the text card where the sketch is to go. Hold down the 'copy' key (which for most machines is the shift key). Move the cursor into the sketch card and select the pieces of the sketch to be inserted. The whole sketch can be selected by double clicking on a sketch element. While selecting pieces, the *right* button will remove elements from the selection. When the desired elements have been selected, release the 'copy' key. The sketch will be inserted into the text card. The sketch will be the size of the card from which it is copied. To change its shape, reshape the sketch card. If you want to reshape the sketch card to make it less high than the sketch menu, it is necessary to close the menu (otherwise the card must be at least the height of the menu). To get the menu back after closing it, press the middle button in the title of the sketch card. The copy-selection process can be aborted by move the cursor outside of the sketch card while a button is down.

The sketch will be adjusted so that the lower left corner of the inserted sketch is on a grid point. This is done so the when the sketch is edited, the grid remains aligned with existing points. The grid can be made smaller before the copy insertion to reduce the amount of adjustment.

Note: Current implementation limitation, the text editor will only display the sketch if the text card is at least as tall as the sketch.

A sketch that is in a text card can be edited by pressing the *left* button inside it and selecting the offered menu item 'Edit sketch'. This will prompt for a position of a sketch window the same size as the sketch. In this window, the sketch can be edited, scrolled, zoomed, etc. A sketch so edited may contain elements that are not visible in the printed document. These elements can be exposed by zooming or scrolling. When the window is closed, you will be offered a chance to place the sketch back into the text card. If accepted, the current view of the sketch will be placed into the document. The sketch window will have the menu already fixed beside it. If you want to reshape the window to make it smaller, it is necessary to close the menu (otherwise the window must be shaped to at least the height of the menu).

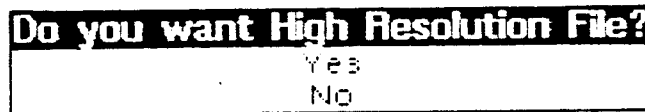
9.8 Copying from one sketch to another

Parts of a sketch can be copied from one sketch to another using a protocol similar to the above. Select the target sketch. Hold down the 'copy' key. Move into the source card and select the desired elements. When the 'copy' key is released, position the cursor back in the target card. You will be dragging an image of the selected elements that can be placed in the target by pressing and releasing the *left* button. For now, the scale of the transferred elements is not changed so the image in the new card may be a different size after the transfer.

9.9. Maps

Some versions of NoteCards have a map capability included in the Sketch NoteCard title bar middle button menu. This capability allows the user to designate a portion of the world and include it as an object in a sketch. Both high and low resolution map files are available. If the map option appears on the sketch menu, a map may be created in a Sketch NoteCard by choosing that option and responding to a sequence of prompts.

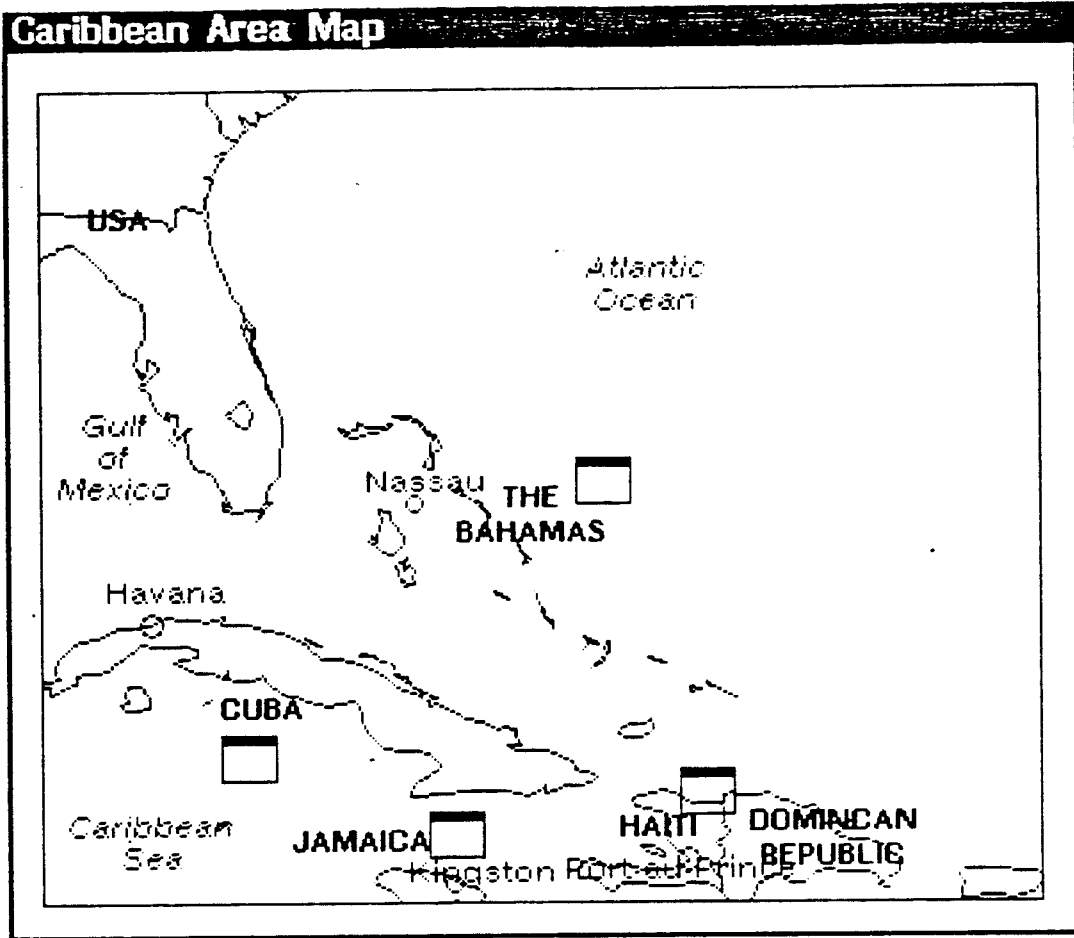
First, the region of the world to be included must be specified. A map of the world will appear on the screen for that purpose when the map option is chosen. Outline the section of the map that is to appear in the sketch card by placing the cursor at the top of a region surrounding this section, depressing the left mouse button, sweeping the cursor across the area and releasing the button. The map of the world closes. A menu will appear asking the user whether the higher resolution map data file should be used in creating the map.



For small areas of the world, this high resolution map data looks significantly better: the low resolution file contains very rough straight line outlines of world boundaries. However, creating a high resolution map display uses considerable computer time before displaying the selected map region in the sketch card.

The user is now required to specify an area in the card where the portion of the map is to be displayed. Place the cursor in the card, depress the left mouse button and sweep the cursor down to indicate the desired area. The area may be smaller or larger than the original map selection. The area designated on the card is constrained by the map region selected: the region maintains the original proportions to avoid creating distortion.

Maps created in sketch cards may occupy all or part of the card and may be annotated with any other sketch element desired. More than one map may be created in a single sketch card. An example of Sketch/Map NoteCard is shown below. Note that it creates a geographic information context by using links to cards describing the various countries shown on the map. It is also annotated with text labels in various fonts.



Example of a Sketch/Map Card

10. Bitmap Editor

The bitmap editor allows the user to manipulate bitmaps that have been inserted in Text or Sketch NoteCards. It is automatically invoked when the bitmap area is selected during the editing process.

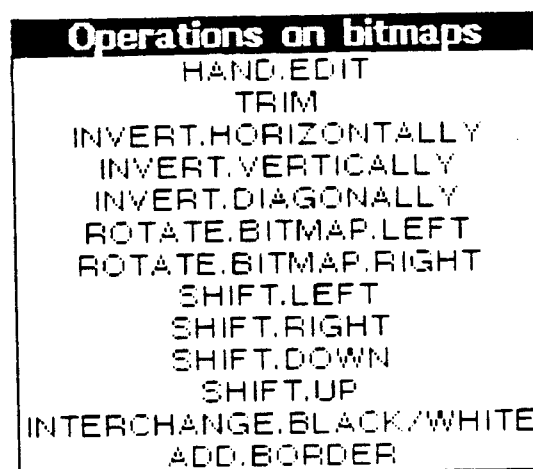
10.1. Inserting Bitmaps into NoteCards

Bitmaps may be inserted into a text NoteCard by holding down the shift key while simultaneously holding down the right mouse button in the screen background. A menu with the single item **ENAP** will appear. Select that item by moving the cursor into the rectangle and releasing the mouse button. The system will prompt for a rectangular region which can be swept out using the left mouse button. The screen area selected will then appear at the type-in point of the active text NoteCard. Moving the cursor into the bitmap region and holding down the right mouse button will bring up the bitmap editing menu described below.

Bitmaps may be inserted into a sketch by holding down the shift key while simultaneously holding down the right mouse button in the screen background. A menu with the single item **ENAP** will appear. Select that item by moving the cursor into the rectangle and releasing the mouse button. The system will prompt for a rectangular region which can be swept out using the left mouse button. A prompt will then appear at the top of the sketch instructing the user to move the figure into place and press the left button. Moving the cursor into the bitmap region and holding down the left mouse button will bring up the bitmap editing menu described below. The bitmap is a sketch element identified by a knot at its corner, so its position may be manipulated with the *Move* command and it may be deleted with the *Delete* command.

10.2. The Bitmap Editor

Moving the cursor into the bitmap region and holding down the right mouse button in a text NoteCard or the left mouse button in a sketch NoteCard will bring up the bitmap editing menu described below.



HAND.EDIT - causes the Interlisp-D bitmap editor to be invoked on the entire bitmap. The Interlisp-D bitmap editor is documented in the Interlisp Reference Manual.

TRIM - trims the white columns and rows from all four edges of the bitmap.

INVERT.HORIZONTALLY - flips the bitmap about its vertical centerline.

INVERT.VERTICALLY - flips the bitmap about its horizontal centerline.

INVERT.DIAGONALLY - flips the bitmap about its X=Y diagonal so that the resulting bitmap's width will be the original bitmap's height.

ROTATE.BITMAP.LEFT - rotates the bitmap by 90 degrees in a counter-clockwise direction so that the resulting bitmap's width will be the original bitmap's height.

ROTATE.BITMAP.RIGHT - rotates the bitmap by 90 degrees in a clockwise direction so that the resulting bitmap's width will be the original bitmap's height.

SHIFT.LEFT - shifts the bitmap a specified number of bits to the left by extending the bitmap by that many bits to the right. The added space is filled in by white. The following number pad appears to allow the number of bits to be specified:

Number of bits to shift the bitmap left:

0	-	clr
	1	2
	4	5
	7	8
	bs	0 ok

This number pad is used much like a simple calculator to enter numbers. The "ok" button returns the number to the system.

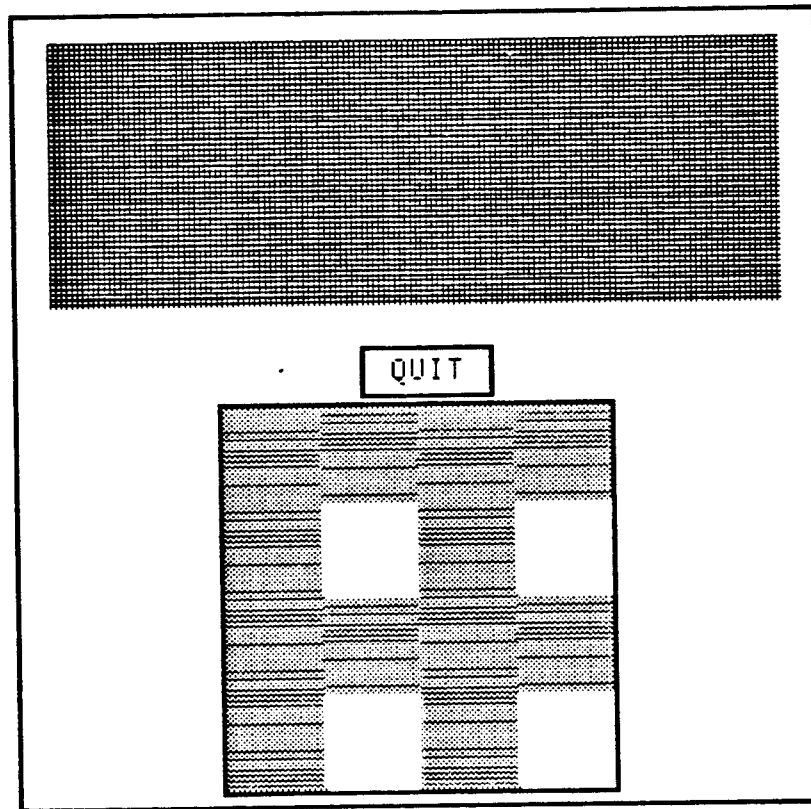
SHIFT.RIGHT - shifts the bitmap a specified number of bits to the right by extending the bitmap by that many bits to the left. The added space is filled in by white. The number of bits is specified by using the number pad shown above.

SHIFT.DOWN - shifts the bitmap a specified number of bits downward by extending the bitmap by that many bits upward. The added space is filled in by white. The number pad shown above appears to allow the number of bits to be specified.

SHIFT.UP - shifts the bitmap a specified number of bits upward by extending the bitmap by that many bits to the downward. The added space is filled in by white. The number pad shown above appears to allow the number of bits to be specified.

INTERCHANGE.BLACKIWHITE - inverts all of the bits in the bitmap (exchanges black for white and white for black).

ADD.BORDER - adds a border to the bitmap by first prompting for the width of the border using the number pad described above, then prompting for the texture of the border with a four bit by four bit bitmap editor. The example below shows the bitmap editor for specifying texture. Note that the area at the top of the window shows what the texture looks like in true screen scale and the bottom area contains a four-by-four array of bits. The quit button is selected with the left mouse button when the desired texture has been specified. That texture will then appear as the border around the bitmap



11. Programmer's Interface to NoteCards

The NoteCards Programmer's Interface is a facility allowing users with programming know-how to interact with NoteCards through an Interlisp-D interface. By using Programmer's Interface functions, a user can create and modify NoteFiles, cards, and links under program control. A detailed description of the functions provided by the NoteCards Programmer's Interface is contained in Appendix A.

The Programmer's Interface functions are divided into seven groups: functions for creating and accessing NoteFiles, functions for creating and accessing the NoteCard type mechanism, functions for creating NoteCards and FileBoxes, functions for accessing NoteCards and FileBoxes, functions for creating and accessing links, functions for creating and accessing link labels, and various utility functions.

11.1. NoteFile Functions

The Programmer's Interface contains a set of functions for NoteFile creation and access; these functions cover many of the operations provided by the Main Menu NoteFile Ops selection. Included are functions to create, open, close, repair, compact, and delete a NoteFile; to back up and restore a NoteFile from floppy disk; and to return the current NoteFile stream and the full name of the currently active NoteFile. For example, to open a NoteFile named *DEMO.NOTEFILE* on the connected directory, a user may invoke the following function (interaction is assumed to be from the Top level typescript window):

```
←(NCP.OpenNoteFile 'DEMO)
{STREAM}#7,1000
```

11.2. NoteCard Type Functions

The Programmer's Interface contains a set of functions to give the user access to the NoteCard user-defined types facility. For an explanation of this mechanism, see Appendix B. Global functions allow the programmer to find out what all of the existing NoteCard types and NoteCard substance types are and check the validity of a given card or substance type. For example, calling the NoteCard types function before the user has defined any card types will return the list of system-defined card types and calling the NoteCard substance types function before the user has defined any substance types will return the list of system-defined substance types as follows (interaction is assumed to be from the Top level typescript window):

```
←(NCP.CardTypes)
(Text Graph Sketch FileBox Browser Search LinkIndex Document)
←(NCP.SubstanceTypes)
(SKETCH GRAPH TEXT)
```

Corresponding functions allow the programmer to define new NoteCard types and NoteCard substance types.

11.3. NoteCard and FileBox Creation Functions

The Programmer's Interface contains a set of functions to allow the user to create various sorts of cards and boxes within the currently open NoteFile. Functions correspond to the system-defined card types on the middle-button Main Menu Create Option and provide a mechanism for creating card types defined by the user. For example, to create a text notecard called *A Sample Text NoteCard* without displaying the card on the screen or filing it in a parent filebox, the following function may be invoked, returning the card's ID number (interaction is assumed to be from the Top level typescript window):

```
←(NCP.CreateTextCard "A Sample Text NoteCard" T)
NC00162
```

11.4. NoteCard and FileBox Access Functions

The Programmer's Interface contains a set of functions to provide access to cards and boxes which exist in the currently open NoteFile. The cards and boxes may be manipulated whether they're on or off the screen. If the card or box is currently being displayed, the changes made will cause an automatic update to the screen. Thus, users can switch between program-driven and screen-interface-driven interaction modes at any time

Cards are either active or inactive at any given time; all cards visible on the screen are active. An active card has its information cached (on its property list) thus saving time at the expense of using up memory. Most of the access functions leave the card in the same state as it was when the function was invoked; thus, if a user wishes to perform several consecutive operations on a card, the card should probably be made active by calling *NCP.ActivateCards*. When the series of operations have been completed, *NCP.DeactivateCards* will reclaim the memory space by uncaching the card.

An example of how the Programmer's Interface card access functions may be used to activate a card, add a property to its property list, add some text to the body of the card, file it in the *Table of Contents* filebox, and deactivate it follows (interaction is assumed to be from the Top level typescript window):

```
←(NCP.ActivateCards 'NC00162)
NC00162
←(NCP.CardProp 'NC00162 'Certainty 0.5)
NIL
←(NCP.CardAddText 'NC00162 "Some text for this card" NIL)
NIL
←(NCP.FileCards 'NC00162 (NCP.GetContentsFileBox))
NC00001
←(NCP.DeactivateCards 'NC00162)
NC00162
```

11.5. Link Creation and Access Functions

The Programmer's Interface contains a set of functions to create links and provide access to links present in the currently open NoteFile. Links may be manipulated whether the cards they are connecting are on or off the screen. An example of how the Programmer's Interface link creation and access functions may be used to create a link with the label *Example* between two existing cards and change its display mode to both destination title and label follows (interaction is assumed to be from the Top level typescript window):

```
←(NCP.LocalGlobalLink 'Example 'NC00162 'NC00163 'START)
(387 NC00162 NC00163 NIL Example Icon)
←(NCP.LinkDisplayMode IT 'Both)
Icon
```

11.6. Link Label Creation and Access Functions

The Programmer's Interface contains a set of functions to manipulate link labels in the currently open NoteFile. An example of how the Programmer's Interface link label manipulation functions may be used to get a list of all of the link labels in the system, including system-defined labels, follows (interaction is assumed to be from the Top level typescript window):

```
←(NCP.GetLinkLabels)
(Source ListContents BrowserContents FiledCard SubBox DocBackPtr
LinkIndexBackPtr Explanation Example Comment Remark Question Rebuttal
Argument Unspecified)
```

11.7. Utility Functions

The Programmer's Interface contains a set of utility functions to provide a convenient way of accessing some of the miscellaneous NoteCards operations within the currently open NoteFile. For example, a Programmer's Interface function may be used to perform a property search for the property *Certainty* to find all of the cards with that property (interaction is assumed to be from the Top level typescript window):

```
←(NCP.PropSearch 'Certainty)
(NC00088 NC00162)
```

12. Recovering From System Bugs

A system bug is not a result of incorrect input by the user. System bugs cause occasional problems in the behavior of the system and often can be attributed to a bug in a particular user-initiated function. Since NoteCards is a research prototype, sporadic problems may occur, and it is advisable to be aware of some of these sorts of problems and to know what to do when they occur.

Because it is helpful to the designers of this system to know of any errors that occur, we request that a log be kept of problems encountered. Some common problems and their recoveries are discussed in this section, as well as preventive measures, and the relative severity of various bugs.

Error recovery may be approached at several different levels, depending on the programming knowledge of the user. If a user has some programming experience, often the Programmer's Interface offers useful functions for error recovery. This section addresses both simple methods for coping with errors as well as more complex procedures which may be useful to the programming user.

12.1. Break Window

Occasionally a break window may pop up displaying an error message such as *Unknown Card Type*. Write down what occurred before the window popped up, including the message displayed in the break window. Place the cursor in the body of the window, depress the *middle* mouse button (depress both buttons if using a two-button mouse) and select "↑" (which may be the left double quote on some keyboards) to close the window. This operation must be performed before proceeding with the normal input. If a break window appears when a card or link is being created, it is advisable to delete that card or link and begin the creation process over again.

The following screen image shows a typical break window, which in this case occurred when a user tried to bring up the *To Be Filed* box from a damaged NoteFile database:

```
NC.GetNoteCard break: 1
NC000003 Error in Database file -- incorrect item identif
ier.
NIL

("NC.GetNoteCard" broken)
96:
```

In the above example, since a system-maintained portion of the NoteFile damaged, the user should probably select "↑" in the break window as described above, then close the NoteFile and initiate the *Repair NoteFile* operation from the NoteFile Ops menu.

12.2. Types of Bugs, Their Severity, and Suggested Fixes

The bugs that may be encountered while using NoteCards vary in their symptoms, severity, and potential for recovery. Some errors may just be ignored or easily repaired by a simple procedure; others require that the user install a fresh version of the NoteCards sysout; while still others are best handled by reverting to the newest backup of the NoteFile involved in the problem.

The simplest type of error involves the display of notecards on the screen. These errors are not severe, and are often very easy to correct. Sometimes text in Text NoteCards and FileBoxes is not displayed properly. If this occurs, scrolling the card up or down usually corrects the display (see Section 2.4. Scrolling). If this doesn't work, select *Restart Editor* from the Text Editor Menu (see Section 7.4. Text Editor Menu: *Restart Editor*). The Redisplay option on the window menu (see Section 7.4. Window Menu: *Redisplay*) will often clear up odd looking Sketch NoteCard displays.

A second type of display-related error is caused by missing fonts. For example, a break window will appear that looks like the one shown below. This type of error usually occurs when the user is trying to bring up a card from the NoteFile.

```

FONT NOT FOUND: (TIMESROMAN 7 ...) ERROR break:1
FONT NOT FOUND
(TIMESROMAN 7 (MEDIUM REGULAR REGULAR) @ DISPLAY)

(ERROR broken)
16:
    
```

ERRORSET
BREAK1
ERROR
FONTCREATE
EVAL
LISPM
ERRORSET
EVALOT
ERRORSET
T

This break window was caused by the system's inability to find the display font TimesRoman 7. The easiest way of correcting this situation is to type an ↑ in the break window (as described above in Section 12.1.), put the floppy disk containing the display fonts into the floppy disk drive (this floppy disk is part of the Interlisp-D release), and try again. If the break occurs again, substituting a font that does exist (using the Interlisp-D *SETFONTDESCRIPTOR* and *FONTCREATE* functions) may solve the problem. For example, for the break window shown above, the user may substitute TimesRoman 8 for TimesRoman 7 by typing the following:

```

FONT NOT FOUND: (TIMESROMAN 7 ...) ERROR break:1
FONT NOT FOUND
(TIMESROMAN 7 (MEDIUM REGULAR REGULAR) @ DISPLAY)

(ERROR broken)
16: (SETFONTDESCRIPTOR 'TIMESROMAN 7 'MRR @
'DISPLAY (FONTCREATE 'TIMESROMAN 8 'MRR @ 'DISPLAY))
{FONTDESCRIPTOR}#13,61566
17:
    
```

ERRORSET
BREAK1
ERROR
FONTCREATE
EVAL
LISPM
ERRORSET
EVALOT
ERRORSET
T

The successful return of a value from that function indicates that it is alright to continue. To complete the action originally initiated, the value of the new font must be returned; to do this, type

RETURN IT

followed by a carriage return, and the interrupted process should continue. If this procedure doesn't work, then it may be necessary to delete the card through the Programmer's Interface and recreate it, using the procedure described later in this section.

A second type of error occurs when a card is being created. Sometimes this is symptomatic of another, deeper error, but usually it may be corrected by simply deleting the card, either on the screen, or through the Programmer's Interface, and trying again.

A more severe type of error occurs when the NoteCards sysout has been damaged. The symptoms of this type of error are break windows with messages such as *Arrays Full* or *Stack Overflow*. Sometimes the system may even refuse to respond at this point. The best course of action to take if the system is still responding is to ↑ out of the break window, close any open NoteFile without trying to do any more work on it, logging out, and installing a fresh NoteCards sysout, a procedure described in the Installation Guide. If the system crashes during this process, and the NoteFile has not been closed, it is advisable to perform a Repair on the NoteFile after the fresh sysout has been installed.

The most severe type of error involves the Dandelion local disk. This error is actually an operating system level error, and is difficult to correct. The symptom characterizing this type of error is a break window occurring during card access with a message that End of File has been encountered. This usually means that the open NoteFile has been truncated. The NoteFile should be closed and a repair attempted. If the repair is unsuccessful, the NoteFile should probably be restored from the last backup, and if the user feels that significant work has been lost, the procedure to scavenge NoteFile contents (described later in this section) may be followed.

12.3. Preventive Measures

There are several kinds of preventive measures which may be taken to reduce the damage done when a system error is encountered. One of the best methods of protecting the contents of a NoteFile is to perform regular backups of important NoteFiles. The Backup To Floppy command on the NoteCards Main Menu allows a user to make a copy of a specified NoteFile on a floppy. (See Section 5. Main Menu: *Backup To Floppy*.) If frequent backups are performed, recovery from severe system crashes may include restoring the most recent backup using the NoteCards Main Menu command Restore From Floppy. (See Section 5. Main Menu: *Restore From Floppy*.) Backups may also be performed using the Interlisp-D *COPYFILE* function to make a second copy of the NoteFile on a file server (if available), a formatted floppy, or the local disk. For example, if a local disk backup copy of a NoteFile called WorkingNotes is desired, the following command may be typed in a Lisp Exec window:

```
(COPYFILE '{DSK}<LISPFILERS>WORKINGNOTES.NOTEFILE
          '{DSK}<LISPFILERS>SAVEDNOTES.NOTEFILE)
```

The *Repair NoteFile* command may be helpful in repairing the NoteFile when there are inconsistencies or problem areas in the actual NoteFile. Select this command from the Main Menu. (See Section 5. Main Menu: *Repair NoteFile*.) Using the Repair NoteFile and the Compact NoteFile commands

periodically ensures that the NoteFile is consistent and that any anomalies in the link structure or notecards are uncovered and fixed. If a NoteFile does not successfully complete a Repair process, it is advisable to return to the previous consistent version of the NoteFile and follow the suggestions for rebuilding a NoteFile given later in this section.

If a break window occurs when a card is being created or if a card starts displaying eccentric behavior during interaction, deleting the questionable card will often prevent compounded problems in the future. The contents of a card may be saved and reused in a new card as discussed later in this section. If a card cannot be brought up onto the screen to delete it, the Programmer's Interface function, *NCP.DeleteCards*, may be useful in removing the suspect card from the NoteFile.

12.4. An Example of Error Recovery Using the Programmer's Interface

The Programmer's Interface functions can be extremely useful in error recovery. For example, suppose the culprit card is a damaged text card, and in fact cannot be brought up onto the screen for normal menu-based interaction. Further, assume, the title is known to be *Sick NoteCard*. Starting with this knowledge, a procedure you might follow is to first identify the card by internal ID, then check the type of the card to make sure it's a text card, find all of its links (note and delete them as necessary - you may wish to examine them), extract the unformatted text from the damaged card, put the saved text in another text card, and delete the damaged card. If this procedure were performed in the Lisp Exec, the dialog between system and user would appear as follows (comments are in italics):

```
(* Identify the card by number)
←(NCP.TitleSearch "Sick")
(NC00164)

(* Check card type)
←(NCP.CardType 'NC00164)
Text

(* Find out which cards point to it, including FileBoxes)
←(NCP.GetLinks NIL 'NC00164 NIL)
((388 NC00022 NC00164 NIL FiledCard Title))

(* Delete it from its FileBox)
←(NCP.DeleteLinks IT)
NIL

(* Find out which cards it points to and save the links)
←(SETQ SavedLink (NCP.GetLinks 'NC00164 NIL NIL))
((389 NC00164 NC00031 NIL Unspecified Icon))

(* Take a look at the destination card and note the link)
←(NCP.BringUpCard 'NC00031)
{WINDOW}#11,132226
```

(Remove the link)*
 ←(NCP.DeleteLinks SavedLink)
 NIL

(Grab the text and put it in a new text card)*
 ←(NCP.CardAddText (NCP.CreateTextCard "Contents of Damaged Card" T) (COERCETEXTOBJ
 (NCP.CardSubstance 'NC00164) 'STRINGP))
 NC00165

(Delete the damaged card)*
 ←(NCP.DeleteCards 'NC00164)
 NIL

12.5. Scavenging a Broken NoteFile

In some of the situations described earlier in this section, particularly those in which a Repair fails, it is advisable to return to an earlier version of the NoteFile saved in a backup. In these cases, the user may wish to attempt to save as much as possible of the work involved with the card or cards that have been created in the NoteFile since the last backup.

One way of performing this type of scavenge when a backup has been performed recently is to:

- 1) Note the highest card number in the backup version (opening or closing the file provides that information).
- 2) Open the damaged NoteFile (a break window may appear during the open operation)
- 3) Open a TEDIT window by selecting TEDIT off the background menu using the right mouse button.
- 4) For each card ID number after the highest card number in the backup version, evaluate *(NCP.CardType <ID>)*.
- 5) If the card exists, evaluate *(NCP.BringUpCard ID)*
- 6) If a card is of type Text or Sketch, copy the contents to the open TEDIT window by using the shift-select process (described in Section 7.1. Selecting Text and in Section 9.7. Putting Sketches into Text NoteCards). You may also wish to note the title. Link icons should be excluded from the copy, since TEDIT will not understand them.

- 7) If a card is a Graph, FileBox, or Browser, then its contents are more difficult to save. It is perhaps easiest to make hardcopies or *SNAPs* of these to capture as much of the structural and organizational information as possible and use these hardcopies as reminding tools when the NoteFile is being rebuilt.
- 8) When as much of the new information as possible has been scavenged from the damaged notefile, close the notefile, and if desired, write the TEDIT file to disk using the Put command from the TEDIT window's left button title bar menu.
- 9) Delete or Rename the damaged notefile so there is no confusion later.
- 10) Restore the most recent backup to the local disk
- 11) Open the new notefile and begin rebuilding, copying text and sketches from the TEDIT file and using hardcopies of browsers, fileboxes and graphs as a guide to recreating the structure.

Sometimes it is necessary to do a more complete type of scavenge because there is no recent backup. In this case, creating a document starting with the highest level intact node is the most efficient way of gathering text. It may be necessary to create several documents to avoid the broken portion of the NoteFile. Sometimes the *Table of Contents* card may be used as the starting point for document creation if, for example, *To Be Filed* is the damaged segment of the NoteFile. Copy the document's (or documents') contents to a TEDIT window and follow the procedure explained above. Sketches must be treated as they are in the previous description since they are not automatically dumped to document cards.

13. NoteCards Release 1.2k Description

This section updates the NoteCards Release 1.1 User's Manual, describing changes and new features for Release 1.2k. The Intermezzo release of Interlisp-D is the basis for NoteCards Release 1.2k. Future releases will use the letter suffix following the release number to indicate the appropriate version of Interlisp-D.

Changes from 1.1 are mostly in the following areas: the NoteCards browser, notefiles interface, link icon display and user interface. In addition, there are various miscellaneous changes, a couple of new card types, and fixes of several outstanding 1.1 bugs.

Important documentation for NoteCards Release 1.2k is included in the appendices to this manual. Included are documents describing the notefile inspector (Appendix D), how a notefile is organized (Appendix C), the types mechanism (Appendix B), and the programmer's interface (Appendix A).

13.1. Notefile Maintenance and Recovery Operations

Checkpoint Session and Abort Session

A fundamental change was made to the way Notecards updates its working notefile that provides a checkpointing facility in Release 1.2k. Users may checkpoint their work (thus saving changes without closing), abort a session (losing work since the last checkpoint), and recover more gracefully from crashes. The current structure of NoteCards notefiles is discussed below to provide a background for the description of the new functionality. A more detailed explanation may be found in the document entitled "An Introduction to the Internal Organization of Notefiles" in Appendix C.

A notefile consists of two parts, an index area and a data area. The index includes for each notecard, several pointers into the data area. There are separate pointers for the notecard's substance, title, prop list, and links. When, say, a notecard's title is changed, the new title is written at the end of the data area (in fact the end of the file) and the index pointer is changed. In Release 1.1 (and earlier), the index modifications happened out on the file as they occurred. Now, in Release 1.2k they happen in an in-core array and are not written to the file until checkpoint (or close) time. In addition, there is a checkpoint pointer that points to the end of file at the time of the last checkpoint or close. New data (such as a new title) is still written to the file, but always at the end of the file. Thus if a crash occurs and later the notefile is reopened, Notecards can notice the extra data beyond the checkpoint pointer and truncate the file at that point (if you confirm).

Two new NoteFile Ops menu entries *Checkpoint Session* and *Abort Session* implement this new functionality. Checkpointing causes any active cards to have their contents saved to the notefile, the index array to be written back out to the file, and the checkpoint pointer to be reset to the end of the file. Thus checkpointing provides a mechanism for saving changes to a notefile without clearing the screen and closing the notefile. Closing a notefile also causes the system to automatically perform a Checkpoint Session; however, the screen is cleared after the checkpointing process is completed. Aborting a session closes the current notefile, discarding all work since the last checkpoint or close.

When a notefile is opened, the checkpoint pointer is compared with the end of file pointer. If they don't agree, then a message is printed out to that effect giving the number of bytes beyond the checkpoint pointer. You must then choose one of three options from a menu: "Abort," "Inspect & Repair," and "Truncate File." Choosing "Abort" causes the open notefile operation to be aborted. Choosing "Inspect & Repair" causes the notefile inspector to be called on your notefile. This is an easy (but slow) way to incorporate these post-checkpoint changes into the notefile. (See the description below of the inspector and the documentation in Appendix D.)

Finally, you have the option of truncating your file. This should only be done if you feel that the post-checkpoint work is not worth saving. You are then given the option of saving the extra work since the last checkpoint to a file. The information so saved is mildly accessible in the event you change your mind and want to recover some of the post-checkpoint work. In that case, you should open the truncated notefile and bring up a separate TEdit window on the file containing the truncated information. Though TEdit formatting information is lost, you can recover a card's text by browsing this file. (Note that scrolling from back to front will retrieve the most recent version of each card.)

[Note that closing (or saving without closing) a card writes it out to the file, but does not force the index to be updated. Thus, if crashes are anticipated, do CheckpointSession often.]

Compact Notefile

Because Notecards never actually overwrites any information in the data area of a notefile, it is necessary to periodically compact the notefile. This facility has been improved in Release 1.2k in two ways. It is now possible to specify a target file name for the compaction (rather than always going to the same name), and it is now possible to compact a notefile in place. These two choices form a submenu of the CompactNotefile entry in the Notefile Ops menu.

Compacting in place means that no new copy of your notefile is created, rather the contents of the old file are rearranged in such a way that old versions of cards are written over. This is handy when your notefile is large and there isn't room to store another version. But be aware that you'll not be able to back up cards to previous versions (using the notefile inspector) on such a notefile. We recommend compacting to a target file and saving the uncompactd version until you can be sure that old versions of cards are not needed.

Compacting to a target file asks for names of the source notefile and destination file. Specifying the same name for the destination as for the source will cause a new version of the file name to be written. (This is the old 1.1 functionality.)

Copy NoteFile (Copying, Restoring, and Backing up Notefiles)

The menu entries for *RestoreFromFloppy* and *BackupToFloppy* have been removed from the NotefileOps menu. In their place is a general *CopyNotefile* option. It prompts you for source and target file names for the copy.

There is a new facility for checking in and out notefiles using locks so multiple users may safely share a notefile without overwriting each others changes. Still in the experimental stages, it must be called via the programmer's interface. The programmer's interface documentation in Appendix A describes this option.

Inspect&Repair Notefile

The old Repair option on the Notefile Ops menu is now called *Inspect&Repair NoteFile* and has been improved considerably. Before rebuilding the links of your notefile, it reads the entire data area looking for good card parts (including outdated and deleted versions). It then allows you to delete and/or back up card parts to previous versions. All this is done interactively through a menu driven interface. Only when the notefile is deemed healthy are you allowed to perform the link rebuilding. For details on the operation of *Inspect&Repair NoteFile*, see Appendix D.

13.2. Changes to the Notecards user interface.

Stylesheets:

Several menu sequences in NoteCards have been modified to use stylesheet package for user interaction, in particular, changing a link's display mode, a browser's specs, or the default text and link icon fonts from the global parameters menu. Stylesheets allow packaging of several menus together with "buttons" governing individual menus and the stylesheet as a whole. Note that the stylesheet in Figure 13.2-1 represents what was originally a sequence of three pop-up menus: stylesheets present this set of choices in a single context.

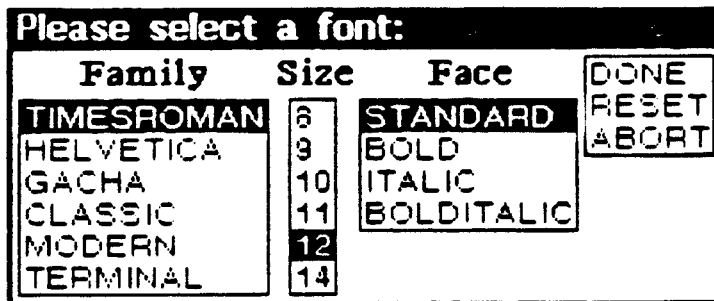


Figure 13.2-1 Sample Stylesheet for Changing a Font

Menus within a stylesheet can optionally allow multiple selections. All stylesheets have three global buttons *DONE*, *RESET*, and *ABORT*. *DONE* causes the new values to be accepted. *RESET* causes the original values (when the stylesheet was entered) to be recovered. *ABORT* exits the stylesheet without changing any values. Menus allowing multiple selections also have the buttons *ALL* and *CLEAR* attached. *ALL* causes all values in the menu to be selected while *CLEAR* unselects the entire menu. Toggling of menu entries is accomplished by left clicking the entry.

New Global Parameters:

The Release 1.2k top level global parameters menu reflects an ability to control some new aspects of functionality, as illustrated in Figure 13.2-2. These new parameters are described below; parameters available in Release 1.1 (DefaultCardType, FixedTopLevelMenu, ShortWindowMenus, ForceSources, ForceFiling, ForceTitles, MarkersInFileBoxes, AttachBitmapsToLinkIcons, LinkDashingInBrowsers, SpecialBrowserSpecs, and AnnoAccessible) are described in Section 5. To change the value of a global parameter, a user may click on the variable name. The value will toggle between "Yes" and "No" if binary, and allow selection from an appropriate menu otherwise.

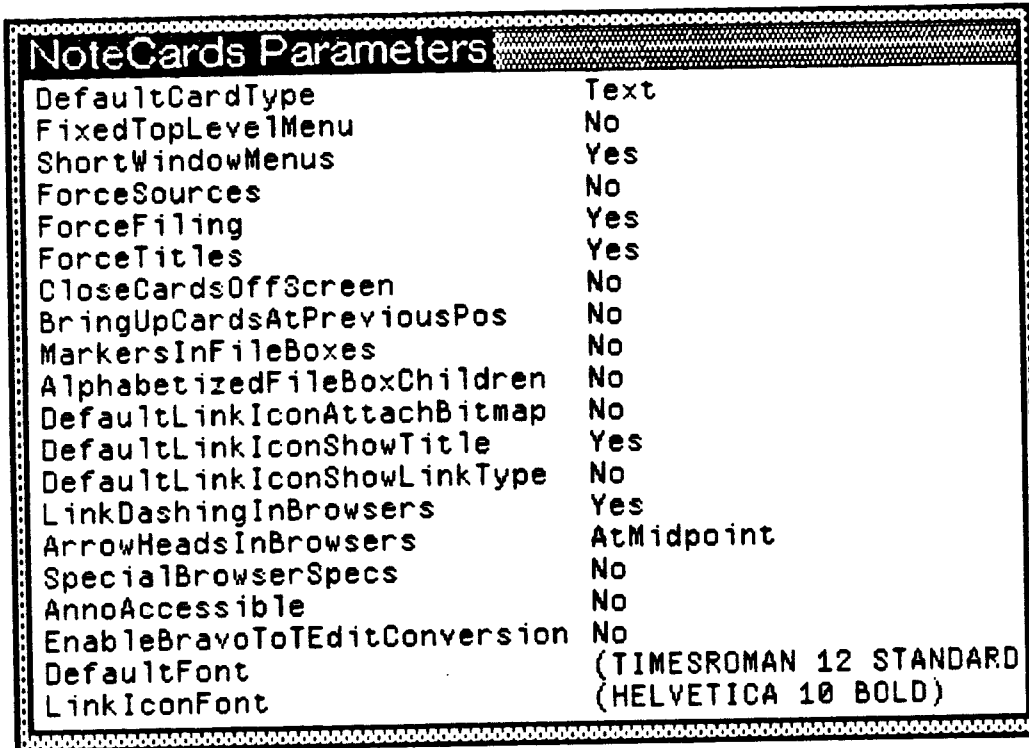


Figure 13.2-2: The Global Parameters menu.

ForceSources, ForceFiling, ForceTitles: These dictate whether to bother you at card closing time about incomplete information for the card. If ForceFiling is set, for example, then you are asked to designate parent fileboxes of the card before closing. Similarly, for sources and titles. If ForceFiling is off (value is "No"), then cards without parents will be filed automatically in the ToBeFiled filebox at closing time. If ForceTitles is off, then an untitled card will be left with the title "Untitled." ForceTitles and ForceFiling default to "Yes," while ForceSources defaults to "No." See Section 5 for additional description of these parameters.

CloseCardsOffScreen controls whether or not a card is closed invisibly; if this parameter's value is *Yes*, then when a card is closed, it is given an off-display screen position so that the close happens invisibly from a user's perspective.

BringUpCardsAtPreviousPos establishes whether a user wishes to specify a new screen position each time a card is brought up on the screen. If its value is *Yes*, then when a card is brought up on the screen, a user is not asked to position a frame image. Rather, the card is brought up in the position it occupied at the time of last closing.

MarkersInFileBoxes: If "Yes," then new fileboxes will contain the markers *FILE BOXES* and *NOTE CARDS*. New child boxes are inserted under the *FILE BOXES* marker and new child cards under the *NOTE CARDS* marker. If "No," then new fileboxes come up without markers and new children are inserted at the current cursor position (unless the filebox has an *OrderingFn* - see Section 13.4). Note that regardless of the *MarkersInFileBoxes* setting, if a filebox has no markers (because you've deleted them) then new children are inserted at the cursor position (or according to the *OrderingFn*).

AlphabetizedFileBoxChildren: If "Yes," then new fileboxes will have the property *OrderingFn* set to *NC.IDAlphaOrder*. This will cause any new cards put into such a filebox to be inserted in alphabetical order. For further details on *OrderingFn*'s for fileboxes see Section 13.4.

DefaultLinkIconAttachBitmap, **DefaultLinkIconShowTitle**, **DefaultLinkIconShowLinkType**: These parameters dictate the manner in which link icons are displayed if not currently specified in the icon. There are three fields of a link's display mode that can be set, unset, or floated independently. If a field is floated, then the global parameter for that field is consulted. For example, if a link icon's display mode has value *FLOAT* for the *ShowTitle* field, then whether the title gets shown inside the link icon depends on the value of *DefaultLinkIconShowTitle*. See below for a further description of a link's display mode.

LinkDashingInBrowsers: If "Yes," then browser links are drawn with dashed lines with the dashing style corresponding to the link's type. See Section 13.3 for further details on browser changes. Defaults to "No."

ArrowHeadsInBrowsers: This dictates whether arrow heads are drawn on browser links. The variable can be set to either "AtMidpoint," "AtEndpoint," or "None." See Section 3 for details. Defaults to "None."

EnableBravoToTEditConversion: If "Yes" then TEdit checks when getting a file whether that file is in Bravo format and if so, converts. This defaults to "No" for efficiency.

DefaultFont: This dictates the font that new text cards default to.

LinkIconFont: This dictates the font for text appearing in link icons.

Link Icon Display Mode:

The display mode of a link icon can be changed by middle buttoning in the icon, choosing *Change Display Mode*, and selecting from the three menus in the resulting stylesheet. The stylesheet for changing the link icon display mode is shown in Figure 13.2-3. The three menus incorporated in the stylesheet are: *AttachBitmap*, *ShowTitle*, and *ShowLinkType*. A *Yes* value for *AttachBitmap* causes link icons to be shown with a bitmap representing the type of the destination card attached at the left. *Yes* values for *ShowTitle* and *ShowLinkType* cause the link icon to contain the title of the destination card and/or the link type. Any of the three fields can have the value *FLOAT*, in which case the appropriate global parameter will be consulted as discussed above. If all fields are set to *No* (or the floating ones inherit *No* from the global parameters), then a small, uninformative icon (U) is used to display the link.

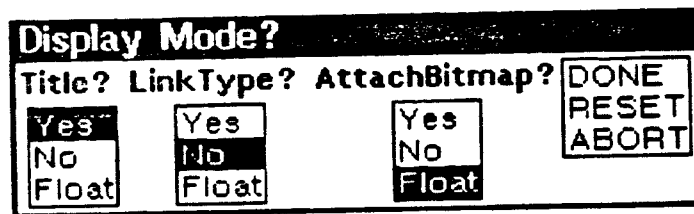


Figure 13.2-3 Link Icon Display Mode Stylesheet

"Pushing" and "Pulling" Link Icons:

It is possible to add links to a text card without having to select *InsertLink* from the title bar menu. This is done by copying or moving a link icon from one card to another. The two styles of link icon moving are called "pulling" and "pushing." When pulling, you must first select the place to move to, and then the thing to move. When pushing, this is reversed. The thing to move is selected first followed by its new home.

Pulling works like TEdit shift-select. That is, to move an icon, put the cursor where you want to move to and hold down the shift key (or shift and ctrl keys) while left clicking in the left or right quarter of the icon.

Pushing is done by holding down the shift key while left clicking in the icon. Then move the cross-hairs cursor to the icon's new home and left-click. To abort a "push," just left click in the background. Note that "pushing" currently only works for copying, not moving.

Specifying Notefile Names and Card Titles:

A different editor has been incorporated into Notecards for obtaining card titles, file names, etc. This editor is the same one used in the top level lisp exec window (TTYIN). Thus you can change the title (or file name) given as prompt via mouse edits.

13.3. Changes to the Notecards Browser.

Figure 13.3-1 shows a typical browser. Note especially the arrowheads on links, the multiple link dashing styles, and the new browser editor menu. Each of these, along with other features, is described below.

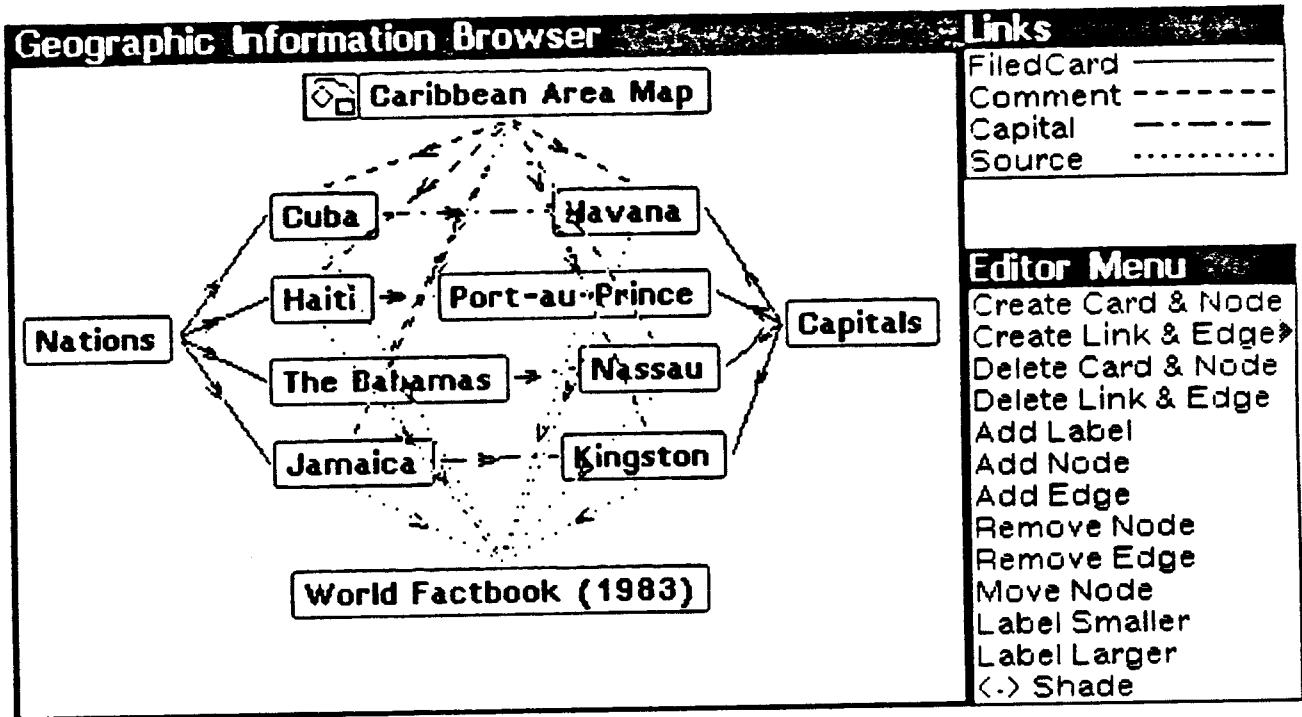


Figure 13.3-1: A NoteCards browser.

Multiple roots:

Browsers can now contain multiple roots, in which case the graph will be laid out as a forest. For example, the browser shown in Figure 13.3-1 has three roots, *Nations*, *Caribbean Area Map*, and *Capitals*. The roots are either selected when the browser is created, or added with *Add Node*.

Dashed links:

Dashed browser links was a rarely used option in Release 1.1, largely because of speed considerations. The speed of drawing dashed links has improved in Release 1.2k by taking advantage of improvements in Grapher. There are currently nine different dashing styles possible. If a browser contains instances of more than nine different link types, then the last dashing style will be used repeatedly for each link type beyond the ninth. As before, link dashing is a user-settable option in the GlobalParameters menu (see Section 13.2 and Section 5). In Figure 13.3-1 the links legend is attached to the upper right corner of the browser and shows four dashing styles: _____ to represent FiledCard links, ----- to represent Comment links, ----- to represent Capital links, and to represent Source links.

Arrowheads:

Arrowheads can now be drawn on browser links. These show the direction of the notecards link being represented in the browser. This is a user-controlled parameter in the Global Parameters menu with possible values *AtMidpoint*, *AtEndpoint*, or *None*. If *AtMidpoint* or *AtEndpoint* is specified, then arrowheads will be drawn at link midpoints or endpoints, respectively. However, in either case, if two browser nodes are connected by more than one link, then any arrowheads for those links will appear at the midpoints (so as not to overlap). The browser example shown above has the *ArrowHeadsInBrowser* global parameter set to *AtMidpoint*.

Browser Specs:

Whereas in Release 1.1 only the link types to traverse could be specified, in Release 1.2k, link types is one of a number of browser specs. Also included are browser depth, format, and orientation. These are accessible through a BrowserSpecs stylesheet, a collection of 5 menus. The browser specs stylesheet is shown in Figure 13.3-2. For general details on the stylesheet interface see Section 13.2. In this case, the forward and backward link types menus are multi-selectable, that is, more than one entry can be chosen. The other three menus are used to make single selections.

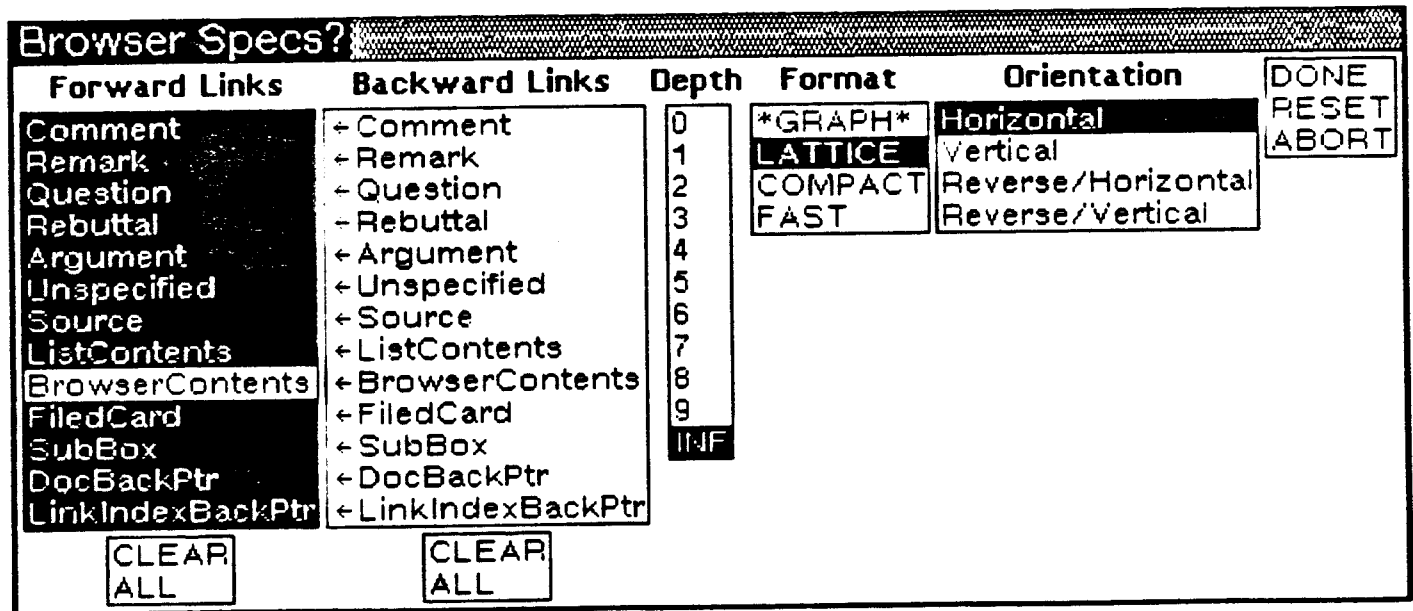


Figure 13.3-2 The Browser Specs Stylesheet.

Forward and backward link types function as in Release 1.1. That is, the browser will contain only nodes for cards reachable from the root cards by following forward links in "line of direction" or backward links in "reverse line of direction." A description of how links work may be found in Section 4.

Browser depth is chosen from a menu containing entries for the integers 0 through 9 and INF (or infinite depth). The default is INF, meaning that the browser will not be cut off until there are no more

links to follow from leaf nodes. Choosing depth 0 means that only the root nodes will appear (and no links).

Browser format is one of **GRAPH**, *LATTICE*, *COMPACT*, or *FAST*. The latter three are provided by the grapher package and correspond to lattice, compact forest and fast forest, respectively. *COMPACT* and *FAST* generate virtual nodes (in double boxes) whenever two or more links would be drawn to the same node. *LATTICE* only generates virtual nodes when a cycle exists in the graph. **GRAPH** is a new format that never generates virtual nodes. The drawback to using **GRAPH** is that a cycle can cause lines to be drawn that cross boxes or overlap other lines. Thus you may have to move nodes around for legibility after computing the browser. The example shown in Figure 13.3-1 uses the default value, *LATTICE*.

Browser orientation is one of *Horizontal*, *Vertical*, *Reverse/Horizontal*, or *Reverse/Vertical*. These specify whether the graph is laid out left-to-right, top-to-bottom, right-to-left, or bottom-to-top, respectively. The default is *Horizontal*.

New Middle Button Title Bar Menu Options:

Several new entries have been added to the middle button menu invoked from a browser's title bar. The options are illustrated by the menu below:

Recompute Browser
Relayout Graph
Reconnect Nodes
Unconnect Nodes
Expand Browser Node
Graph Edit Menu
Change Browser Specs

Recompute Browser causes the current contents of the browser to be thrown away and recomputed as in Release 1.1. However, in Release 1.2k, you can optionally specify a new set of root nodes.

Relayout Graph does not rebuild the graph, but rather causes the nodes and links of the graph to be repositioned on the screen (using Grapher's LAYOUTGRAPH). This will destroy any work you have done moving nodes within the graph.

Reconnect Nodes first causes any link edges in the graph to be erased. (Note, however, that non-link edges, those created by "AddEdge" as described below, are ignored.) Then, each node in the graph is connected to every other node in the graph for which there is a link between them having one of the currently selected link types. This can be useful in several situations:

1. when the linking structure between cards has changed, but the current browser layout needs to be preserved.

2. when some browser nodes need to be moved, but dragging the connected links is too slow. In this case, do `UnconnectNodes` followed by `ReconnectNodes` (after you've moved the nodes around).

3. when special browser layouts are desired. For example, suppose you like the layout that Grapher gives you when certain links are left out or when you limit the depth. Then calling `ReconnectNodes` will fill in the missing links without affecting the graph's layout.

Unconnect Nodes simply erases all edges in the browser. This is useful for positioning a browser's nodes before invoking `ReconnectNodes`.

Expand Browser Node allows you to enlarge the graph under a given node. After selecting a node, you're asked for a depth (defaults to 1). The graph is then expanded under the selected node to the given depth, following any currently selected links. Note that `ExpandBrowserNode` calls `LAYOUTGRAPH` so any existing special node arrangements will be lost.

Graph Edit Menu brings up the graph editing menu. See the description below.

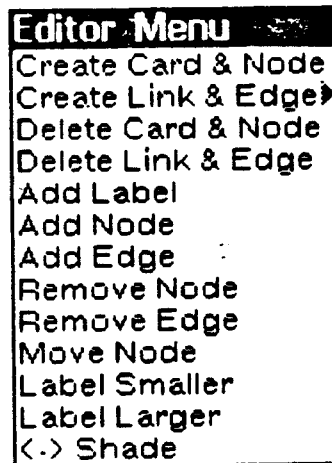
Change Browser Specs brings up the `BrowserSpecs` stylesheet to allow you to change any of the browser specs. These changes will be noticed at the next `RecomputeBrowser`, `ExpandBrowserNode`, etc.

Editing the Browser Manually and "Structure Editing":

Browsers have always been useful as static views of a portion of a notefile. Now, they can also serve as "reach-through" ports, allowing changes to be made to the notefile. This capability is handy when you want to create structure, but not content. For example, when brainstorming ideas for a paper, you might have card titles and links in mind, but want to put off the job of filling in each card's text till later.

A browser consists of **nodes** and **edges** (or lines). These usually correspond to cards and links in the notefile. However, it is also possible to annotate a browser with **labels** which are pieces of text representing nothing in the notefile (at least not as far as NoteCards is concerned). Edges can also be created joining labels and/or normal nodes which do not correspond to links in the notefile. These are called **non-link edges**.

Creation and deletion of nodes and edges in the browser is accomplished using the `GraphEditMenu` shown below. This menu can be obtained either by right-buttoning in the browser window or by choosing ***Graph Edit Menu*** from the title bar middle button menu.



The options in the *Graph Edit Menu* can be used to create either nodes representing existing cards in the notefile, new nodes and corresponding cards, or browser labels with no corresponding card. There are also operations for creating browser edges and accompanying links as well as edges without notefile analogues. Edges and nodes can be deleted in two ways: either such that the corresponding link or card is deleted as well as the node or edge, or just such that the node or edge is removed from the browser. These options are described further below.

Create Card & Node causes a new card to be created in the current Notefile and a corresponding node for it to be included in the browser. You're asked for the type of the new card, its title, and where to position the node representing it.

Create Link & Edge causes a new link to be created between two existing cards and a corresponding edge to be drawn in the browser. (We call such an edge representing a Notecards link, a "link edge." See AddEdge below for creating non-link edges.) You're asked for the "From" and "To" nodes in the browser corresponding to the cards to be linked as well as a link type. The link icon for the new link is positioned at the cursor point in the From card if the card has text substance and an open window. Text cards with closed windows have links inserted at the start of the text stream. Otherwise, the new link is a global link. You can have multiple link edges between pairs of cards. In this case the edges are displayed in a spline or "flower" arrangement.

Delete Card & Node causes a card to be deleted and its corresponding node in the browser to be removed. You are asked first to choose the node representing the card to be deleted and then to confirm the removal of the node (type "y" to confirm) and the deletion of the card. If the selected node is one of a set of virtual nodes (double boxed), then all nodes in the set (i.e. representing the given card) are removed.

Delete Link & Edge causes a link in the Notefile to be deleted and the corresponding edge in the browser to be removed. You first pick the "From" and "To" nodes corresponding to the source and destination ends of the link respectively. Then, if there is only one link

between those two cards, the link is deleted after user confirms. If there are multiple links between the two cards, then the user chooses from a menu of link types.

Add Label puts a "label node" into the browser that does not represent a Notecard. You are prompted for a string forming the node's label and then must position the label node. This node is not boxed. (But note that "virtual" label nodes can be boxed and thus can be confused with non-virtual regular nodes.)

Add Node adds a node into the browser corresponding to some existing card. You are asked to point to a card (title bar or link icon) on the screen that this node is to represent and then to position the node.

Add Edge draws a line between two nodes in the browser. This edge does not correspond to a real link in the Notefile. To avoid confusion, it is best to have the arrowheads option on (see Sections 13.3 and 13.2) in this case, since edges formed by AddEdge do not have arrowheads (or dashing). Only one such edge is allowed between any two nodes and none if there are already link edges between the nodes. Thus doing CreateLink&Edge will remove any existing non-link edge.

Remove Node removes a node from the browser. It does not delete the card (if any) that the node represents. Edges into and out of the node are also removed. If the selected node is one of a set of virtual nodes representing the same card, then you will be told how many nodes will be removed with this one and will be asked to confirm. The only way to remove only one node of a set of virtual nodes, is to first manually remove edges into and out of it using RemoveEdge. Then RemoveNode can be used to remove only the one virtual node.

Remove Edge removes an edge from the browser. It does not delete the link (if any) that the edge represents. The user is asked to select the "From" and "To" nodes of the edge.

Move Node allows you to change the position of any node, rubber banding any edges pointing to it. You're asked to point to the node by left-buttoning, and holding down the left button, drag the node to its new position.

Label Smaller is used to decrease the font size of label nodes. Note that it does not work for regular non-label nodes.

Label Larger is used to increase the font size of label nodes.

<-> Shade toggles the shade of a node between black-on-white and white-on-black. This can only be performed on label nodes (not on nodes representing Notecards).

FIX MENU causes the GraphEditMenu to be affixed to the lower right edge of the browser window. Note that this does not prevent you from obtaining the menu via right button inside the window.

[Note that the above editing commands do not work on old 1.1 browsers. Such browsers should either be recomputed (via `RecomputeBrowser`) or unconnected and reconnected.]

Browser operations have generally been sped up somewhat, though not nearly enough in the case of large browsers. One small change in the interests of speed is that the edges in the graph representing multiple links between the same pair of cards, which used to be drawn using curves ("flower links"), are now drawn with straight lines.

13.4. Miscellaneous System Enhancements.

Links ordering within text cards:

The internal list of outgoing links in a text card is now kept in the same order that the links appear in the card's text. This means, for example, that the daughters of a browser node for a filebox will appear in the correct order.

Link insertion:

The title bar menu entry for *Insert Link* now has an attached submenu containing entries for adding single links, multiple links, and global links. When inserting multiple links (or adding multiple global links) you're only asked for one link type which is used to label all the new links and all are inserted at the same place in the text.

Show links:

Show Links is now a standard entry in the left button title bar menu of a card (rather than a subentry under *Edit Properties*). The format of the *Show Links* display has been changed slightly. The prefix is now either TO, FROM, or Global TO. The link type is shown in the icon. Also, for text cards, the TO links should appear in the correct order.

Sketch changes and fixes:

NoteCards now uses the latest version of sketch. See the sketch documentation for details on changes. Several long-standing bugs having to do with link icons in sketch cards have been fixed.

Sketches and graphs in text cards:

It is possible to shift-copy the contents of sketch and graph/browser cards into text cards. In addition, the Document card is now able to include the contents of sketch and graph cards if encountering them during card gathering. (It is still not possible for Document to include the contents of cards having user-defined substance types such as NCFfile cards.)

Card Date Information:

Dates have been added to all entries in the data area of notefiles; they have also been made accessible from *Edit Property List* notecard title bar menu operation, since there is now date information stored with every card. In addition to the notecard's ID, there are four dates listed, one each for title, links, proplist, and substance. These indicate the last time that the given card part was saved to the notefile. This information may also be accessed from the programmer's interface through the card's property list by calling `NCP.CardPropList`. An example of the time stamps put on each card is shown in Figure 13.4-1.

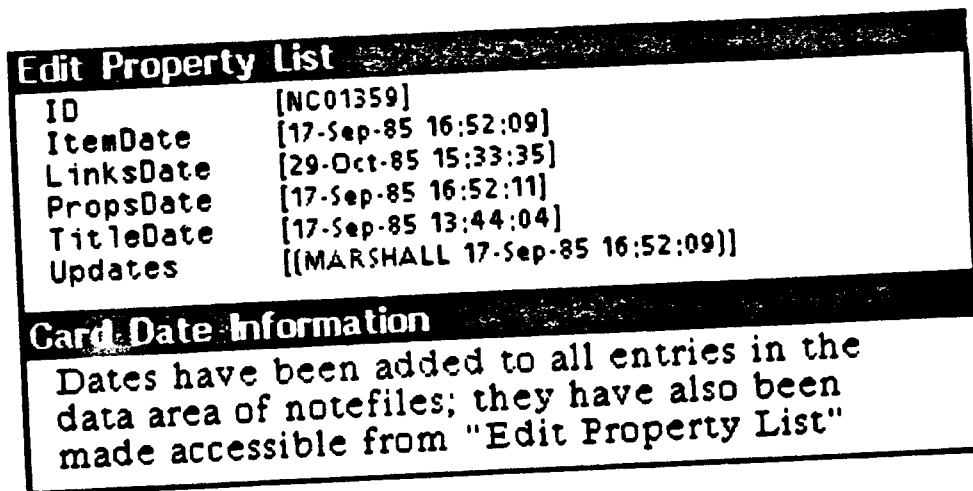


Figure 13.4-1 Example of Card Date Information

Ordering cards in a filebox:

It is now possible to dictate the relative placement of new cards in a filebox. If the `OrderingFn` property of a card has a value, it should be a lisp function that takes two card ID arguments and returns T if the first should appear before the second and NIL otherwise. You can make such a function appear automatically on new boxes for the case of alphabetizing by using the global parameter `AlphabetizeFileBoxChildren`. See Section 13.2.

Programmer's interface:

The Programmer's interface has been updated. Thus users with existing programmer's interface code should read the revised Programmer's Interface documentation in Appendix A. The changes are not all forward compatible.

Appendix A: NoteCards Programmer's Interface

This appendix describes a facility whereby users with some programming know-how can obtain a lisp interface to NoteCards. In this way, they can create and modify Notefiles, cards, and links under program control.

The functions described below are divided into 7 groups:

1. NoteFile Creation and Access
2. Creating and Accessing NoteCard Types
3. Creating NoteCards and FileBoxes
4. Accessing NoteCards and FileBoxes
5. Creating and Accessing Links
6. Creating and Accessing Link Labels
7. Handy Miscellaneous Functions

A.1. NoteFile Creation and Access

For each of the following functions (except `NCP.CloseNoteFile`), the argument is a filename. The suffix `".NoteFile"` is added if not already present. In any case, the filename used by NoteCards always has this suffix.

(NCP.CreateNoteFile <filename>)

If `<filename>` is not already a notefile, then create a notefile `<filename>.NoteFile`, and return this filename which can later be passed to `NCP.OpenNoteFile`.

(NCP.OpenNoteFile <filename> <don'tCreateFlg> <convertw/oConfirmFlg>)

If there is no currently open notefile, then open `<filename>` and make it the currently active NoteFile. Returns resultant stream if successful, else nil. If `<don'tCreateFlg>` is non-nil, then a new file will not be created if the given one doesn't exist. If `<convertw/oConfirmFlg>` is non-nil, then if needed, the file will be converted to the most recent format without user confirmation.

(NCP.CloseNoteFile [<stream>])

Closes `<stream>` if it corresponds to a currently open Notefile. Returns its filename if successful. If `<stream>` is nil, then closes current open notefile.

(NCP.CheckpointSession)

Checkpoint the current Notecards session, first writing out any dirty cards. In case of a system crash or abort, the notefile can be recovered to the last checkpoint. Note that closing a notefile does a checkpoint.

(NCP.AbortSession)

Abort the current Notecards session, losing all work since last checkpoint or successful close.

(NCP.RepairNoteFile <filename>)

Rebuilds the link structure of `<filename>`. It must **not** be currently open.

(NCP.CompactNoteFile <filename>)

Copies `<filename>` to a later version, recovering space. Must not be open.

(NCP.CompactNoteFileInPlace <filename>)

Compacts `<filename>` in place, replacing the old version. Must not be open.

(NCP.DeleteNoteFile <filename>)

Removes the `<filename>` notefile. Must not be open.

(NCP.CurrentNoteFileStream)

Returns the currently open notefile stream if there is one, else nil.

(NCP.CurrentNoteFile)

Returns the full name of the currently active notefile if there is one, else nil.

(NCP.CheckOutNoteFile <fromFilename> <toFilename>)

Copies `<fromFilename>` to `<toFilename>` unless `<fromFilename>` is locked. If successful, creates a lock file in `<fromFilename>`'s directory. The name of the lock file is formed by concatenating the atom LOCKFILE onto `<fromFilename>`.

(NCP.CheckInNoteFile <fromFilename> <toFilename>)

Check lock file for <toFilename>. If none, then just copy <fromFilename> to <toFilename>. If there is one and it's owned by us, then do the copy and remove the lock file. If there is a lock file owned by someone else or if date of <toFilename> is more recent than date of lock file, then print a message and do nothing.

A.2. Creating and Accessing NoteCard Types

These functions give the user access to the NoteCard user-defined types facility. For an explanation of this facility, see the NoteCards Types Mechanism documentation.

(NCP.CardTypes)

(NCP.SubstanceTypes)

Returns lists of all currently defined NoteCard types and substances, respectively.

**(NCP.CreateCardType <TypeName> <SuperType> <SubstanceType> <FnsAssocList>
<VarsAssocList>)**

Makes a new NoteCard type with name <TypeName>, super type <SuperType>, substance <SubstanceType>. Any functions not appearing in <FnsAssocList> will be inherited from <SuperType>. The CardWidth and CardHeight vars fields will be inherited if not specified in <VarsAssocList>. Other vars fields default to nil. Note that, for now, specializing the FileBox card type is not allowed.

(NCP.CreateSubstanceType <SubstanceName> <FnsAssocList> <VarsAssocList>)

Makes a new substance type with name <SubstanceName> and the given functions and vars fields. None of the function fields should be nil (but might conceivably be the function NILL).

(NCP.CardTypeSuper <type>)

Returns the super type of <type>.

(NCP.CardTypeSubstance <type>)

Returns <type>'s substance type.

(NCP.CardTypeLinkDisplayMode <type>)

Returns the link display mode of <type>.

(NCP.CardTypeFn <type> <fn>)

(NCP.CardTypeVar <type> <var>)

Returns the <fn> (<var>) field for <type>.

(NCP.CardTypeInheritedField <type> <field>)

Returns the value of the card type function or variable <field> for <type>. This is possibly different from the value returned by NCP.CardTypeFn or NCP.CardTypeVar in that if the defined value for <field> of <type> is nil, then the super is checked for a non-nil value. This checking continues until either a non-nil <field> is found or we reach the top of the super hierarchy. In that case, the value of <type>'s substance's <field> is used. Note that among the variable fields, only CardDefaultWidth and CardDefaultHeight inherit, so for the other Var fields, the result of NCP.CardTypeVar is valid (even if it's nil).

(NCP.SubstanceTypeFn <substance> <fn>)**(NCP.SubstanceTypeVar <substance> <var>)**

Returns the <fn> (<var>) field for the substance <substance>.

(NCP.CardTypeDisplayedInMenu <type> [<new val>])

Returns the old value of CardDisplayInMenuFlg for the given type. (That is, the flag indicating whether <type> currently appears on the Types menu.) If a second argument is given, then the value of CardDisplayInMenuFlg for the type is changed accordingly. Thus, if it is nil, then <type> is put on the Types menu. If non-nil, then it's taken off.

(NCP.ValidCardType <type>)

Returns non-nil if <type> is an existing NoteCard type.

(NCP.ValidSubstanceType <substance>)

Returns non-nil if <type> is an existing NoteCard substance type.

(NCP.ValidCardTypeFn <fn>)**(NCP.ValidCardTypeVar <var>)**

Returns non-nil if <fn> (<var>) is a valid function (variable) field for NoteCard types, for example, the litatom MakeCardFn (CardDefaultWidth). In other words, <fn> (<var>) can serve as the <fn> (<var>) arg to NCP.CardTypeFn (NCP.CardTypeVar).

(NCP.ValidSubstanceTypeFn <fn>)

(NCP.ValidSubstanceTypeVar <var>)

These return non-nil if <fn> (<var>) is a valid function (variable) field for substance types. In other words, <fn> (<var>) can serve as the <fn> (<var>) arg to NCP.SubstanceTypeFn (NCP.SubstanceTypeVar).

(NCP.CardTypeFns)

(NCP.CardTypeVars)

(NCP.SubstanceTypeFns)

(NCP.SubstanceTypeVars)

These return lists of all valid Fn (Var) fields for NoteCard types and substances respectively.

A.3. Creating NoteCards and FileBoxes

The following functions create various sorts of cards and boxes within the currently open notefile.

(NCP.CreateTextCard <title> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns a new notecard having type Text. If <title> is non-nil, it is installed as the Notecard's title, otherwise the title is "Untitled." <props>, if non-nil, should be a prop-list of properties and values to be placed on the user property list of the Notecard. If <parentfileboxes> is non-nil, then it should be a list of FileBoxes in which to initially file this card.

(NCP.CreateFileBox <title> <nodisplayflg> <props> <childcardsboxes> <parentfileboxes>)

Creates and returns a new Filebox with title <title> (or a gensym'ed name if <title> is nil). It will initially contain child cards and boxes from the list <childcardsboxes> (if that arg is non-nil). If <parentfileboxes> is nil, then the new filebox will be filed in the value of (NCP.GetToBeFiledFileBox). The <props> arg is handled as it was for NCP.CreateTextCard.

(NCP.CreateBrowserCard <title> <paramList> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns a new browser card with given title, props and parents. <paramList> should be a prop list of browser parameters. The properties currently recognized are:

ROOTCARDS: A list of Notecards to serve as roots of the forest or lattice generated by the browser. If omitted or NIL then user is asked to choose root cards.

LINKTYPES: A list of link types to follow when creating the browser. Any label present in the list having the backarrow prefix ("←") represents that link type but in the reverse direction. This list can also contain the atoms ALL or ←ALL in which case browsing will be done on all links in either the forward or reverse direction. If both ALL and ←ALL are specified, then links in both directions will be used (generally making a mess).

DEPTH: The depth at which to cut off the browser. This should be a non-negative integer. If NIL or omitted, then will assume no limit. (Currently integers greater than 9 are assumed equivalent to infinity.)

FORMAT: This should be a list of one, two or three elements. The first should be an atom indicating grapher format. The choices are FAST (laid out as a forest, sacrificing screen space for speed), COMPACT (laid out as a forest, using minimal screen space), LATTICE (laid out as a directed acyclic graph, the default), *GRAPH* (laid out as a graph, i.e. virtual nodes are eliminated). The second element of the FORMAT list, if present, should be either HORIZONTAL (the default) or VERTICAL specifying whether the graph is laid on its side or up and down. The third element, if present, should be the atom REVERSE. This indicates that horizontal graphs should be laid out from right to left instead of left to right and that vertical graphs should be laid out from bottom to top rather than vice versa.

(NCP.CreateSketchCard <title> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns an initially empty sketch/map card having given title, props, and parents.

(NCP.CreateGraphCard <title> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns an initially empty graph card having given title, props, and parents.

(NCP.CreateCard <type> <title> <nodisplayflg> <props> <parentfileboxes> <otherargs>)

Creates and returns a card of the given (possibly user-defined) type, with given title, props, and parents. <otherargs> is a possibly nil list of args that will be passed to the MakeCardFn of <type>. Card is initially displayed or not according to value of <nodisplayflg>.

(NCP.MakeDocument <rootcard> <parametersProplist> <nodisplayflg> <props> <parentfileboxes>)

Creates and returns a Document card starting from <rootcard>. The default set of parameters for making documents can be accessed via NCP.DocumentParameters, but some of

these can be given new values just for the duration of this MakeDocument by specifying a non-nil <parametersProplist>. For example, a value of '(TitlesFromNoteCards Bold ExpandEmbeddedLinks ALL) for <parametersProplist> would cause temporary changes to the values of the parameters TitlesFromNoteCards and ExpandEmbeddedLinks. As usual, the resulting card will have the given props and parents.

**(NCP.MakeLinkIndex <linktypes> <backpointersP> <nodisplayflg> <props>
<parentfileboxes>)**

Creates and returns a LinkIndex text card consisting of a sorted record of all instances of links in the current notefile having one of the given link types. <linktypes> can contain the literals ALL and/or ←ALL as well as any particular backwards links. (See the above description of NCP.MakeDocument.) Backpointer links are inserted in the text if <backpointersP> is non-nil. Resulting card will have given props and parents.

A.4. Accessing NoteCards and FileBoxes

The following functions provide access to the cards and boxes present in the current notefile. Note that whether a card's window has been brought up on the screen has little or no effect on the following functions. If the user changes some field of a card while that card is visible on the screen, then the field will update itself automatically. Thus, users can switch between program-driven and screen-interface-driven modes at will.

Cards can be active or inactive. An active card has its information cached (on its property list) thus saving time at the expense of memory. All cards visible on the screen are active. Most of the following functions leave the card in the same state as it was when they started (except NCP.BringUpCard, which makes it active). Thus, users needing to do several consecutive operations to the same card should consider temporarily caching the card's information via NCP.ActivateCards (and then uncache with NCP.DeactivateCards).

Most of the following functions take as first argument a card or filebox. If this does not in fact correspond to an existing card or box, then an error message is printed and nil is returned.

(NCP.BringUpCard <card> <region/position>)

Brings up on the screen the given card in the given region or at the given position. If <region/position> is nil, then user is asked to specify position with mouse.

(NCP.ActiveCardP <card>)

Returns non-nil if given card or box is currently active (i.e. information is currently cached in memory).

(NCP.ActivateCards <cardList>)

For each card or box in <cardList> (or just the one, if the argument is atomic), make it active (i.e. cache its information in memory).

(NCP.DeactivateCards <cardList>)

For each card or box in <cardList> (or just the one, if the argument is atomic), make it inactive (i.e. uncache its information back into the file). If any cards in <cardList> were on the screen then this will close their windows.

(NCP.CardType <card>)

Returns the type of <card> or NIL if the card does not exist.

(NCP.ValidCard <card>)

Returns non-nil if <card> exists (hasn't been deleted). (This is currently a synonym for NCP.CardType.)

(NCP.CardTitle <card> [<newtitle>])

Returns old title of <card>. If <newtitle> is present, then set <card>'s title to <newtitle>. <newtitle> can be an atom or string. Note, however, that all titles are converted internally to strings by NoteCards.

(NCP.FileCards <cards> <fileboxes>)

Every card or box in <cards> is filed in every box in <fileboxes>. Either arg may be an atom or a list.

(NCP.UnfileCards <cards> <fileboxes>)

Every card or box in <cards> is unfiled from every box in <fileboxes>. Furthermore if <cards> is the litatom ALL, then the boxes in <fileboxes> will be cleared of all children. Similarly, if <fileboxes> is the litatom ALL, then the cards and boxes in <cards> will be unfiled from all their parent boxes. Either arg may be an atom or a list.

(NCP.CardParents <card>)

Returns list of fileboxes in which <card> has been filed.

(NCP.FileBoxChildren <filebox>)

Returns list of children of <filebox> in the order in which they appear in the box's textstream.

(NCP.GetLinks <cards> <destinationCards> <labels>)

Returns list of all links from any of <cards> to any of <destinationCards> having any label in <labels>. Any of these arguments can be nil. For example, if <destinationCards> is nil, then all links pointing from <cards> to anywhere with a label in <labels> are returned. If both <cards> and <destinationCards> are nil, then this returns all links having a label in <labels>. If all three args are nil, then this is a slow synonym for NCP.AllLinks.

(NCP.CardPropList <card>)

Returns the prop list of the given card.

(NCP.CardProp <card> <propname> [<newvalue>])

Returns old value of property <propname> on <card>'s prop list. If <newvalue> is present, then set <card>'s <propname> property to <newvalue>. (Semantics are analogous to the Interlisp function WINDOWPROP.)

(NCP.CardAddProp <card> <propname> <newitem>)

Adds <newitem> to the list present on the <propname> property of <card>. Returns old value of property. (Semantics are analogous to WINDOWADDPROP.)

(NCP.CardDelProp <card> <propname> <itemToDelete>)

Deletes <itemToDelete> from the <propname> property of <card> if it is there, returning the previous value of that property. If not there, return nil. (Semantics are analogous to WINDOWDELPROP.)

(NCP.CardSubstance <card>)

Returns the substance of <card>. This is a textstream in the case that the type of <card> has TEXT substance. Otherwise, it is the appropriate underlying structure if <card> has GRAPH or SKETCH substance.

(NCP.CardRegion <card>)

Returns the region of <card>. This works even if <card> is not currently up on the screen, since the region information is stored on the notefile.

(NCP.CardAddText <card> <textstr> <loc>)

Adds the text within the string <textstr> to the text card <card>. If <loc> is the litatom START or END, then the text will be placed at the start or end of the card respectively. If <loc> is a number, then it is assumed to be a character count within the card at which to place the new text. If <loc> is NIL, then the text is placed at the current cursor location.

(NCP.ChangeLoc <card> <loc>)

Changes the cursor's location in <card>'s textstream to <loc>. Possible values for <loc> are as described for NCP.CardAddText.

(NCP.DeleteCards <cards>)

Deletes the given cards and fileboxes from the current notefile, or deletes just the one if <cards> is atomic.

(NCP.FileBoxP <card>)

Returns non-nil if <card> is a filebox.

(NCP.AllCards)

Returns a list of all extant cards for the current notefile.

(NCP.AllBoxes)

Returns a list of all fileboxes in the current notefile.

(NCP.MapCards <fn>)

Maps down the set of all cards in the current notefile, applying <fn> to each.

(NCP.MapBoxes <fn>)

Maps down the set of all fileboxes in the current notefile, applying <fn> to each.

(NCP.GetContentsFileBox)**(NCP.GetOrphansFileBox)****(NCP.GetToBeFiledFileBox)**

These functions retrieve the three predefined FileBoxes for the currently open NoteFile. These boxes can be modified (but not deleted) by the user in the same way as any other filebox.

A.5. Creating and Accessing Links

Links can be connected to points within a card or to the card as a whole, thus the following four link creation functions are provided. Those that connect to points within a card specify at least one of `<fromloc>` or `<toloc>`. If nil, then the link icon is placed at the current cursor location in the card. If the arg is the litatom `START` or `END`, then it is placed at the front or end of the text respectively. If the loc arg is a number, then it is assumed to be a character count at which to place the link icon.

(NCP.GlobalGlobalLink <label> <sourceCard> <destinationCard>)

Creates and returns a new link with label `<label>`, connecting `<sourceCard>` to `<destinationCard>`.

(NCP.LocalGlobalLink <label> <sourceCard> <destinationCard> <fromloc> <displaymode>)

Creates and returns a new link with label `<label>`, connecting from `<fromloc>` of `<sourceCard>` card to `<destinationCard>`. If `<displaymode>` is non-nil, then the new link is displayed in the given mode. Otherwise the default displaymode for the source card's type is used.

(NCP.GlobalLocalLink <label> <sourceCard> <destinationCard> <toloc>)

Not implemented at this time.

(NCP.LocalLocalLink <label> <sourceCard> <destinationCard> <fromloc> <toloc>)

Not implemented at this time.

(NCP.LinkDesc <link>)

Returns list of three items (`<label>` `<sourceDesc>` `<destinationDesc>`) where `<label>` is the link type and `<sourceDesc>` and `<destinationDesc>` have the form (`<anchor mode>` `<card>` `<loc>`). `<anchor mode>` is either `LOCAL` or `GLOBAL`, `<card>` is the card at this end of the link, and `<loc>` gives a position in the text of `<card>` if `<anchor type>` is `LOCAL` and `<card>`'s substance's type is `TEXT`.

(NCP.LinkDisplayMode <link> [<newdisplaymode>])

Returns old display mode of `<link>`. If `<newdisplaymode>` is present, then set `<link>`'s displaymode accordingly. If non-nil, it can be one of the litatoms `Icon`, `Title`, `Label`, or `Both`. Or

it can be an instance of the LINKDISPLAYMODE record. This has the 3 fields SHOWTITLEFLG, SHOWLINKTYPEFLG, and ATTACHBITMAPFLG. Each field can have one of the three values T, NIL, or FLOAT. If a field, say SHOWTITLEFLG, has value FLOAT then the corresponding global parameter (DefaultLinkIconShowTitle, in this case) will be consulted to decide whether or not to display the destination card's title in this icon. (See Section 7 for a description of the global parameters.)

(NCP.LinkLabel <link> [<newlabel>])

Returns old label of <link>. If <newlabel> is present, set <link>'s label to <newlabel>.

(NCP.GetLinkSource <link>)

Returns the card at the source end of <link>.

(NCP.GetLinkDestination <link>)

Returns the card at the destination end of <link>.

(NCP.DeleteLinks <links>)

Removes all links in <links> (or the single one if <links> is atomic).

(NCP.ValidLink <link>)

Returns non-nil if <link> is a link in the current notefile.

(NCP.AllLinks)

Returns a list of all existing links in the current notefile. (This is equivalent to but faster than (NCP.GetLinks NIL NIL NIL).)

(NCP.MapLinks <fn>)

Maps down the set of all links in the current notefile, applying <fn> to each one.

A.6. Creating and Accessing Link Labels

The following functions allow the user to manipulate link labels.

(NCP.CreateLinkLabel <label>)

Creates a new link label with name <label> for current notefile unless there is already one defined by that name.

(NCP.DeleteLinkLabel <label>)

Deletes the link label <label> from the current notefile. The label must exist and must not be the label of any existing link, and it must not be a system-defined link label (e.g. SubBox or FiledCard).

(NCP.RenameLinkLabel <label> <newlabel>)

Changes any links having label <label> to have label <newlabel>. <label> must exist and neither <label> nor <newlabel> should be a system-defined label.

(NCP.GetLinkLabels)

Returns a list of all existing link labels including system-defined ones.

(NCP.GetReverseLinkLabels)

Returns a list of the reverse labels for every existing link label. Thus, whereas SubBox would appear in the list returned by NCP.GetLinkLabels, ←SubBox would appear in the list returned by NCP.GetReverseLinkLabels.

(NCP.GetUserLinkLabels)

Returns a list of all existing user-defined link labels.

(NCP.ValidLinkLabel <label>)

Returns non-nil if <label> is a defined link label for current notefile.

A.7. Handy Miscellaneous Functions**(NCP.TitleSearch <key> <key> ...)**

Returns a list of all cards having all of the <key>s (can be atoms, numbers or strings) within their titles.

(NCP.PropSearch <propOrPair> <propOrPair> ...)

Returns a list of all cards such that for every <propOrPair> arg, if it is atomic, then the card must contain that property. If it is a list of two elements, then the card must have a property EQ to the first element with value EQ to the second element.

(NCP.WhichCard <x> <y>)

Returns the card currently displayed on the screen whose window contains the position in screen coordinates of <x> if <x> is a POSITION, the position (<x>,<y>) if <x> and <y> are numbers, or the position of the cursor if <x> is NIL. Returns NIL if the coordinates are not in the window of any card. If they are in the window of more than one card, then returns the uppermost. If <x> is a window, then NCP.WhichCard will return the card associated with that window.

(NCP.CardFromWindow <window>)

Returns the card associated with <window>, or NIL if not a notecards window.

(NCP.CardWindow <card>)

Returns <card>'s window if <card> is currently displayed somewhere on the screen.

(NCP.SelectCards)

Returns a list of those cards selected from the screen. A menu appears near the top of the screen with buttons for "DONE" and "CANCEL". Selections are made by left buttoning in the title bars of the desired cards.

(NCP.DocumentParameters <parametersProplist>)

Returns the old value of the document parameters in the form of a proplist. If <parametersProplist> is non-nil then it should be a proplist whose properties are (some of the) valid document parameter names and whose values are permissible values for those parameters. The valid parameters and possible values are as follows:

HeadingsFromFileboxes: NumberedBold, UnnumberedBold, NONE.

TitlesFromNoteCards: Bold, NotBold, NONE.

BuildBackpointers: ToCardsBoxes, ToCards, ToBoxes, NONE.

CopyEmbeddedLinks: ALL, NONE, <listOfLinkLabels>.

ExpandEmbeddedLinks: ALL, NONE, <listOfLinkLabels>.

[See Section 2.1 for an explanation of these parameters and how their values affect the document created.]

(NCP.NoteCardsParameters <parametersProplist>)

Returns the old value of the global Notecards parameters in the form of a proplist. If `<parametersProplist>` is non-nil then it should be a proplist whose properties are (some of the) valid document parameter names and whose values are permissible values for those parameters. The valid parameters and possible values are as follows:

DefaultCardType: `<legalCardType>`

FixedTopLevelMenu: T or NIL

ShortWindowMenus: T or NIL

ForceSources: T or NIL

ForceFiling: T or NIL

ForceTitles: T or NIL

CloseCardsOffScreen: T or NIL

BringUpCardsAtPreviousPos: T or NIL

MarkersInFileBoxes: T or NIL

AlphabetizedFileBoxChildren: T or NIL

DefaultLinkIconAttachBitmap: T or NIL

DefaultLinkIconShowTitle: T or NIL

DefaultLinkIconShowLinkType: T or NIL

LinkDashingInBrowsers: T or NIL

ArrowHeadsInBrowsers: one of the listatoms {AtEndpoint, AtMidpoint, None}

SpecialBrowserSpecs: T or NIL

AnnoAccessible: T or NIL

EnableBravoToTEditConversion: T or NIL

DefaultFont: a font

LinkIconFont: a font

Here, `<legalCardType>` should be an existing Notecard type, i.e. one that appears in the list returned by `NCP.CardTypes`.

(NCP.PrintMsg <window> <clearFirstFlg> <arg1> <arg2> ...)

Prints a message in the prompt window of `<window>`. If `<window>` is NIL, then prints message in the Lisp prompt window. If `<clearFirstFlg>` is non-nil, then clears the prompt window first. The args are PRIN1'ed one at a time.

(NCP.ClearMsg <window> <closePromptWinFlg>)

Clears the prompt window associated with `<window>` (or with the main Lisp prompt window if `<window>` is NIL) and closes it if `<closePromptWinFlg>` is non-nil.

**(NCP.AskUser <Msg> <prompt> <FirstTry> <ClearFirstFlg> <MainWindow>
<DontCloseAtEndFlg> <DontClearAtEndFlg> <PROMPTFORWORDFlg>)**

This function can be used to ask questions of the user in a window's prompt window. The `<Msg>` and `<prompt>` are printed along with `<FirstTry>` (if non-nil). The value returned is whatever the user types. If `<ClearFirstFlg>` is non-nil, then the prompt window is cleared first. If `<MainWindow>` is nil, then the top level prompt window is used. If `<DontCloseAtEndFlg>` is non-nil, then the prompt window won't be closed after the question is answered and if `<DontClearAtEndFlg>` is non-nil, then the prompt window won't be cleared at the end. If `<PROMPTFORWORDFlg>` is non-nil, then the PROMPTFORWORD typein protocol will be used rather than TTYIN. The former doesn't allow mouse editing of the string typed in. On the other hand, typing automatically overwrites the prompt when PROMPTFORWORD is used.

(NCP.AddTitleBarMenuItems <Win> <NewMenuItems>)

Adds the given menu items to the left button title bar menu of `<Win>`. `<Win>` should be the window of a visible notecard.

(NCP.GetDates <Card>)

Returns a NOTECARDDATES record structure containing the dates of last modification of each of the four card parts of `<Card>`. The fields of the record are SUBSTANCEDATE, TITLEDATE, LINKSDATE and PROPLISTDATE.

Appendix B: The NoteCards Types Mechanism

1. Introduction

The NoteCards types mechanism allows a user with some knowledge of Interlisp to add new types of note cards to the system. The types mechanism is built around an inheritance hierarchy of note card types. If the user needs to create a new card type that is a small change from an already existing card type, he or she need only define the few functions or parameters that account for the differences between the new card and the existing card. However, if the user wishes to create a totally new type of card, then he or she must define the 20-odd functions and parameters that make up a note card type.

Every note card has a substance. A substance is essentially a data structure that contains the information in the note card. Different types of note cards have different types of substances. Associated with every substance type is an editor that can be used to create and/or modify the data structure of that substance type. For example, the substance of a Text card is a TEXTSTREAM that can be edited using TEdit. Similarly, the substance of a Browser card is a GRAPH record that can be edited using GRAPHER. Defining a new note card type involves specifying the functions necessary to handle the card's substance and its editor.

B.1.1 The Inheritance Hierarchy

The inheritance hierarchy in NoteCards has two parts: a tree of NoteCardTypes and a list of SubstanceTypes. Every NoteCardType has a super-type and a substance type. The super-type is an already existing NoteCardType from which the NoteCardType will inherit fields. Thus, the set of NoteCardTypes forms a tree structure based on the super-type field. The substance type of a NoteCardType is an already existing SubstanceType.

The inheritance process for a given field of a NoteCardType works as follows: if the field has a non-NIL value in the NoteCardType then this value is used, otherwise the field value is inherited from its super-type. If there are no non-NIL values anywhere in the inheritance path for the NoteCardType, then the field value is taken from the corresponding field in the substance type for the NoteCardType. Substance types are guaranteed to have values in all of their fields.

Example: ProtectedText is a card with super-type Text. Text in turn has super-type NoteCard (the null root of the NoteCardType tree). In addition, Text has substance type TEXT. If an EditCardFn is not defined in ProtectedText, then it will be inherited from Text. If Text doesn't have an EditCardFn then the EditSubstanceFn from the TEXT SubstanceType will be used (since NoteCard by definition does not have an EditCardFn).

Functions are inherited all or none. Often, however, a new NoteCardType will require only a minor addition to the corresponding function of its super-type. In this case, the new card type should define a

new function, but this function can call the corresponding function of its super-type to do the bulk of the work. The following construction will accomplish this goal:

```
(APPLY* (NCP.CardTypeInheritedField (NCP.CardTypeSuper <type>) <fn>) <arg1> <arg2> ...)
```

where <type> is the TypeName of the card type in question, <fn> is the name of the function in question, and <arg1> <arg2> ... are the arguments to that function. For example the following might be the definition of the EditCardFn for the passworded Text card called ProtectedText:

```
(DEFINEQ
(NC.EditProtectedTextCard
(LAMBDA (ID Substance Region/Position)
(* * Edit a Protected Text card, asking for the password first.)
(PROG (Password Result)
(* * Get this card's password from the prop list)
(SETQ Password (NCP.CardProp ID (QUOTE Password)))
(COND
((EQUAL Password (NC.GetPassword ID))
(* Password is okay.
Call the EditCardFn of my super-type)
(SETQ Result (APPLY*
(NCP.CardTypeInheritedField
(NCP.CardTypeSuper (QUOTE ProtectedText))
(QUOTE EditCardFn))
ID Substance Region/Position)))
(T (* Password is bad. Express condolences)
(NCP.PrintMsg Window T "Sorry." (CHARACTER 13)
"You do not know the password!!"
(CHARACTER 13)
"Bye."
(CHARACTER 13))
(DISMISS 2000)))
(RETURN Result))))
```

B.1.2 Links and Link Icons

An integral part of NoteCards is the ability to create a link between two note cards. Presently, there are two kinds of links: *GlobalToGlobal* links and *LocalToGlobal* links. *GlobalToGlobal* links connect one entire card with another entire card and are stored separately from either card's substance. *GlobalToGlobal* links are maintained (almost) entirely by the NoteCards system code and therefore do not vary across note card types.

LocalToGlobal links connect a particular position within the substance of one card (the *source* card) to the entirety of the other card (the *destination* card). Within the source card, the link is represented by an image object called a link icon that must be contained by the card's substance. Since substances vary across note card types, the handling of link icons varies across note card types. The destination (or Global) end of a LocalToGlobal links is maintained by the NoteCards system code.

Not all note cards can be the source of LocalToGlobal links. Card types that support LocalToGlobalLinks must have their LinkAnchorModesSupported parameter set to T. If a this parameter has any other value, then cards of this type can be the source of only GlobalToGlobal links. These Global-links-only card types need to provide only one piece of functionality in support of the linking mechanism. In particular, they must provide user access to the function NCP.GlobalGlobalLink from the editor that runs when the card is being displayed. For example, the editor's command menu might include an "Insert Global Link" command. All other link maintenance is carried out by the NoteCards system.

If a card type supports LocalToGlobal links, then it must contain the necessary mechanisms for supporting link icons in its substance. Link icons are instances of standard Interlisp-D image objects (See documentation of Image Objects in Interlisp-D). The mechanisms supporting link icons include functions for inserting, deleting, updating, and collecting the link icons contained in a card's substance. These functions are described in detail below. In addition to these functions, a note card type supporting LocalToGlobal links must provide user access to the function NCP.LocalGlobalLink from the editor that runs when the card is being displayed. In addition the editor must provide user access to the function NCP.GlobalGlobalLink.

Inside the link icon image object is a link record containing all of the information about the link. These link records can be manipulated using the link manipulation functions provided by NoteCards' programmer's interface (e.g., NCP.GetLinkDestination returns the destination field of a link record). The functions required to define a note card or substance type deal in both link records and link icons. You can translate between these two representations using the functions NC.MakeLinkIcon and NC.FetchLinkFromLinkIcon; NC.MakeLinkIcon will create a link icon image object from a link record, while NC.FetchLinkFromLinkIcon will return the link record contained in a link icon.

B.1.3 Using the Types Mechanism

Most uses of the types mechanism involve defining new NoteCardTypes. Usually, these new NoteCardTypes involve specifying a TypeName, a SuperType, a SubstanceType, and one or two functions that differ from the SuperType. The most commonly defined functions are the MakeCardFn, the EditCardFn and the QuitCardFn.

Definition of new substance types occurs only when a new kind of substance (e.g., a spreadsheet) and its corresponding editor are to be added to the system. When defining a substance, all of its fields must be fully defined since there is no inheritance among SubstanceTypes.

B.2. The NoteCardType

Each note card type in the system is defined by a record structure (i.e., a NoteCardType) containing about 20 names, functions and parameters. The functions implement behaviors that are required by the NoteCards system but vary across the different card types. For example, one function is responsible for writing the card's substance to the NoteFile. The parameters represent specifications that inform NoteCards about the specific properties of each card type, e.g., whether it handles local links or not.

The NoteCardTypes are organized into an inheritance hierarchy. Each NoteCardType has a super-type. If any of the functions or parameters is not specified for a given NoteCardType, that function or parameter is inherited from its super-type (or its super-type's super-type, if the function or parameter is not specified for the super-type either). Each NoteCardType also has a SubstanceType. If any of the functions or parameters cannot be found along the super-type chain of the NoteCardType, then the card type inherits the function or parameter from its SubstanceType.

Overall, a card type is a data structure with the following 21 fields:

Inheritance Hierarchy Specifications

- 1) TypeName
- 2) SuperType
- 3) SubstanceType

Functions

- 4) MakeCardFn
- 5) EditCardFn
- 6) QuitCardFn
- 7) GetCardFn
- 8) PutCardFn
- 9) CopyCardFn
- 10) MarkCardDirtyFn
- 11) CardDirtyPFn
- 12) CollectLinksInCardFn
- 13) DeleteLinksInCardFn
- 14) UpdateLinkIconsInCardFn
- 15) InsertLinkInCardFn
- 16) TranslateWindowPositionToCardPositionFn

Parameters

- 17) LinkDisplayMode
- 18) CardDefaultWidth
- 19) CardDefaultHeight
- 20) CardLinkAnchorModesSupported
- 21) CardDisplayedInMenuFlg

These fields are defined as follows:

1. **TypeName:** The atom that is the name of this card type. TypeNames must be unique among the NoteCardTypes tree, though they may overlap with SubstanceNames. The convention is that NoteCardType TypeNames have only the first letter capitalized. This is to set them apart from SubstanceNames which are by convention all caps.
2. **SuperType:** The TypeName of the NoteCardType that is the super-type for this NoteCardType. When a new NoteCardType is created, its SuperType must be an existing NoteCardType.
3. **SubstanceType:** The SubstanceName for the substance of this card type. When a new card is created, its SubstanceType must be the name of an existing SubstanceType (see Section 3.0 below). The basic NoteCards system includes the following substance types: TEXT, SKETCH, GRAPH which represent the substances handled by the TEdit, Sketch, and Grapher packages respectively.
4. **MakeCardFn:** The name of a function to be applied to an ID, a Title, and a NoDisplayFlg. The function should create a new card of this type. The ID is the note card ID that will be assigned to the newly created card. It should be used to set the various properties of the new card. The title is a string specifying the title of the new card. It can be used in messages to the user or to set the title of any windows created. NoDisplayFlg determines whether the new card is to be displayed on the screen or not. If NoDisplayFlg is NIL, then the card is to be displayed in a window on the screen. If NoDisplayFlg is non-NIL, then the card is to be created but not displayed on the screen.

The MakeCardFn should return the window of the new card if NoDisplayFlg is non-NIL and the ID if NoDisplayFlg is NIL.

Before returning, every MakeCardFn is required to set the substance property of ID by calling (NC.SetSubstance ID *Substance*) where *Substance* is whatever is considered a substance for this card type. For example, a TextStream for Text cards, a Graph record for Graph cards, or a Sketch record for Sketch cards.

By convention, every `MakeCardFn` sets the `SHRINKFN` of any window it creates to the function `NC.ShrinkFn` using `WINDOWPROP`.

5. `EditCardFn`: The name of a function to be applied to `ID`, `Substance`, and `Region/Position`. The function should start an editor for the given card. `ID` is the note card ID of the card. `Substance` is the substance of the card; it will be a thing of whatever type is considered a substance for this card type, e.g., a `TextStream` or `Sketch` record. `Region/Position` is a `Region` or a `Position` on the screen that specifies where the card is to be placed. (`NC.DetermineDisplayRegion ID Region/Position`) is a function that will determine the exact region for the card's window given the `ID` and the `Region/Position`.

The `EditCardFn` should return the editor window.

The `EditCardFn` is responsible for checking to make there is not already an editor for card `ID` already on the screen. If there is, the `EditCardFn` should just flash the previous editor window.

By convention an `EditCardFn` sets the `SHRINKFN` of any window it creates to the function `NC.ShrinkFn` using `WINDOWPROP`. Also by convention, an `EditCardFn` should set the title of the editor window to be the value of (`NCP.CardTitle ID`).

6. `QuitCardFn`: The name of a function to be applied to `WindowOrSubstanceOrID` which is either the editor window for a card or the substance of a card or a note card `ID`. `QuitCardFn` should quit out of the editor currently operative on the specified card and close the window containing the card.

The value returned by `QuitCardFn` is unspecified.

Before returning the `QuitCardFn` should apply the function `NC.DeactivateCard` to the `ID` of the card. Note that the `ID` may have to be computed from the `Window` or `Substance` passed to the `QuitCardFn`. The function `NC.CoerceToID` will do this computation.

The `QuitCardFn` should also insure that all processes related to this card are completed (or guaranteed to eventually complete) before returning.

7. `GetCardFn`: The name of a function to be applied to the `DatabaseStream`, a card `ID`, and a screen `Region`. The `GetCardFn` should read the substance of the note card specified by `ID` from the `DatabaseStream`. The format of the data to be read is determined by the `PutCardFn` (see below). When the `GetCardFn` is called, the file pointer for `DatabaseStream` is positioned on the first byte of the data to be read.

The `GetCardFn` should return a pointer to the substance read from the `DatabaseStream`.

GetCardFn need produce no side-effects. The ID and the Region are for reference purposes only.

Note that the GetCardFn need only read the substance of the card, i.e., that information about the card which is specific to its card type. General information about a card such as its title, its property list, its list of links, etc. is read from the DatabaseStream by the system.

8. PutCardFn: The name of a function to be applied to a note card ID and the DatabaseStream. The PutCardFn should write the substance of the note card specified by ID to the DatabaseStream. When the PutCardFn is called, the file pointer for DatabaseStream is positioned at the first byte assigned to the card. When the PutCardFn returns, the file pointer should be positioned immediately after the last byte written.

The format for writing the card's substance is fairly unrestricted. The data written on the DatabaseStream can take up any number of bytes, but the bytes must be contiguous. It must be written so that it can be recovered by reading from the DatabaseStream using the GetCardFn. The only other restriction is that the first 6 bytes of the substance must contain the file position of the start and the end of the substance: 3 bytes for the start file pointer and 3 bytes for the end file pointer. These pointers are for use by the CopyCardFn.

The value returned by the PutCardFn is unspecified.

Note that the PutCardFn need only write out the substance of the card, i.e., that information about the card which is specific to its card type. General information about a card such as its title, its property list, its list of links, etc. is written to the DatabaseStream by the system.

9. CopyCardFn: The name of a function to be applied to a note card ID, a "from" DatabaseStream, and a "to" DatabaseStream. The CopyCardFn should copy the substance for the note card specified by ID from the "from" DatabaseStream to the "to" DatabaseStream. When the CopyCardFn is called the file pointer for the "from" DatabaseStream is positioned on the first byte of the data to be copied. The file pointer for the "to" DatabaseStream is positioned at the first byte of the space assigned to the card on the "to" DatabaseStream.

The format for writing the substance on the "to" DatabaseStream has the same restrictions as for the PutCardFn.

Most often the CopyCardFn is a simple COPYBYTES that uses the start and end pointers written by PutCardFn in the first 6 bytes of the substance. Note, however, that all file absolute pointers (including the start and end pointers) must be updated; the file location on the "to" DatabaseStream is almost never the same as the original file location on the "from" DatabaseStream.

The value returned by the CopyCardFn is unspecified.

The CopyCardFn is used primarily by the compactor that eliminates "dead" space in the database. Thus, it is important that the CopyCardFn be as time efficient as possible.

10. MarkCardDirtyFn: The name of a function to be applied to a note card ID and a ResetFlg. If the ResetFlg is non-NIL, the function should mark the card specified by ID as being dirty (i.e., changed since it was last written to the DatabaseStream). If the ResetFlg is NIL, the function should reset the "dirtiness" of the card.

The MarkCardDirtyFn is called by NoteCards system functions that change the card. It is not necessarily called by user operations inside the editor on the card. Therefore, it is best if the mechanism used by the MarkCardDirtyFn is somehow coordinated with the corresponding mechanism used by the editor on the card. (See the CardDirtyPFn below.)

The value returned by the MarkCardDirtyFn is unspecified.

The card specified by ID is guaranteed to be active.

11. CardDirtyPFn: The name of a function to be applied to a note card ID. The function should return a non-NIL value if the card specified by ID is dirty, i.e., if it was changed since it was last written to the DatabaseStream. NIL should be returned otherwise.

Note that a "dirty" card is one that has been changed in any way. Only NoteCards specific changes to a card will result in a call to the card's MarkCardDirtyFn. Changes made through the editor on the card will use the editors "mark dirty" mechanism and will not call the MarkCardDirtyFn. Therefore, the CardDirtyPFn should check all dirty flags, i.e., the dirty flag set by the MarkCardDirtyFn as well as any set by the card's editor.

The card specified by ID is guaranteed to be active.

12. CollectLinksInCardFn: The name of a function to be applied to a note card ID, a CheckAndDeleteFlg, a DatabaseStream, a ReturnLinkIconsFlg, and a ReturnLocationsFlg. The function should examine the substance of the card specified by ID and produce a list of the links (or link icons) contained by the substance. The ReturnLinkIconsFlg and the ReturnLocationsFlg determine the contents of the list to be returned as follows:

ReturnLinkIconsFlg and ReturnLocationsFlg both NIL: the list to be returned should be a list of link records.

ReturnLinkIconsFlg is non-NIL, ReturnLocationsFlg is NIL: the list to be returned should be a list of link icons.

ReturnLinkIconsFlg is NIL, ReturnLocationsFlg is non-NIL: the list to be returned should be a list of pairs where the first member of the pair is a link record and the

second member of the pair is the "location" of the link icon for that link inside the substance.

ReturnLinkIconsFlg and **ReturnLocationsFlg** both non-NIL: the list to be returned should be a list of pairs where the first member of the pair is a link icon and the second member of the pair is the "location" of that link icon.

If **CheckAndDeleteFlg** is non-NIL, then the list produced by **CollectLinksInCardFn** should contain valid links only. Any links found to be invalid should be deleted. To check the validity of a link, the function **NC.ValidLinkP** should be applied to the link record and the **DatabaseStream**. To delete a link, apply the function **NC.MakeInvalidLink** to the link icon.

The **CollectLinksInCardFn** should return the list produced **CONSEd** to a dirty flag. The dirty flag should be non-NIL if any links were deleted, NIL otherwise.

The card specified by ID is guaranteed to be active.

13. DeleteLinksInCardFn: The name of a function to be applied to a "source" note card ID and a link record or "destination" note card ID. If the second argument is a link, the function should remove from the substance of the card specified by "source" ID the link icon corresponding to link. If the second argument is a "destination" note card ID, the function should remove from the substance of the card specified by "source" ID all link icons corresponding to links pointing to the card specified by "destination" ID.

To "remove" a link icon, the link icon should be replaced in the substance by the image object that is the value of **NC.DeletedLinkImageObject**. Note that before deleting the link icon, it is best to replace the **IMAGEOBJFNS** of the link icon with the value of **NC.NoDeleteImageFns**. This will prevent the link icon's **WHENDELTEDFN** from being activated when the deletion takes place.

The value returned by the **DeleteLinksInCardFn** is unspecified.

The card specified by "source" ID is guaranteed to be active.

14. UpdateLinkIconsInCardFn: The name of a function to be applied to a "source" note card ID or window and a "destination" note card ID. The function should update (i.e., force a redisplay of) all link icons in the "source" card that represent links pointing to the "destination" card. This function is called when some property of the link is changed by the NoteCards code. It is also called when certain properties of the destination card (e.g., its title) are changed.

The value returned by the **UpdateLinkIconsInCardFn** is unspecified.

The "source" card is guaranteed to be active.

15. InsertLinkInCardFn: The name of a function to be applied to a window, a link, and a position. The function should insert a link icon containing the link into the card being edited in the window at the position specified. The position is whatever object is returned by the `TranslateWindowPositionToCardPositionFn`.

The value returned by the `InsertLinkInCardFn` is unspecified.

The ID of the card being edited by the window is guaranteed to be the `SOURCEID` of the link.

16. TranslateWindowPositionToCardPositionFn: The name of a function to be applied to a window, an X-coordinate in that window, and a Y-coordinate in that window. The window is an editor window on the substance of some card. The function should return a position object that describes the position in the card substance that is currently located at the given X-Y position in the window. The format of the position object is undefined. It will be passed to the `InsertLinkInCardFn` and used as the position at which to insert a links in the card being edited in the window.

17. LinkDisplayMode: determines the default display mode for link icons inserted into cards of this type. It must be a record of type `LINKDISPLAYMODE`. `LINKDISPLAYMODE` describes what information will be displayed in a link icon. It consists of three flags: `SHOWTITLEFLG`, `SHOWLINKTYPEFLG`, and `ATTACHBITMAPFLG`. If `SHOWTITLEFLG` is non-NIL, the link icon will display the destination card's title. If `SHOWLINKTYPEFLG` is non-NIL, the link icon will display the type of the link. If `ATTACHBITMAPFLG` is non-NIL, a bit map describing the type of the destination card will be attached to the right of the link icon.

Note: This property is NOT inherited.

18. CardDefaultWidth: The default width for editor windows on cards of this type.

19. CardDefaultHeight: The default height for editor windows on cards of this type.

20. CardLinkAnchorModesSupported: an atom that determines the kind of links this card type will support (i.e., the kind of links for which cards of this type can be a source). If NIL, then this card type does not support links of any type. If `Global`, this card supports only `Global` links. If `Local`, this card supports only `local` links. If `T`, this card supports both `Global` and `Local` links.

Note: This property is NOT inherited.

21. CardDisplayedInMenuFlg: if non-NIL then this card type will appear in the choice of card types in the menu used during card creation using the "Create" entry in the main NoteCards menu. If NIL, then this card type will not appear in this menu.

B.3. The SubstanceType

The SubstanceType is a record structure whose fields are virtually identical to those of the NoteCardType record. In particular, the SubstanceType has the following 17 fields:

- 1) SubstanceName
- 2) CreateSubstanceFn
- 3) EditSubstanceFn
- 4) QuitSubstanceFn
- 5) GetSubstanceFn
- 6) PutSubstanceFn
- 7) CopySubstanceFn
- 8) MarkSubstanceDirtyFn
- 9) SubstanceDirtyPFn
- 10) CollectLinksInSubstanceFn
- 11) DeleteLinksInSubstanceFn
- 12) UpdateLinkIconsInSubstanceFn
- 13) InsertLinkInSubstanceFn
- 14) TranslateWindowPositionToSubstancePositionFn
- 15) SubstanceDefaultWidth
- 16) SubstanceDefaultHeight
- 17) SubstanceLinkAnchorModesSupported

These fields are defined as follows:

1. SubstanceName: The atom that is the name of this substance type. SubstanceNames must be unique among the substance types, though they may overlap with card TypeNames. The convention is that SubstanceNames are all in caps. This is to set them apart from card TypeNames which by convention have only their first letter capitalized.

2 Thru 14. Functions: All of the functions are identical to the corresponding functions in the NoteCardType record structure. Note the (arbitrary) use of "create" instead of "make" in the name of the CreateSubstanceFn.

15 Thru 17. Parameters: The parameters are identical to the corresponding parameters in the NoteCardType data structure. There are no parameters for the LinkDisplayMode and the

DisplayInMenuFlg because these two parameters are not inherited. They must be specified separately for each card type.

B.4. Adding a New NoteCardType or SubstanceType to the System

The functions `NCP.CreateCardType` and `NCP.CreateSubstanceType` can be used to add new Types to the system.

`NCP.CreateCardType` takes 5 arguments: the `TypeName`, its `SuperType`, its `SubstanceType`, a functions list, and a parameters list. The functions list is an ASSOC list where the CAR of each sub-list is one of the function field names given above (e.g., `EditCardFn`, `MakeCardFn`, etc.). The CDR of the sublist should contain the name of the required function. Any function field name for which there is no entry will be set to NIL and will thus be inherited. The parameters list is analogous to the functions list, except that it applies to the parameter field names (i.e., `LinkDisplayMode`, `CardDefaultWidth`, `CardDefaultHeight`, and `CardLinkAnchorModesSupported`).

`NC.CreateSubstanceType` takes 3 arguments: the `SubstanceName`, a functions list, and a parameters list. The functions and parameters list are analogous to those for `NCP.CreateCardType` except that all of the function and parameter fields specified above MUST have an entry in the ASSOC lists.

Both `NCP.CreateCardType` and `NC.CreateSubstanceType` will overwrite existing types (`NoteCard` and `Substance`, respectively) of the same name.

B.5. Example: Defining the ProtectedText NoteCardType

The following is an example of defining a new card type called the `ProtectedText` card. The card type is created by specifying new `MakeCardFn` and `EditCardFn` functions. All other functions are inherited from from the super-type, i.e., the `Text` card. All of the parameters are specified directly for this card.

- The function that creates the new `ProtectedText` card type:

```
(NC.AddProtectedTextCardType
(LAMBDA NIL (* fgh: "26-Mar-85 15:48")
(* * Create the ProtectedText card type)
(NCP.CreateCardType (QUOTE ProtectedText)
(QUOTE Text)
(QUOTE TEXT)
(QUOTE ((MakeCardFn NC.MakeProtectedTextCard)
```

```
(SETQ Password (NCP.CardProp ID (QUOTE Password)))
(COND
  ((EQUAL Password (NC.GetPassword Window))
    (* Password is okay.
      Call the EditCardFn of my super-type)
    (SETQ Result (APPLY*
      (NCP.CardTypeInheritedField
      (NCP.CardTypeSuper (QUOTE ProtectedText))
      (QUOTE EditCardFn))
      ID Substance ExactRegion)))
  (T (* Password is bad. Express condolences)
    (NCP.PrintMsg Window T "Sorry." (CHARACTER
      13)
      "You do not know the password!!"
      (CHARACTER 13)
      "Bye."
      (CHARACTER 13))
    (DISMISS 2000)))
  (* * Close the window you created.
    The super-types EdityCardFn will
    have created another window.)
  (CLOSEW Window)
  (RETURN Result))))
```

- A utility used by the MakeCardFn and the EditCardFn:

```
(NC.GetPassword
(LAMBDA (Window) (* fgh: "26-Mar-85 15:50")
  (* * Get a password from the user.
    Window is the main window for the card in question)
  (NCP.AskUser "What is the password for this card?" " -- "
    NIL T Window)))
```

```

(EditCardFn NC.EditProtectedTextCard)))
(QUOTE ((LinkDisplayMode (T NIL NIL))
(CardDefaultHeight 300)
(CardDefaultWidth 400)
(CardLinkAnchorModesSupported T)
(CardDisplayInMenuFlg T))))))

```

- The MakeCardFn for the ProtectedText card type:

```

(NC.MakeProtectedTextCard
(LAMBDA (ID Title NoDisplayFlg) (* fgh: "26-Mar-85 15:23")
(* * Make a protected Text card
by calling the make card fn for a Text card
and then attaching a password to the card)
(PROG (Window WindowOrID)
(* * Create the Text card)
(SETQ WindowOrID (APPLY*
(NCP.CardTypeFn (NCP.CardTypeSuper
(QUOTE ProtectedText))
(QUOTE MakeCardFn))
ID Title NoDisplayFlg))
(* * Get the window for the card, if there is one)
(SETQ Window (WINDOWP WindowOrID))
(* * Get the password from the user
and add it to the cards prop list)
(NCP.CardProp ID (QUOTE Password)
(NC.GetPassword Window))
(* * Return whatever the super-type's MakeCardFn returned)
(RETURN WindowOrID))))))

```

- The EditCardFn for the ProtectedText card type:

```

(NC.EditProtectedTextCard
(LAMBDA (ID Substance Region/Position)
(* fgh: "26-Mar-85 17:21")
(* * Edit a Protected Text card, asking for the password first.)
(PROG (ExactRegion Window Password Result)
(* * Open a window for this card)
(SETQ ExactRegion (NC.DetermineDisplayRegion ID
Region/Position))
(SETQ Window (CREATEW ExactRegion))
(* * Get this card's password from the prop list)

```

Appendix C: An Introduction to the Internal Organization of Notefiles

This document provides a brief explanation of the manner in which notefiles are organized. The hope is that it will serve as a quick reference for questions about notefiles as well as a prerequisite for competent use of the Notefile Inspector facility. (See the document entitled "Inspecting and Repairing Notefiles.")

Included in what follows are descriptions of the notefile index, card parts, redundant storage of links, and the notion of an inconsistent notefile.

C.1. High level notefile structure: index and data areas.

A notefile consists of two parts, the index and the data area. Each card in the notefile has an entry in the index. An index entry consists of a status field and 4 pointers. The status field specifies whether the index entry is free or occupied by an active or deleted card. The four pointers in the index entry point into the data area to each of the 4 parts of a card: substance, title, links and property list. Whenever you change, say, the title of a card, the new title is written to the end of the data area and the index entry title pointer for that card is updated to point to the new location in the data area. Thus, in general, a notefile's data area grows every time any part of any card is changed. (But see Section 6 below on notefile checkpointing.) To throw away the old versions of card parts, it is necessary to compact the notefile.

C.2. Card IDs.

Every card in the notefile has a unique ID, e.g. NC00023. The top level fileboxes; **Contents**, **Orphans**, and **To Be Filed** have IDs NC00001, NC00002, and NC00003, respectively. These boxes are furnished by the system in all new notefiles and cannot be deleted.

The IDs from NC00004 through NC00020 are unusable, thus user-created cards start with ID NC00021. Currently, an old ID is never reused, even if its card is deleted and the notefile compacted. Thus, when a notefile is opened, the message "Processing item number 50 of 113" doesn't mean that the notefile has 113 cards. Rather, the notefile has used up 113 card IDs. The actual number of active cards could be far less if many cards have been deleted.

C.3. Card parts.

Of the four parts of a card, the title and property list are simplest. The title is simply a string while the property list is a list of attribute value pairs. Currently the only property maintained automatically by the system is "Updates" with value equal to a list of dates on which the card was updated. The list is ordered chronologically moving backwards in time from the front of the list to the back. [Though

"Updates" is the only system-maintained property on every card, NoteCards also displays the ID and card parts dates in the EditPropList window.)

The substance of the card is simply its contents. Thus a text card's substance is a text stream, a browser card's is its graph, etc. These are stored on the file in a manner appropriate to the substance type. Thus, for example, a text substance on the notefile looks like the way TEdit writes out text streams (text followed by "looks" information).

C.4. Links on the notefile.

The links of a card are divided into three groups: **to links**, **from links**, and **global links**, where the global links form a subset of the to links. The to links of a card are those links pointing from this card to some other card. The from links are those links pointing from another card to this one. Finally, the global links are those to links that are global, that is, they point from this card as a whole to another card. (Global links are the ones that aren't anchored in the source card's substance. Source links are an example.)

The confusing thing about links out on the notefile is that they are stored in several places. All links are stored as to links with their source card and as from links with their destination card. Furthermore, links that are not global are also stored within the substance of their source card inside of a link icon. Links that are global are also stored on the global links list of their source card. Thus, all links are stored in three places: as a to link on the source card, as a from link on the destination card and either in the substance of the source card or on its global links list. If these three records of a link don't agree for some reason, then we say that the notefile is **inconsistent** and needs its links rebuilt.

C.5. Inconsistent notefiles and links rebuilding.

You'll know that your notefile's links are inconsistent if you discover link icons displaying the string "DELETED" or "FREE." This usually means that the card at one end of a link was deleted, but somehow the links of the card at the other end were not updated. Such link icons cause NoteCards to break when you try to follow them.

The best way to fix an inconsistent notefile is to use the Notefile inspector facility accessible via **Inspect&Repair** from the Notefile Ops menu. The inspector's third phase rebuilds the links in a notefile as follows. First it removes all to and from links for every card. Then it reads the substances for each card and recreates to links and from links by looking at the links found inside the link icons in the card's substance.

In addition, the links rebuilder phase of the notefile inspector can rebuild filebox contents from cards pointed to by the filebox, and the set of all link types from the list of all links. Finally, it can rebuild global links for cards.

C.6. Notefile checkpointing.

When Notecards is in progress on an open notefile, the index area is cached in core. When a card part changes, the appropriate pointer in the in-core index is updated and the new version of the card part written out to the end of the file. There is a pointer called the **checkpoint pointer** pointing to that location in the data area that was the end of the notefile the last time it was closed or checkpointed. Thus, although the notefile is continually being appended to, the portion that dates from the last close or checkpoint is unaffected.

As described in the 1.2k release notes, checkpointing a notefile (via `CheckpointSession`) writes out the cached index to the notefile and moves up the checkpoint pointer. But first, any cards visible on the screen and having been changed since the last card update, are saved to the notefile. Thus, the state of a checkpointed notefile should be consistent.

This suggests that a notefile can be backed up to the last checkpoint or close discarding any interim work. In fact, the `AbortSession` command does just that. `AbortSession` causes the in-core index to be destroyed and the file to be truncated at the checkpoint pointer. Thus the notefile's state reverts to the time of the last checkpoint or close. (Actually, that's not quite true - changes to a card's region are written directly to the notefile. Thus card reshaping cannot be undone by `AbortSession`.)

C.7. Esoterica.

There are two other aspects of a card not included in the description of card parts above. These are a card's type and its region, both of which are stored with the substance. Thus whenever a card's substance part is written to the notefile, its type and region are written as well. Note that although a card's display region is recorded in the notefile, NoteCards actually uses only the region's extent when bringing up the card. The last position of the card is ignored.

Appendix D: Inspecting and Repairing Notefiles

This appendix describes the Notefile inspector facility available via the **Inspect&Repair** option on the **Notefile Ops** menu in NoteCards Release 1.2k.

The old **Repair Notefile** facility rebuilt the links in a Notefile from the contents of card substances. This was used whenever a notefile was thought to have inconsistent links. The problem was that notefiles with inconsistent links often had other problems that caused **Repair** to break. Thus the motivation for developing the notefile inspector documented here was to verify a notefile's readability before invoking the link rebuilder. As it turns out, this inspector is useful generally for checking the health of a notefile, deleting cards and backing up other cards (or more precisely, card parts) to previous versions. Thus, you may want to use the inspector even if your notefile is healthy and doesn't need its links rebuilt. [Note that the inspector can also be entered from **OpenNotefile** if NoteCards finds work after the last checkpoint. See Section D.4 below for details.]

The notefile inspector has three separate phases: reading the notefile's data area searching for healthy card parts, allowing the user to make modifications, and rebuilding the links. The process can be aborted after phase 1 or 2 if desired. This document describes each of the three phases in turn and concludes with tips, strategies and pitfalls to watch for.

Using the notefile inspector requires some knowledge of the internal organization of a notefile. It is recommended that you read Appendix C: An Introduction to Notefile Organization before continuing.

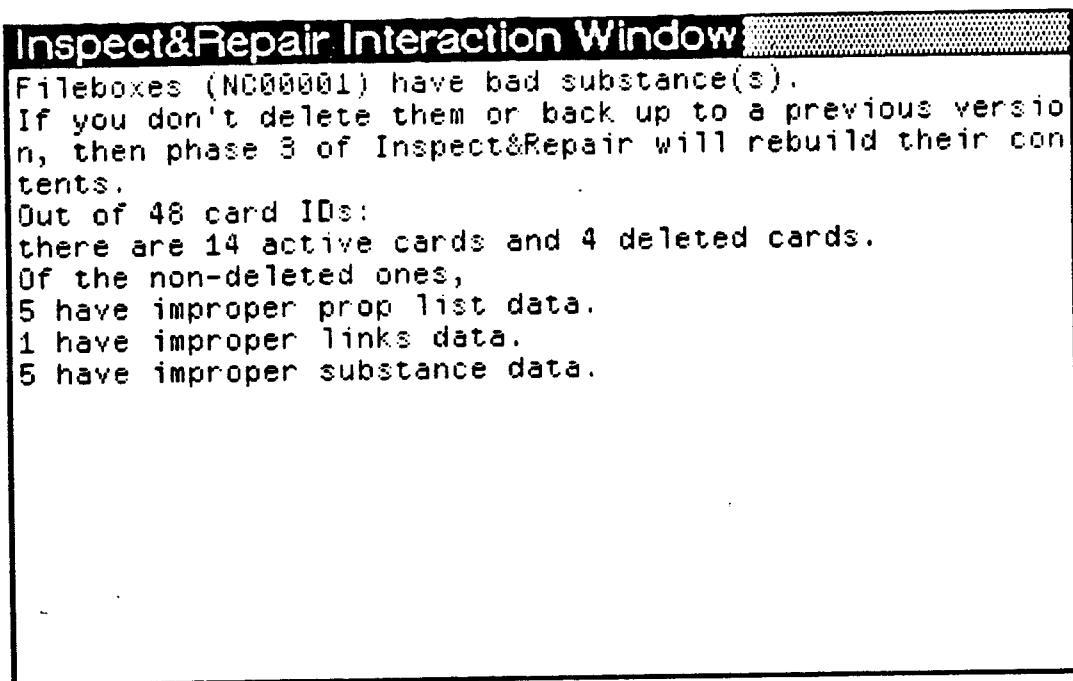
D.1. Running the Notefile Inspector: Phase 1: Scouring the Data Area

To start the inspector, first be sure that there is no open notefile. Then select the item **Inspect&Repair** from the **NoteFile Ops** menu. There is one option available at this level by "pulling to the side" called **ReadSubstances**. This ensures that substances of all cards pronounced valid by the inspector are readable. If this option is not invoked, then a check is still run on the length of the substance, but not on its contents. Unfortunately, the **ReadSubstances** option requires MUCH more work by phase 1. It is recommended that you only use this option if Phase 3 (link rebuilding) breaks with some error like "Bad Piece Tbl" from TEdit. In that case, up-arrow out of the break and start the **Inspect&Repair** process over again, this time using the slower but more comprehensive **ReadSubstances** option.

Selecting **Inspect&Repair** will invoke phase 1 of the inspector, wherein the data area of the notefile is scoured for valid card parts. A record of all such parts is kept and statistics printed out at the end. You'll be asked to position the window in which those statistics as well as later inspector

communications will be printed. (Note that closing this window will abort the Inspect&Repair process if you confirm.) You can monitor the progress of phase 1 by watching the prompt window. It will be printing messages like "Processing byte xxxxx of yyyy."

When phase 1 has completed and you've positioned the interaction window, statistics on your notefile will be provided. You'll be told the total number of card IDs used and the number of those that are currently associated with active and deleted cards. (The rest should be free and will never be reused.) If all seems well with the world, the next line will read "All non-deleted cards look okay." If not, there will be various messages outlining the problems. (See Figure D.1.)



```

Inspect&Repair Interaction Window
Fileboxes (NC000001) have bad substance(s).
If you don't delete them or back up to a previous versio
n, then phase 3 of Inspect&Repair will rebuild their con
tents.
Out of 48 card IDs:
there are 14 active cards and 4 deleted cards.
Of the non-deleted ones,
5 have improper prop list data.
1 have improper links data.
5 have improper substance data.

```

Figure D.1: Snapshot of a Sample Interaction Window

Note that a filebox has bad substance and that a special message is printed on its behalf. In general, if you don't wish to delete or back up such fileboxes to previous versions, then phase 3 will rebuild them. In this case, however, the filebox is the top level Contents box which can't be deleted. (See Section D.3.)

If there are cards having user-defined types whose type definition code has not been loaded, then you'll get a message to that effect, something like "<n> cards have unknown card types (FOO BAR)." At this point you should load the lisp files containing the definitions of the unknown card types. If not, then these cards will show up with bad substance in phase 2. If you do load the appropriate files, then run **Recheck Bad Cards** to get the bad cards list recomputed before inspecting any cards.

If you have a card for which no substance versions could be read, then you'll also get unknown card type messages for it (reading something like "<n> cards have unknown card types (NIL)"). This is because the inspector couldn't find a card type on the notefile for that card.

A menu of options appears attached to the upper right corner of the interaction window. (See Figure D.2.) The particular options you get in that menu depend on the state of your notefile and are described below. The last three options appear in all cases. The other two may or may not be present in the menu you get. In any case, you should select one of the options before attempting any other NoteCards-related work.

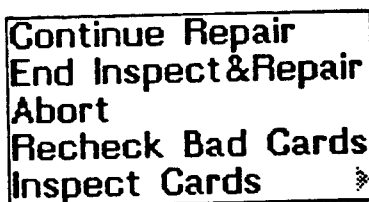


Figure D.2: The Phase 1 Options Menu.

Continue Repair: This option is only available if the notefile is in fairly good health (i.e. okay except for fileboxes to rebuild or global links to rebuild - see Section D.3). Selecting it causes Inspect&Repair to move to phase 3 and rebuild your notefile's links.

End Inspect&Repair: This option is only available if it seems that you don't need to continue to the link rebuilding phase. You will not get this option if you've deleted any cards, or generally if there are problems with the notefile. Choosing this option causes the Inspect&Repair process to end gracefully (via a normal checkpoint and close notefile), thus skipping phase 3, rebuilding links.

Abort: Choosing this option aborts the Inspect&Repair process entirely, throwing away any changes you might have made (such as card deletions or back ups.) It requires your confirmation in the prompt window.

Recheck Bad Cards: Recompute the bad cards list by running the last part of phase 1 again. This is useful if you've loaded files containing definitions of previously undefined card types.

Inspect Cards: This brings up a menu of active card IDs with which you can inspect, delete, or back up particular cards. There is a "pull-across" menu item called **Include Deleted Cards**, which if selected will include card IDs for deleted cards as well as active ones. Using this option, one can undelete deleted cards and restore some previous version.

D.2. Running the Notefile Inspector: Phase 2: Your Chance to Tinker

After selecting **Inspect Cards** in the interaction window's attached menu, a menu containing Notecards IDs will pop up and be attached to the interaction window's lower left corner. It will contain IDs for all active cards and possibly deleted cards as well if you selected the submenu item **Include Deleted Cards** described above. The menu can hold some 200 card IDs. If your notefile has more than that, then the menu will be composed of several pages each containing around 200 IDs. Rapid switching between pages is possible. Figure D.3 below shows a sample card inspector menu.

Attached to the upper right corner of the cards inspector menu is a menu containing at least the three options: **Abort**, **Done** and **Search**. If the menu has multiple pages (there are more than 200 active cards in the notefile), then the attached menu will also include the items **Next Page**, **Previous Page**, and **First Page**. Selecting these causes the current menu to be exchanged for either the next menu, previous menu, or first menu, respectively.

Clicking **Abort** causes the entire **Inspect&Repair** process to quit, throwing away any changes you've made. It requires your confirmation in the prompt window. (This is equivalent to choosing **Abort** from the inspector window as described in Section D.1.)

Choosing **Done** from this attached menu indicates that you're done tinkering with card parts and wish to return to the main interaction window. This causes the card inspector menu to close and the phase 1 process outlined in Section 1 to be performed again (leaving you looking at something like Figure 1). Note, however, that the scanning of the data area is not repeated (it takes way too long). Rather, your changes are made to the in-core index array and the statistics on bad cards are recomputed. This cycle of compute statistics (phase 1) followed by inspect and tinker (phase 2) can be repeated as often as desired. Eventually, you must either abort, end the **Inspect&Repair** via **End Inspect&Repair**, or continue to phase 3 via **Continue Repair**.

Choosing **Search** from the attached menu allows you to find the notecard IDs matching a given string. This works much like the **Search** card in Notecards. That is, you specify a string and the list of Notecard IDs whose titles contain that string is printed out. This is handy if you want to undelete a card, but only know its title. Note that the search is case sensitive.

D.2.1 The Card Inspector Menu

In the card inspector menu, those IDs corresponding to deleted cards have a line drawn through them. Those having some sort of problem appear shaded. In addition, an upper-case letter suffix is attached to such IDs indicating the problem. For example, a shaded menu item NC00023SL indicates that ID NC00023 has bad substance and bad links. The letter codes are S, L, P, and T indicating bad substance, links, property list, and title, respectively. If such a letter code appears in lower case, then the indication is that the current version of that card part is beyond the last checkpoint pointer. For

example, NC00023t indicates that NC00023's current title was changed since the last checkpoint. (There may have been a crash, for example, thus preventing the notefile from closing normally.)

Figure D.3 shows a sample card inspector for a healthy notefile including deleted cards. Sometimes deleted cards (e.g. NC00042) show up shaded with LSPTU suffixes. This is because they were deleted before any data about the card was written to the notefile. Thus there are no valid card parts to back up to for those cards. But often, deleted cards can be backed up to previous versions. (See Figure D.5 and discussion below.)

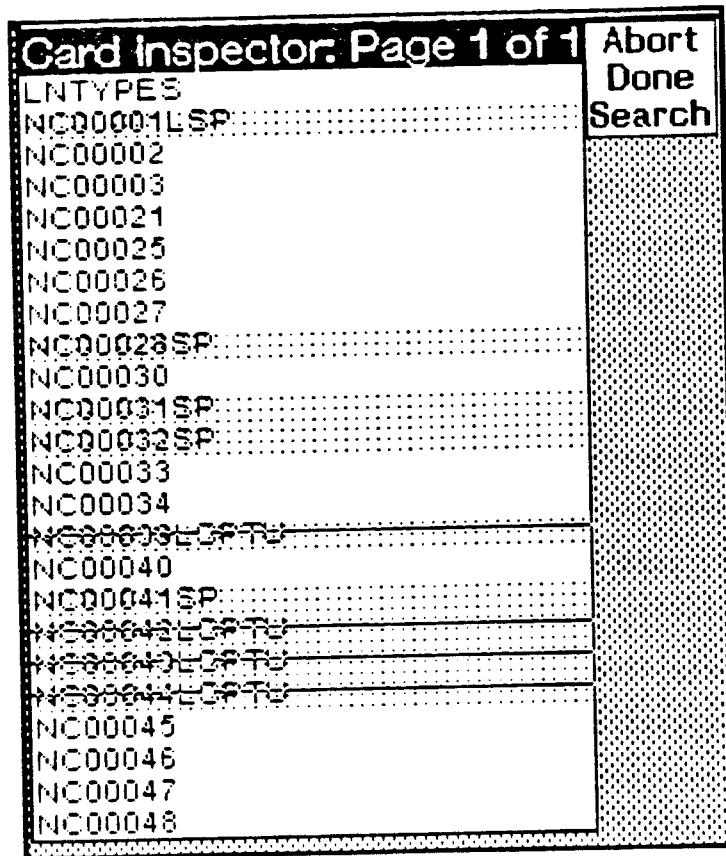


Figure D.3: A Card Inspector Menu.

In addition to menu entries for each card ID, there is also one entry labeled LNTYPES allowing you to inspect (and possibly back up to a previous version) the Link Types for this notefile. Note that if the inspector shows no entry in the card inspector menu for some ID, it is because that card has been deleted. If you've asked to show deleted cards and it still doesn't appear, it's because the notefile has been compacted since that card was deleted.

If you button an ID in the card inspector menu, then a popup menu allows up to two choices **Inspect** and/or **Delete**. If the card is currently deleted (has a line drawn through it), then the **Delete** option is replaced by **Undelete**. Certain card IDs cannot be deleted and thus their popup menus only contain

the Inspect option. These are the top level file boxes NC00001, NC00002, NC00003. The link types menu entry LNTYPES also does not allow deletion.

Choosing Delete or Undelete from this popup menu causes the card to be deleted or undeleted, respectively and the line through the menu item either drawn or undrawn. Note, however, that this action (and all others) can be undone by choosing ABORT from either the card inspector menu or the interaction window menu.

Choosing Inspect from the popup menu for a card ID entry brings up a card parts inspector for that card.

D.2.2 The Card Parts Inspector

Figures D.4 and D.5 below show examples of card parts inspectors. A card parts inspector is composed of four attached menus arranged vertically and one attached operations menu atop the stack.

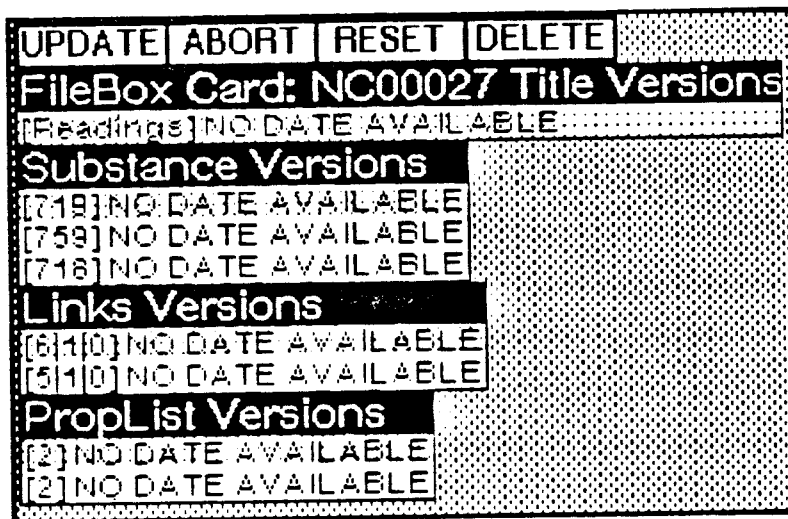


Figure D.4: A Card Parts Inspector

In Figure D.4, the four menus contain entries for every valid version of card parts for the card with ID NC00027. The top menu is for version of titles and below that are menus for versions of the card's substance, links, and prop list. For example, the Substance submenu contains entries for three versions of the substance of this card. The current version of each card part is shaded. Each menu entry gives the date that that version was written if available or the string "NO DATE AVAILABLE" if there is no date on the notefile. (The latter is the case for old notefiles prior to the time the write dates of card parts were recorded.)

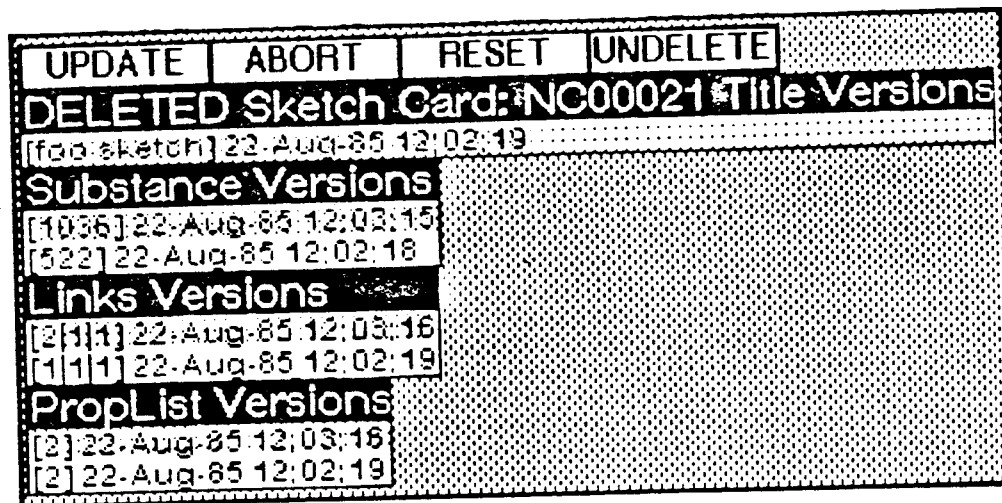


Figure D.5: A Card Parts Inspector for a Deleted Card.

Figure D.5 shows a card parts inspector for a deleted sketch card and thus allows the UNDELETE option (described below). By clicking UNDELETE, this card could be backed up to its last version. Note also that this card being more recent than the one in Figure D.4 means that valid dates of last update exist for each of the card part versions.

If the current version of the card part is bad, then the menu entry will be a string so indicating, for example, "BADSUBSTANCE."

The title of the top menu includes the card's type and ID. In addition, each menu item contains a bit of information, in square brackets, before the date. In the title versions menu, this information is the first few characters of the title. In the substance versions menu, it is the number of bytes in the substance. In the links versions menu, it is a triple of numbers giving the number of to links, from links, and global links for this card. Finally, the proplist versions menu includes the number of entries on the property list for this card (i.e. twice the number of attribute-value pairs).

Atop the stack of menus is an attached menu of operations, described below.

ABORT: This aborts this card parts inspector, throwing away any changes made.

UPDATE: This closes the card parts inspector, incorporating any changes (backing up to previous versions of card parts) you might have done.

DELETE: This option closes the card parts inspector and deletes the card. (Again, this can be undone by choosing ABORT from the card inspector menu.)

UNDELETE: For cards that have been deleted, this option appears instead of DELETE. Choosing it causes the card to be undeleted.

RESET: This causes the selections in the submenus to be restored to the values they had when the card parts inspector was first brought up. (Equivalent to doing **ABORT** and then inspecting this ID again from the card inspector menu.)

Note that cards that can't be deleted (like the top level fileboxes) don't have the **DELETE** option on their card parts inspector.

Buttoning an entry in a submenu of a card parts inspector pops up a short menu unless the entry is for a bad version (e.g. "BADSUBSTANCE"). This menu contains at least the entry **Inspect** and possibly **Change Selection**, if the selected entry is not the same as the current one (i.e. not shaded).

Choosing **Inspect** allows further inspection of the details of the selected card part version. (See Figure D.6 below.) For example, inspecting a title version brings up the Interlisp inspector on a record containing the title, date and card ID. Similarly, inspecting a links or prop list version brings up the Interlisp inspector on a record containing the lists of links (for to links, from links and global links) or the prop list. Note that if you wish to continue inspecting a links version down to the single link level, choose to inspect the link as a **NOTECARDLINK**. This is somewhat more communicative about record field names. Note also that changing values out in the Interlisp inspectors has no affect on the notefile and is ignored.

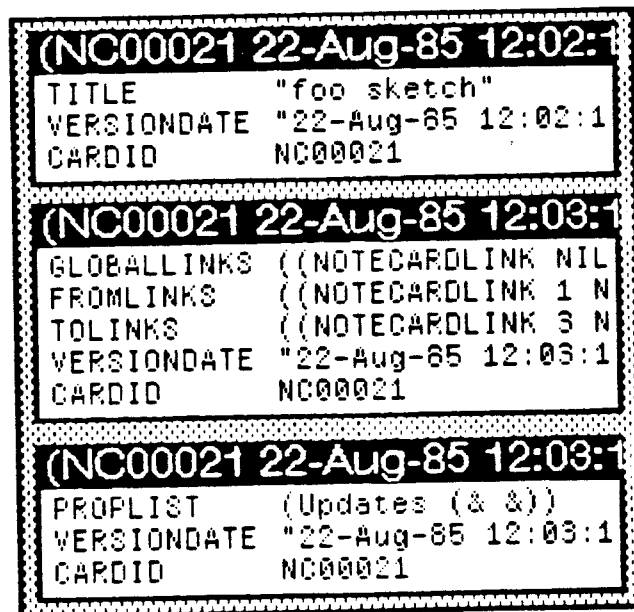


Figure D.6: Card part version inspectors for title, links and prop list.

All substance versions for cards having substance types **TEXT**, **SKETCH** and **GRAPH** can currently be inspected. (This includes all cards except those having user-defined substance types, like the **NCFfile** card.) Inspecting a card's substance version will bring up a window showing a copy of the

substance. (Note that changes to this copy have no effect on the notefile.) Any links in the substance of the card will show up as bracketed strings describing the link.

Choosing **Change Selection** from the card part version popup menu causes the current selection to be changed to the selected one. This changes a card part's current version to another legal one. (This change can be undone by resetting or aborting the card parts inspector as well as by later aborting the card inspector or interaction window.)

D.3. Running the Notefile Inspector: Phase 3: Rebuilding Links

To complete the **Inspect&Repair** process, select the **Continue Repair** option from the interaction window menu. This invokes phase 3, the links rebuilder. Normally, this simply rebuilds links from card substances. In certain circumstances, it may do extra work as well. If your link types list is bad, and you didn't back it up to a previous version, then phase 3 will rebuild it. If there are fileboxes with bad substances that you haven't either deleted or backed up to previous versions, then phase 3 will rebuild them. Finally, if there are cards with bad links that you haven't backed up or deleted, then phase 3 will rebuild those links as well. (It rebuilds ALL to links and from links anyway. For those cards, it will rebuild global links as well.) Details on the operation of the links rebuilder follow.

First the links rebuilder removes all the to and from links for every card. Then it reads the substances for each card and recreates to links and from links by looking at the links found inside the link icons in the card's substance.

The link rebuilder is also able to rebuild bad filebox substances. It does this by looking for all cards in the notefile with from links from the bad box and creating a new substance for the box containing only links to those cards. This process loses any text that the box might have contained as well as scrapping the original ordering of links. Nonetheless, in some cases this may be preferred to backing up the substance to a previous version or to deleting the box altogether.

The links rebuilder can rebuild the notefile's list of link types in a similar manner. That is, it records the set of link types seen on valid links and replaces the old links types with the new set. Note that this throws away any link types for which there are no links in the notefile.

Finally, the links rebuilder can rebuild bad global links for a card. It does this by looking for any cards with from links from the bad card that are global. This assumes that the card at the destination end has good links. Thus, if the cards at both ends of a global link have unreadable links, then there is no way to recover that link.

Note that the links rebuilder automatically files any unfiled cards in the **ToBeFiled** box. A message is printed in that case.

D.4. Tips and Hints for Using Inspect&Repair

This section contains a list of strategies and tips for using Inspect&Repair. For the most part, they are ordered from the useful and obvious to the esoteric. Several of these are implicit in the first four sections of the document, but are repeated here for emphasis and completeness.

When in doubt, abort! All your changes will be lost, but then if you're uncertain about what's happened this is the safe course. Often, in fact, you may simply want to check the health of your notefile and abort without tinkering.

Fixing versus tinkering. There are two main ways to use the inspector, either for fixing a broken notefile, or tinkering with a healthy one. The latter case occurs when you wish to recover some card that you inadvertently deleted. Or back up a card that you inadvertently changed to a previous version.

Inspect&Repair after a crash. If you try to open a notefile and you get the infamous "Work was done since last checkpoint..." message, then you're given the chance to run Inspect&Repair on the notefile. This is a good idea whenever you wish to recover that post-checkpoint work. The inspector will integrate the changes you made since that checkpoint. [You do have to continue through phase 3, however.]

Compacting. Old versions of card parts have always been stored in notefiles, but up till now have been inaccessible. Thus, there was little reason not to compact your notefile often. Now there is a tradeoff between the need to save space by compacting versus the need to be able to back up using Inspect&Repair. Probably the safest course is to keep a backed up copy of the pre-compacted notefile around until you have confidence that the compacted one is healthy and that you have no need for previous versions of any of its cards.

Inspect&Repair can't run when notefile is open. This means that if you are working in your notefile and notice a card you'd like to inspect a previous version of, you must record the card's title and close the notefile. Then, run Inspect&Repair, find the card's ID in the inspector menu using the Search facility, and tinker with it as desired.

Fixing enough problems to allow phase 3 to run. You can't run phase 3 unless Inspect&Repair thinks your notefile is above a certain threshold of health. There are certain problems it can handle (e.g. bad filebox substances, see Section 3), and others that it can't (e.g. bad substance for non-filebox). You have to decide either to fix these sorts of problems yourself in phase 2, let phase 3 attempt to rebuild them, or just abort the whole thing (always an option).

Sometimes these decisions can be tricky. For example, suppose a filebox's substance is bad. Call it **BadBox**. Should you (a) delete **BadBox** altogether, (b) back its substance up to a previous version, or (c) allow phase 3 to rebuild it by looking for from links in other cards from **BadBox**? Option (c) may not be advisable if there was important text in **BadBox** or if the order of cards in **BadBox** was important. On the other hand, option (b) may be of little use if the last good version is too out of date (or if there is no good version at all).

Index

In this index, menu commands are listed in italics.

<i>Abort Session</i> (NoteFile Op)	70-71
<i>Add Edge</i>	81
<i>Add Label</i>	81
<i>Add Link</i> (graph editing menu)	40
<i>Add Node</i> (graph editing menu)	40, 81
<i>Advanced Editing Menu</i>	34
AlphabetizedFileBoxChildren	74, 83
AnnoAccessible	27
ArrowHeadsInBrowsers	74, 77
<i>Assign Title</i>	12, 17, 75
<i>AttachBitmap?</i> (NoteCard Symbol)	21-22, 75
bitmap editor	58-60
break window	64, 65
<i>Bring Up Card/Box</i>	21
BringUpCardsAtPreviousPos	74
browser arrowheads	77
Browser card	2, 6-7, 23-24, 76-82
browser, editing manually	79-82
browser link dashing	76
browser roots	76
Browser Specs stylesheet	77
BrowserContents link type	20
<i>Bury</i> (Window Menu command)	30
cards	1,4-15
card types	4
card type menu	4
<i>Change Browser Specs</i>	79
<i>Change Display Mode</i>	21, 75
<i>Change Font</i>	39
<i>Change Link Type</i>	21
<i>Change Parameters</i> (NoteFile Op)	26-27, 73
<i>Checkpoint Session</i> (NoteFile Op)	70-71
close (using cursor)	31
<i>Close</i> (Window Menu command)	30
<i>Close and Save</i>	14, 17
<i>Close Cards</i>	29
<i>Close/Delete</i> (Main Menu command)	29
<i>Close NoteFile</i> (NoteFile Op)	26, 28
<i>Close w/o Saving</i>	14, 17
CloseCardsOffScreen	74
<i>Collect Children</i>	17
<i>Compact NoteFile</i> (NoteFile Op)	28, 71
<i>Copy NoteFile</i> (NoteFile Op)	28, 29, 71
copying graphs (into text cards)	82
copying notefiles	66
copying sketches (into text cards)	82
<i>Create</i> (Main Menu command)	4, 29
<i>Create Card & Node</i>	80
<i>Create Link & Edge</i>	80
<i>Create New NoteFile</i> (NoteFile Op)	25, 28
creating fileboxes	16
creating links	22
dates	83
DefaultCardType	26
DefaultFont	74

DefaultLinkIconAttachBitmap	74
DefaultLinkIconShowLinkType	74
DefaultLinkIconShowTitle	74
Delete Card	14
Delete Card & Node	80
Delete Cards	29
Delete FileBox	14, 17
Delete Link (graph editing menu)	40
Delete Link & Edge	80-81
Delete Node (graph editing menu)	40
Delete NoteFile (NoteFile Op)	25, 28
Delete Source	13
Designate Sources	13
DocBackPtr link type	20
Document card	3, 8-9
Edit Property List	11-12, 17, 83
editing text in a Sketch window	54
editing the browser	79-82
EnableBravoToTEditConversion	74
Expand (Window Menu command)	31
Expand Browser Node	79
File in FileBoxes	13
filebox menu	17
FileBoxes	2, 5, 13, 16-19
fileboxes, special	18-19
fileboxes and cards, suggested	19
FiledCard link type	20
FixedTopLevelMenu	26
FIX MENU	81
font selection	72
ForceFiling	27, 73
ForceSources	26, 73
ForceTitles	27, 73
Graph card	5, 40
graph card, copying	82
Graph Edit Menu (browser menu)	79, 79-82
graph editing menu	40-41
graph editor	40-41
Insert Link	13-14, 22, 82
Inspect&Repair NoteFile	28, 72, 117-127
label larger (graph editing menu)	41, 81
label smaller (graph editing menu)	41, 81
link icon display mode	75
link icons, moving	75
Link Index card	3, 8, 24
link insertion	82
link ordering	82, 83
link type menu	21-22
link types	20
link types, system reserved	20
link types, user specified	20
LinkDashingInBrowsers	27, 74, 76
LinkIconFont	74
LinkIndexBackPtr link type	20
links	1, 13, 20-24
links, managing	23
List NoteFiles (NoteFile Op)	28
ListContents link type	20
Main Menu	25-29

Maps	56-57
MarkersInFileBoxes	74
move (using cursor)	31
<i>Move</i> (Window Menu command)	30
<i>Move Node</i>	81
notecard menu	11-14
NoteCard Symbol (AttachBitmap?)	21-22
<i>NoteFile Ops</i>	25-29
notefiles	2
notefiles, organization of	114-116
<i>Open NoteFile</i> (NoteFile Op)	27
Orphans filebox	18
property list	3
programmer's interface	61-63, 83, 84-99
<i>Put Cards Here</i>	17
recovering from bugs	64-69
<i>Recompute Browser</i> (browser menu)	7, 78
<i>Reconnect Nodes</i> (browser menu)	78-79
<i>Redisplay</i> (Window Menu command)	30
<i>Relayout Graph</i> (browser menu)	78
<i>Remove Edge</i>	81
<i>Remove Node</i>	81
<i>Restart Editor</i> (text editor menu)	34
<i>Save in NoteFile</i>	14, 17
screen management	30-31
scrolling	14-15
Search card	3, 7
selecting cards	10-11
selecting text	32
<i>Shape</i> (Window Menu command)	30
ShortWindow.Menus	26, 30
<i>Show Box</i> (Main Menu command)	18, 29
<i>Show Links</i>	12, 17, 23, 82
<i>Shrink</i> (Window Menu command)	31
Sketch card	5, 42
Sketch editing menus	44-45
Sketch editor	5, 42-57
Sketch element selection	45
Sketch menu commands	45-54
Sketch mouse functions	43-44
sketches in text cards	55, 82
sketches, copying	55, 82
Source link type	3, 13
SpecialBrowserSpecs	27
<i>STOP</i> (graph editing menu)	41
structure editing	79-82
Stylesheets	72
SubBox link type	20
Table of Contents filebox	18
TEdit	32-39
Text card	4
text editor	32-39
text editor menu	34
<i>Title/Sources/FileBoxes</i>	12
To Be Filed filebox	18
types mechanism	61, 84-99
<i>Unconnect Nodes</i> (browser menu)	79
<i>Unfile from FileBoxes</i>	13
window menu	30

<-> <i>Border</i> (graph editing menu)	41
<-> <i>Directed</i> (graph editing menu)	41
<-> <i>Shade</i> (graph editing menu)	41, 81
<-> <i>Sides</i> (graph editing menu)	41